Bill Pay Application Design

Version 3.0

Author: Vivek Laxman Bhandarkar

Objective

Design Bill Pay Application

Please read through the instructions in full before starting this challenge. Design documentation / diagrams/ specifications should be uploaded to Git repo within 7 days and Mastercard Recruitment Team should be able to access it.

XYZ Corporation would like to create a new bill payment application to act as hub between customers and billers. XYZ Corporation will maintain customer account with balances in it, will maintain biller list along with bill data and will store all transaction data.

Bill Pay Application : Requirement overview and high level features

1. Registration : Provide a feature to “register” customer for paying bills. As part of registration capture email id only. Successful registration should create a “stored value account’ (wallet) associated with the registered customer. Option for customer to move funds to the wallet as part of registration.
2. Pay Biller : Pay your utility bills
   a) Fetch billers and bill from 3rd party external applications.
   b) Select a biller, fetch the bill and pay that using the wallet.
   c) Move funds from customer wallet to biller account.
3. Bulk/File based Bill Payment
   a) Process bill payments for N number of records received in a file. Multiple files can be received in a day. Design should support following use cases
   Requirement 1
   Design assuming XYZ provides a standard specification for Inbound bill payment file and customers (corporates, banks, service providers) adheres to this format when sending the bill pay file.
   Requirement 2
   Design should also support non standard format / adhoc format in future. The design should require minimal code changes to support new file formats introduced in future.

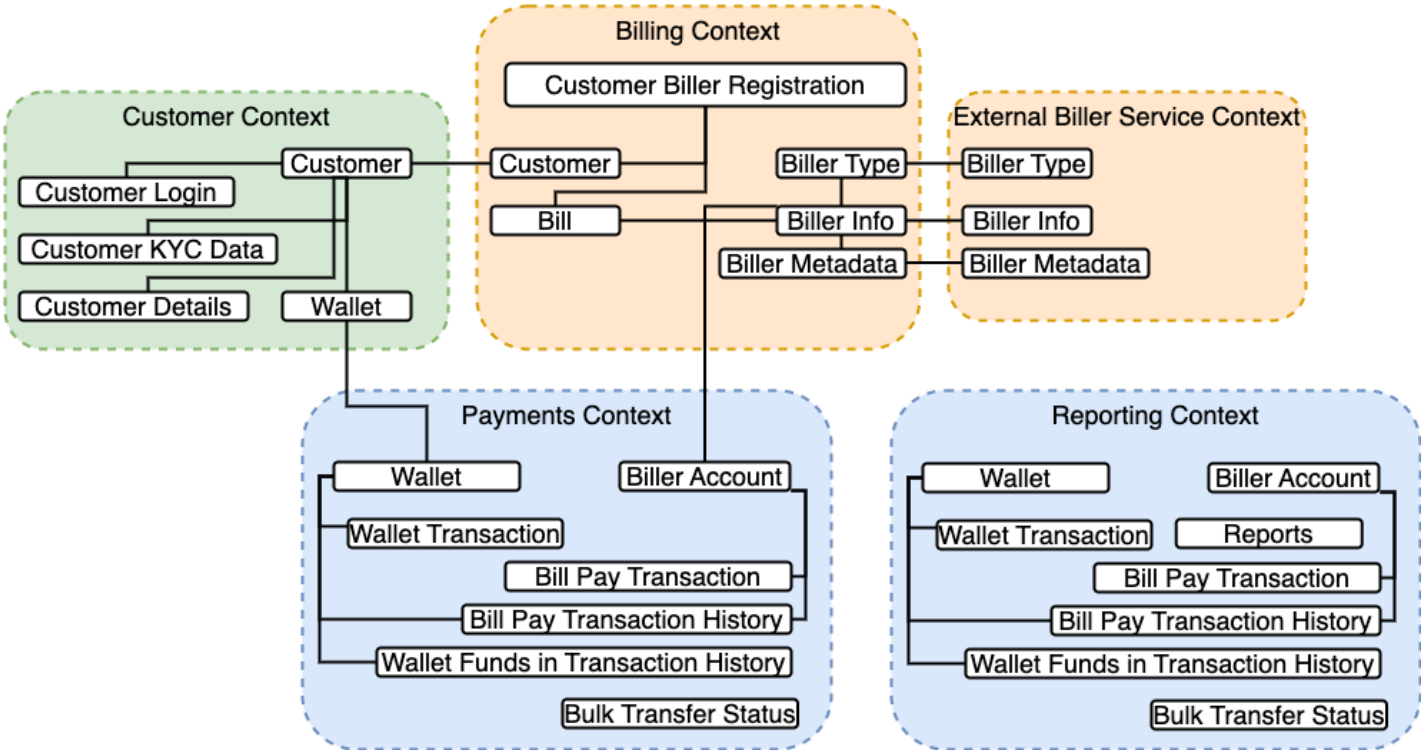
Deliverables Expected : Document detailed design / diagrams / specifications taking all below into consideration but not limited

- Cloud native mS design and cloud agnostic micro services implementation. Propose “design” by detailing asks from below points. No need to submit code.
- Bounded context diagram
- Clearly define below for all micro services
    - roles & responsibilities for each mS
    - tech stack (hosting env, dev technologies, database ..)
    - data model
    - specs for external and inter micro service APIs
    - testing strategy
    - Reporting
- Non functional requirements to be considered
    - Security
    - Logging & Monitoring
    - Auto scaling
    - CI CD processes and tech stack
    - Automation of test suites
    - Support PI/PCI Data. Application should be PCI complaint.

Proposed Design As Below:

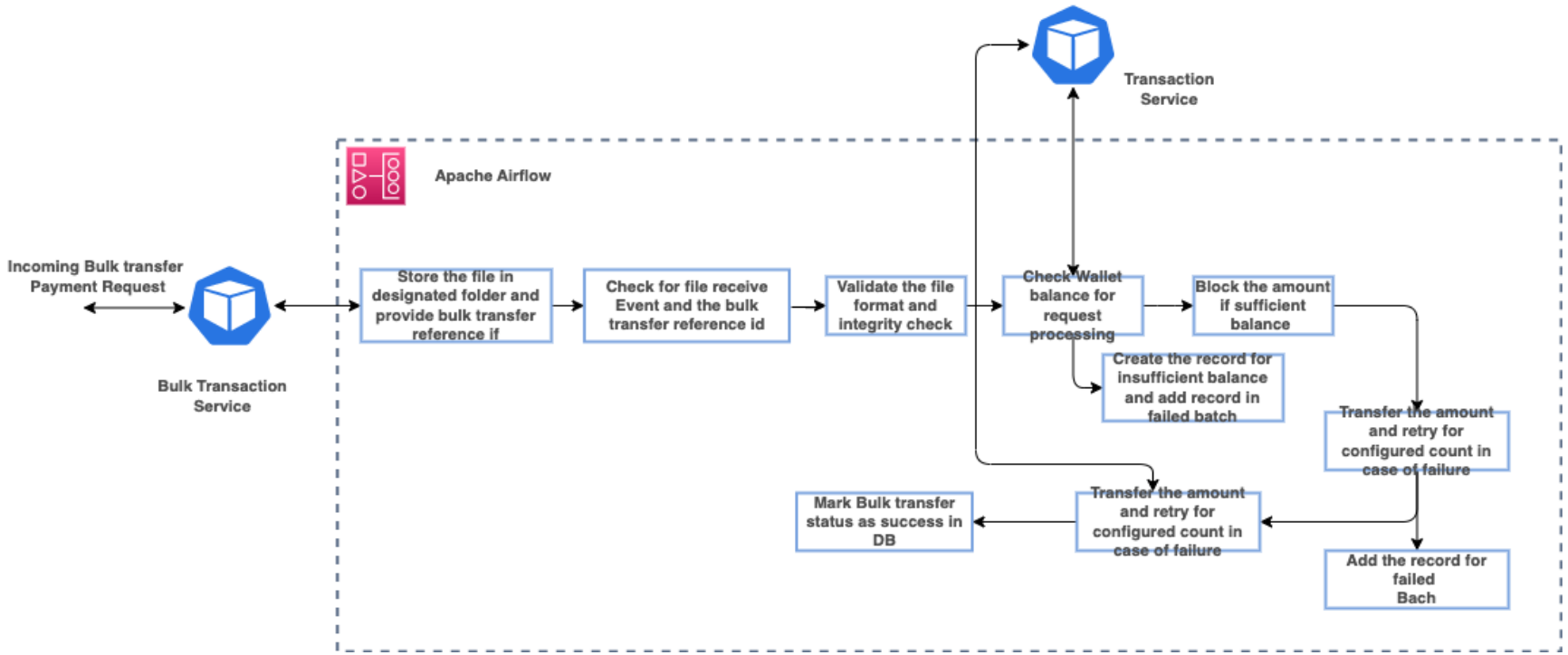
1. **Bounded context diagram**

   Bill Pay application as shown below is divided in Customer, Billing , Externer Billing, Transaction and Reporting Context. Reporting Context has similar entities as Transaction context. Data stored in Reporting Domain has higher data retention policy as per Country specific Data retention Policy compared to Transaction domain.



2. Responsibility of each microservices as below:
   - 2.1. Customer service: This service will handle the registration of new customers and the creation of their associated wallet accounts. All customer and associated Wallet related operations It will also provide the ability for customers to move funds into their wallets. Data associated will be stored in the Multi-region high-availability PostgreSQL server.
   - 2.2. Biller service: This service will manage the list of billers and their associated bills. Associated Biller data will be stored in Redis Cache and MongoDB.
   - 2.3. Payment service: Service will also provide the ability for customers to pay their bills using their wallet accounts and will also handle the processing of bulk payment files, including those in standard and non-standard formats. Associated transaction records will be store in PostgreSQL
   - 2.4. Bulk Payment Service : Bulk Payment service provides a mechanism to post bulk payment file to process payments listed in standard and non-standard format. On receiving the request service returns the Bulk transaction reference Id which can be

referred at any later time for status enquiry. Service internally uses Payment service to check balances and make payment. Apache Airflow is used to manage the Bulk Payment jobs. Detailed workflow as shown below:

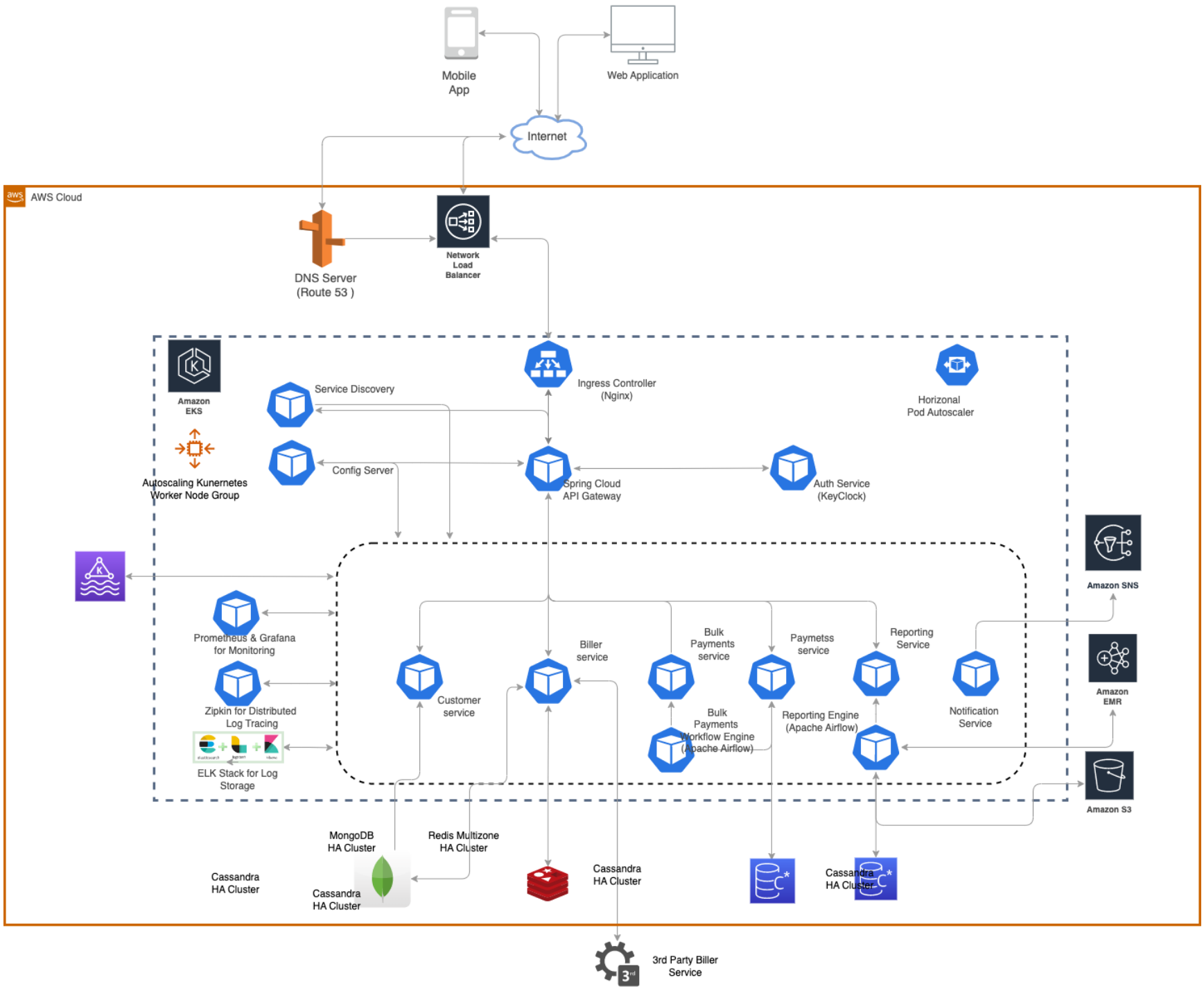

Transaction Service
Apache Airflow
Incoming Bulk transfer Payment Request
Bulk Transaction Service
Store the file in designated folder and provide bulk transfer reference if
Check for file receive Event and the bulk transfer reference id
Validate the file format and integrity check
Check Wallet balance for request processing
Block the amount if sufficient balance
Create the record for insufficient balance and add record in failed batch
Transfer the amount and retry for configured count in case of failure
Add the record for failed Bach
Transfer the amount and retry for configured count in case of failure
Mark Bulk transfer status as success in DB

2.5. Reporting service: This service will provide reporting capabilities, including the ability to track transactions and account balances. Monthly , Yearly or analytical reports can be triggered. Transaction records are stored in Cassandra DB. Apache Airflow is used for generating reports or triggering Analytical Tasks on AWS EMR. Generated reports are stored on S3 and send to intended audience via Notification service using Amazon Simple Notification Service (SNS)

2.6. Notification Service : Service will provide the notifications to customers on transaction , Login OTP , Wallet Monthly reports.

3. **Project Architecture**:

Application is Hosted on AWS Cloud infrastructure with Multi-Zone Availability setup to make sre application is Highly Available.
Microservices Architecture patterns are followed . Applications are developed using Spring Spring and Spring Cloud Framework ensuring Cloud Native Microservices Architecture Principles are followed. Circuit Breaker patterns are followed to ensure the entire application does not have a down-time impact in case of partial failures. Kafka is used for even streaming to send the events in case application state changes across different contexts.
AWS Route 53 DNS server is utilized to map domain name to AWS Network Load Balancer which offer High Network throughput, High Availability.
Nginx Ingress Controller is used to round all the incoming traffic to Spring Cloud API Gateway.
Spring Cloud API Gateway Authenticates and validates the token for all incoming request in conjunction with the Keycloak server.
Hoshicorp Vault or Kubernetes Secrets can be used for secret storage however it’s not depicted in below Architecture diagram.


Mobile App
Web Application
Internet
AWS Cloud
DNS Server (Route 53)
Network Load Balancer
Amazon EKS
Service Discovery
Ingress Controller (Nginx)
Horizonal Pod Autoscaler
Autoscaling Kunernetes Worker Node Group
Config Server
Spring Cloud API Gateway
Auth Service (KeyClock)
Amazon SNS
Prometheus & Grafana for Monitoring
Zipkin for Distributed Log Tracing
ELK Stack for Log Storage
Customer service
Biller service
Bulk Payments service
Paymetss service
Reporting Service
Bulk Payments Workflow Engine (Apache Airflow)
Reporting Engine (Apache Airflow)
Notification Service
Amazon EMR
Amazon S3
MongoDB HA Cluster
Redis Multizone HA Cluster
Cassandra HA Cluster
Cassandra HA Cluster
Cassandra HA Cluster
Cassandra HA Cluster
3rd Party Biller Service

4. **Tech stack (hosting env, dev technologies, database .**.

4.1. Hosting Environment

Tech Stack	Provider / Software	Overview
Cloud Provider	Amazon Web Services (AWS)	Amazon Web Services offers a broad set of global cloud-based products including compute, storage, databases, analytics, networking, mobile, developer tools, management tools, IoT, security, and enterprise applications: on-demand, available in seconds, with pay-as-you-go pricing.
Network Load Balancer	AWS Network Load Balance (NLB)	Elastic Load Balancing automatically distributes your incoming traffic across multiple targets, such as EC2 instances, containers, and IP addresses, in one or more Availability Zones.
Container Orchestration Framework	Kubernetes	Kubernetes is a portable, extensible, open source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available.
Container Framework	Docker	Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly.
Application Server	Apache Tomcat (10.1.0)	Apache Tomcat is a free and open-source implementation of the Jakarta Servlet, Jakarta Expression Language, and WebSocket technologies. It provides a "pure Java" HTTP web server environment in which Java code can also run.
AWS WAF		AWS WAF helps protects your website from common attack techniques like SQL injection and Cross-Site Scripting (XSS). In addition, you can create rules that can block or rate-limit traffic from specific user-agents, from specific IP addresses, or that contain particular request headers.

4.2. Databases and Caching Framework

Service	Database / Caching Framework	Version	Overview
Customer Service	Amazon MCS	Apache Cassandra 3.11	Amazon Managed Apache Cassandra Service (MCS), a scalable, highly available, and managed Apache Cassandra-compatible database service. Cassandra is utilized being cloud scale database and can offer High Availability and scalability at Cloud scale.
Bulk Payment Service			
Payment Service			
Biller Service	Amazon DocumentDB	MongoDB 4.0	Amazon DocumentDB is a managed proprietary NoSQL database service that supports document data structures, with some compatibility with MongoDB version 3.6 and version 4.0. MongoDB is used as preferred database as customer registration has dynamic schema based on Biller .
	Amazon Elasticache Redis	Redis 6.2	Amazon ElastiCache for Redis is a blazing fast in-memory data store that provides sub-millisecond latency to power internet-scale real-time applications. Biller Type, Biller Info and Biller Metadata are stored in Redis Cache for faster access and Are refreshed at regular intervals. In case of 3d party biller application failure redis can still ensure the Billing Service is up and running Without any downtime.

4.3. Auth Server

Keycloak (20.0.2) is an open source software product to allow single sign-on with Identity and Access Management aimed at modern applications and services.

4.4. Cloud Native Microservices Frameworks

Framework	Version	Overview
Java	11	Java is a high-level, class-based, object-oriented programming language.
Spring Boot	3.0.0	Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run".
Spring Cloud Service	v1.4.6	Client-side service discovery allows services to find and communicate with each other without

Registry		hard-coding the hostname and port.
Spring Cloud API Gateway	v4.0.0	This project provides a library for building an API Gateway on top of Spring WebFlux. Spring Cloud Gateway aims to provide a simple, yet effective way to route to APIs and provide cross cutting concerns to them such as: security, monitoring/metrics, and resiliency.
Spring Cloud Config Server		Spring Cloud Config provides server-side and client-side support for externalized configuration in a distributed system.
Zipkin		Zipkin is a distributed tracing system. It helps gather timing data needed to troubleshoot latency problems in service architectures.

4.5. Monitoring Softwares

Micrometer (1.10.2) : Micrometer provides a simple facade over the instrumentation clients for the most popular monitoring systems, allowing you to instrument your JVM-based application code without vendor lock-in. Think SLF4J, but for metrics.

Prometheus (2.40.7) : Prometheus is an open-source systems monitoring and alerting toolkit originally built at SoundCloud. Prometheus collects and stores its metrics as time series data, i.e. metrics information is stored with the timestamp at which it was recorded, alongside optional key-value pairs called labels.

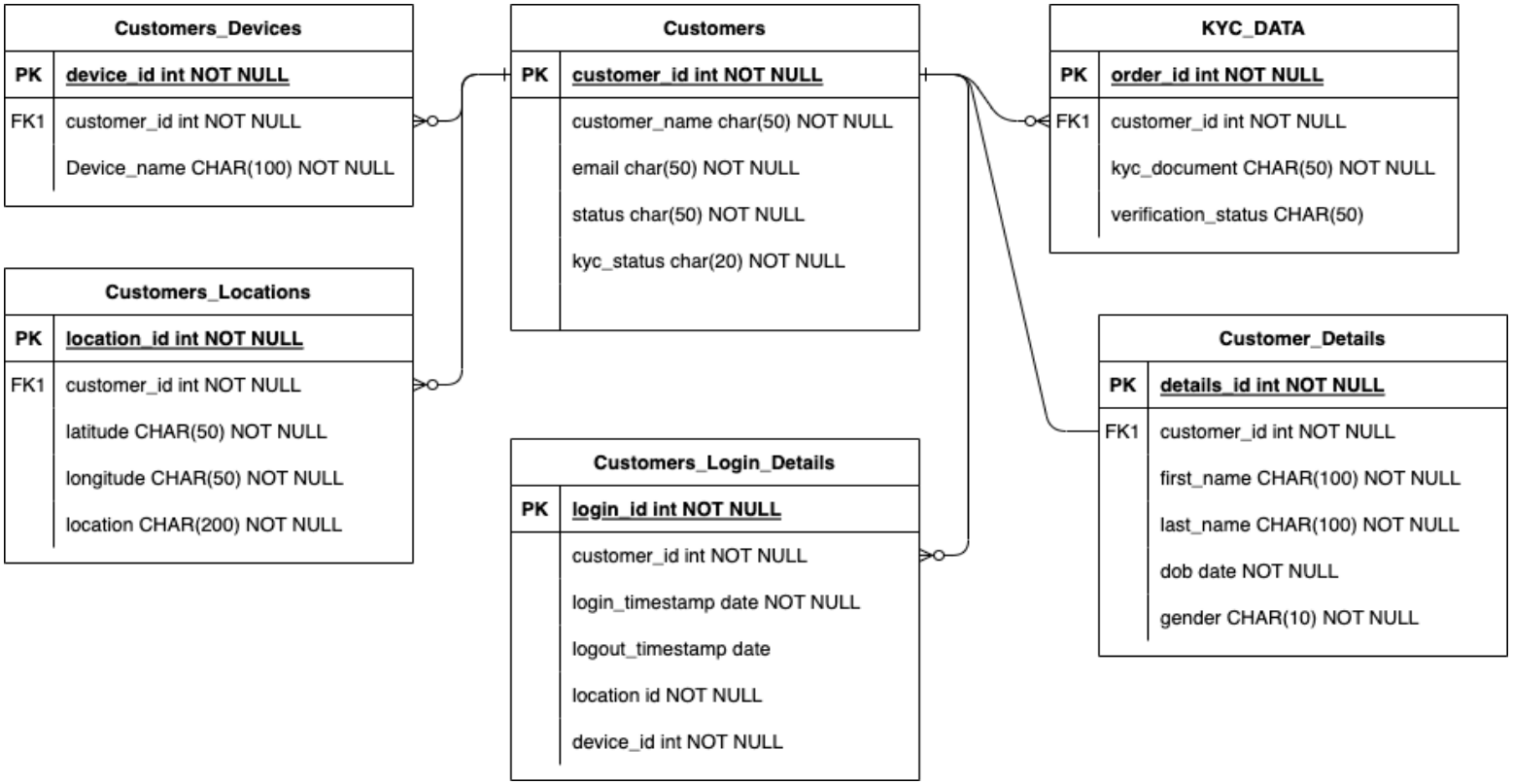
Grafana (9.3.2): Grafana is a multi-platform open source analytics and interactive visualization web application.

4.6. Log Management Software

Elasticsearch , Logstash and Kibana Stack : Elasticsearch is a search and analytics engine. Logstash is a server-side data processing pipeline that ingests data from multiple sources simultaneously, transforms it, and then sends it to a "stash" like Elasticsearch. Kibana lets users visualize data with charts and graphs in Elasticsearch.

5. Data model

5.1. Customer Domain Data Model ER diagram is shown below:



5.2. Billing Domain Data Model

Biller Domain Data is stored in MongoDB as Biller Customer Registration data has dynamic schema. MongoDB is a preferred option. MongoDB json object schema is listed below:

MongoDb JSON Object Schema

1. Biller_Type
2. Biller_Info
3. Biller_Metadata
4. Customer_Biller_Registration
5. Biller_Account

6. Customer
7. Bill

1. Biller_Type


```
{
   "$schema": "http://json-schema.org/draft-04/schema#",
   "$id": "https://biler.com/biller_type.schema.json",
   "title": "Record of employee",
   "description": "This document records the details of an employee",
   "type": "object",
   "properties": {
      "id": {
         "description": "A unique identifier for an biller type",
         "type": "number"
      },
      "biller_type_name": {
         "description": "biller type name",
         "type": "string",
         "minLength":3
      },
      "tooltip": {
         "description": "Tooltip to be displayed when hover over the icon",
         "type": "string",
         "maximum": 50
      },"descrition": {
         "description": "Overview of the Biller Type",
         "type": "string",
         "maximum": 100
      },"tooltip": {
         "description": "Tooltip to be displayed when hover over the icon",
         "type": "string",
         "maximum": 50
      }
   },
   "required": [
      "id",
      "biller_type_name",
      "tooltip",
      "Description",
      “icon_path”
   ],
 "additionalProperties": false
}
```


2. Biller_Info

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "$id": "https://biler.com/biller_type.schema.json",
    "title": "Record of employee",
    "description": "This document records the details of an employee",
    "type": "object",
    "properties": {
        "id": {
            "description": "A unique identifier for an biller information",
            "type": "number"
        },
        "biller_name": {
            "description": "biller name",
            "type": "string",
            "minLength":3
        },
        "tooltip": {
            "description": "Tooltip to be displayed when hover over the icon",
            "type": "string",
            "maximum": 50
        },"descrition": {
            "description": "Overview of the Biller",
            "type": "string",
            "maximum": 100
        },"biller_type_id": {
            "description": "Biller Type Identifier",
            "type": "number"
        },"icon_path": {
            "description": "Reference to path of the icon of Biller ",
            "type": "string"
        },,"ibiller_account_number": {
            "description": "Reference to path of the icon of Biller ",
            "type": "integer"
        }

    },
    "required": [
        "id",
        "biller_name",
        "tooltip",
        "description",
        “icon_path”,
        “biller_type_id”,
        “biller_account_number”
    ],
  "additionalProperties": false
}
```

3. Biller_Metadata

```
{
   "$schema": "http://json-schema.org/draft-04/schema#",
   "$id": "https://biler.com/biller_metadata.schema.json",
   "title": "Record of Biller Metadata required",
   "description": "Biller Metadata Records holds the schema of the data required for customer identification with Biller ",
   "type": "object",
   "properties": {
      "id": {
         "description": "A unique identifier for an biller metadat",
         "type": "number"
      },
      "biller_info_id": {
         "description": "reference to biller information identifier record",
         "type": "integer",
         "minLength":3
      },
      "customer _registration_metadat": {
         "description": "Metadata Records holds the schema of the data required for customer identification with Biller",
         "type": "array",
         "items": {
                        "type": "object",
                            "properties": {
                                "input_type": {
                                    "type": "string"
                                },
                                "Squence_number": {
                                    "type": "integer"
                                },
                                "label": {
                                        "type": "string"
                                },
                                "min_length": {
                                        "type": "integer"
                                },
                                "max_length": {
                                        "type": "integer"
                                },
                                "required": {
                                        "type": "string"
                                },
                                "regular_expression": {
                                        "type": "string"
                                }
```

```
                    },
        "required": [
            "input_type",
            "label",
            "sequence_number",

        ] ,

      }
    },
    "required": [
       "id",
       "biller_info_id",
       "customer_registration_metadata"
    ],
 "additionalProperties": false
}
```

4. Customer_Biller_Registration

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "$id": "https://biler.com/customer_biller_registration.schema.json",
    "title": "Record of employee",
    "description": "This document holds the customer biller registration data",
    "type": "object",
    "properties": {
       "id": {
          "description": "A unique identifier for an customer_biller_registration",
          "type": "number"
       },
       "customer_id": {
          "description": "customer_id",
          "type": "integer",
          "minLength":3
       },
      “registration” :{
       “type” : “array”,
       “items” :{
            "biller_id": {
            "description": "Biller Identified",
            "type": "integer",
            "minLength":3
         },
        “biller_registration “:{
```

```
        “type” : “array”,
         “items” :{
           "nick_name": {
              "description": "Name to quickly identify the customer registration for biller. This is useful in case of multiple biller registration ",
               "type": "string",
               "maximum": 50
           }, "biller_registration_metadata": {
                "description": "Customer Registration Data against the biller",
               "type": "string",
              "maximum": 1000
           }
         },"required": [
            "nick_name",
            "biller_registration_metadata" ],
        }
      }, required": [
           "biller_id,
           "biller_registration" ],
     }
   },
    "required": [
      "id",
      "customer_id",
      "registration"
   ],
 "additionalProperties": false
}
```

5. Customer

```
{
   "$schema": "http://json-schema.org/draft-04/schema#",
   "$id": "https://biler.com/customer.schema.json",
   "title": "Record of employee",
   "description": "This document records the details of an customer in biller domain",
   "type": "object",
   "properties": {
      "custoomer_id": {
         "description": "A unique identifier for an customer",
         "type": "number"
      },
      "customer_name": {
         "description": "customer name",
```

```
        "type": "string",
        "minLength":3
      },
        },
  "required": [
    "customer_id",
    "customer_name"
  ],
"additionalProperties": false
}
```

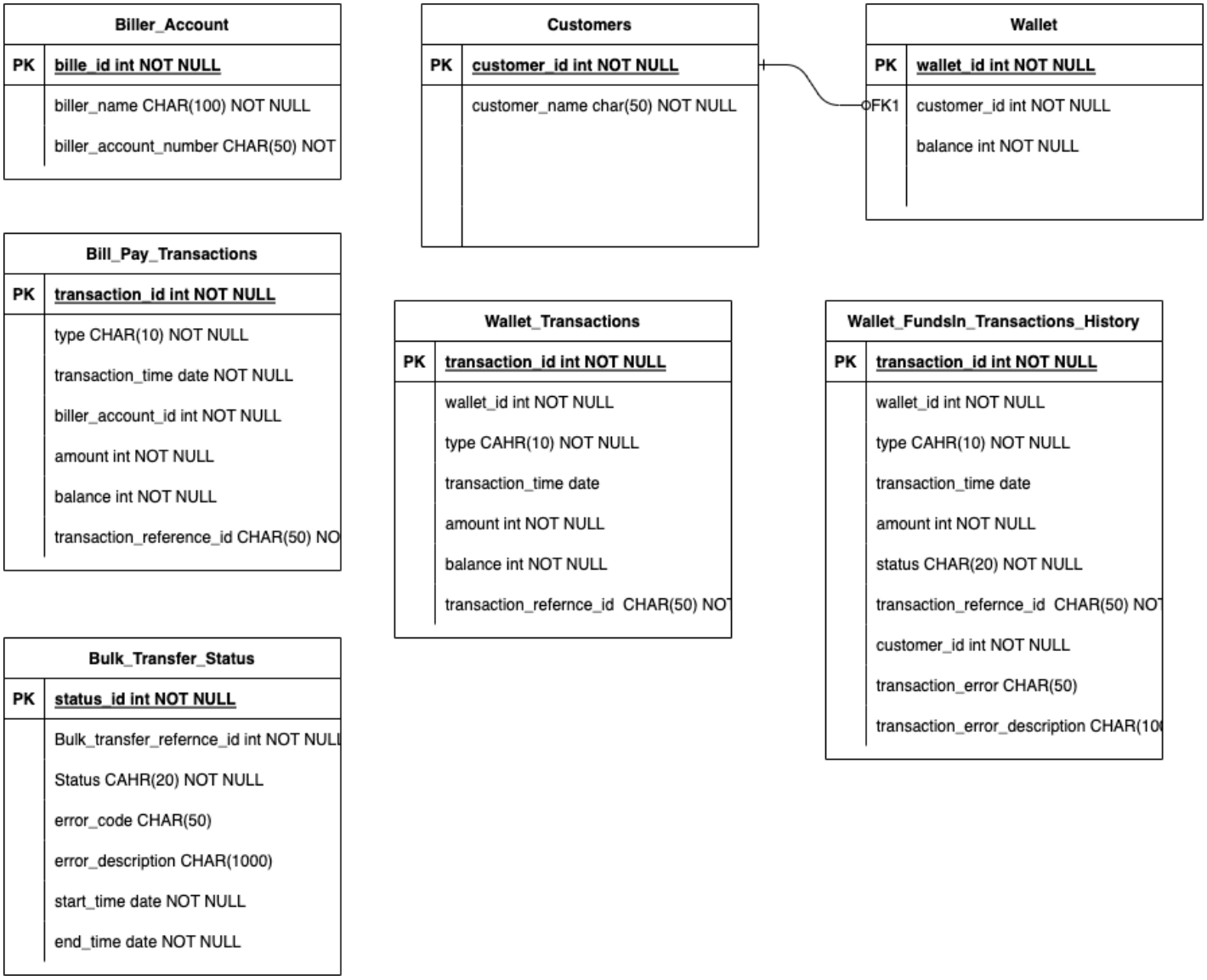
6. Bill

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "$id": "https://biler.com/customer.schema.json",
  "title": "Record of employee",
  "description": "This document records the details of an customer in biller domain",
  "type": "object",
  "properties": {
    "custoomer_id": {
      "description": "A unique identifier for an customer",
      "type": "number"
    },
    "bill_reference_number": {
      "description": " Bill Reference number received from biller",
      "type": "string",
      "minLength":4
    },
    "biller_id": {
      "description": " Identifier for Biller",
      "type": "string",
      "minLength":4
    },
    "amount": {
      "description": " Bill Reference number received from biller",
      "type": "integer"
    },
    “from_date”:{
      "description": " Bill Start Date",
      "type": "string"
     }
    “to_date”:{
```

```
            "description": " Bill Date",
            "type": "string"
        }, “bill_generation_date”:{
            "description": " Bill Date",
            "type": "string"
        }

        },
    "required": [
        "customer_id",
        "bill_reference_number",
        “bill_id”,
        “amount”,
        “from_date”,
        “to_date”,
        “bill_generation_date”
    ],
"additionalProperties": false
}
```


5.3. Payments Domain Data Model



5.4. **Reporting Domain Data Model** : Payments and Reporting have the same data model, however Reporting Data stored in cassandra have higher data retention as per country specific requirement.

6. Specs for external and inter micro service APIs

Please refer the swagger file attached for API Specification in API Spects folder in github
https://github.com/vivek-bhandarkar/bill-pay-application-design/blob/master/API%20Documentation/BillPay_Services.yaml

7. Testing strategy

Application should have 90% testing coverage. Test automation should be done with the listed below toolset.

Unit Tests : JUnit, Mokito and PowerMock should be used to carry out unit test automation.
Integration Test : RestAssured Framework should be used to carry out Integration test automation.
End To End Testing : Selenium Driver should be used to carry out end to end test automation.
Contract Testing : Spring Cloud Contract framework should be used to carry out Contract test automation.
Performance Test : Open Source tools like JMeter or commercial tool HP LoadRunner should be used to perform the load test automation.
Endurance Test : Open Source tools lie JMeter or commercial tool HP LoadRunner should be used to perform the endurance test automation.
RegressionTesting : Selenium Driver should be used to carry out end to end test automation.

8. Non functional requirements :

8.1. Security

1. Configure Rate Limiter at Spring Cloud API Gateway layer to provide projection against the DDOS attack
2. Use SSL certificates for microservice communication. Renew and Refresh the certificate automatically with Hashicorp Valut.
3. SSL should be used for microservice communication
4. Store the sensitive configuration (password, certificates) in encrypted form. Spring Cloud config server should be configured with Hashicorp Vault.
5. Secure application authorization and authentication with OpenId Connect utilizing KeyCloak.
6. Generate database secrets dynamically with Hashicorp Vault and configure with Spring Boot Microservices.
7. Configure AWS Web Application Firewall WAF with ALB. As WAF can not be configured with NLB. ALB can configured with AWS WAF in between NLB and ingress controller configured.AWS WAF helps protects your website from common

attack techniques like SQL injection and Cross-Site Scripting (XSS). In addition, you can create rules that can block or rate-limit traffic from specific user-agents, from specific IP addresses, or that contain particular request headers.

8. Use Container Image scanning tools to check for vulnerability
9. Setup monitoring software Prometheus, Grafana to monitor application for any abnormal pattern and detect any attack.
10. Configured API Gateway provides facility for API Screening, Protocol translation, routing, SSL termination.
11. Configure Container orchestration like Kubernetes it provide inbuilt facility to store configuration files and secrets. Also provides isolation with namespaces.
12. Configure firewall rules diligently to expose only required services on AWS. Also setup IAM roles as required by Application.
13. Ensure all the data at rest is encrypted.
14. Ensure OWASP Top vulnerability mitigation guidelines are followed.
15. Setup infrastructure access controls using AD groups to limit the access to the intended user base only.

8.2. Logging , Monitoring & Alerting

Logging : EKS Cluster Pods should have sidecar logstash container integrated with ELK for log storage and log analysis.

Monitoring : All Spring Boot Microservices should be configured with Spring Boot Micrometer framework for metrics publishing. This will expose application metrics on actuator endpoints. Metrics published should be scraped and stored by the Prometheus database. Grafana dashboard should be integrated with the Prometheus database for application monitoring. Custom metrics can be exposed and monitored on the Grafana dashboard.

Alerting : AlertManager is configured with Prometheus to send slack/email notification in case metrics configured crosses the threshold value.

8.3. Data Backup

Setup policy to backup databases /caches at regular interval

8.4. High Availibility

Ensure AWS infrastructure /services are configure with High Availability multi-AZ setup.

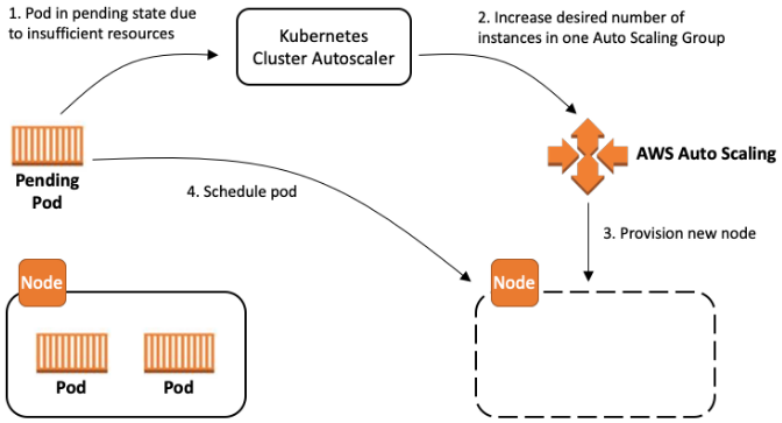
8.5. Auto Scaling

8.5.1. ASW Infrastracture Component Autoscaling

AWS ElacticCache Redis, MCS, Document DB , etc ..should be done as per documentation provided by AWS provider.

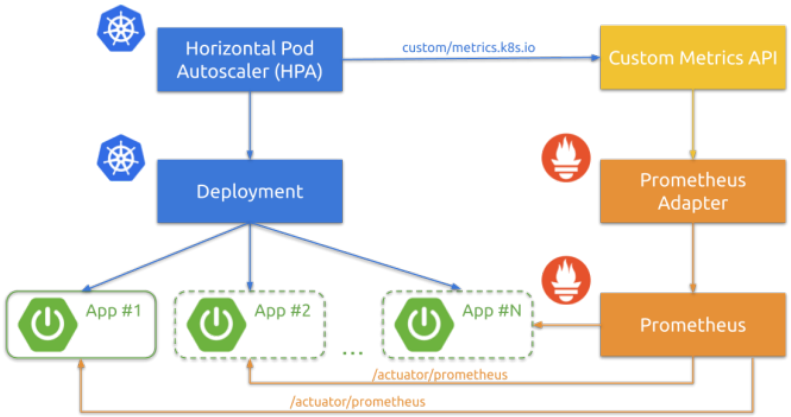
8.5.2. AWS EKS Worker Node Autoscaling

Configure Auto Scaling for EKS Worker Nodes to scale if the worker node reaches set threshold system utilization parameters.



8.5.3. EKS Pods Autoscaling

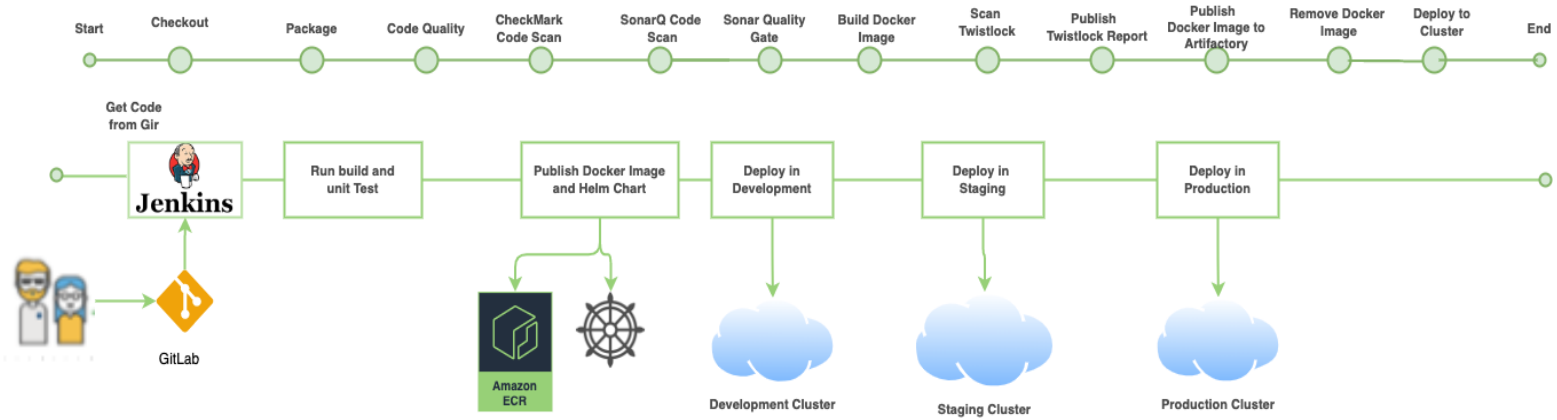
The horizontal pod autoscaler (HPA) automatically scales the number of pods based on CPU, memory, or other custom metrics. It obtains the value of the metric by pulling the Custom Metrics API. In the beginning, we are running a single instance of our Spring Boot application on Kubernetes. Prometheus gathers and stores metrics from the application by calling HTTP endpoint /actuator/promentheus. Consequently, the HPA scales up the number of pods if the value of the metric exceeds the assumed value.



8.6. CI CD processes and tech stack

CI CD process and tech stack is mention below:

Please refer the below diagram for CI-CD Process



CI-CD Tech Stack as below :

Sr. No.	Tool used in the Pipeline	Purpose of the tool
Continuous Integration CI		
1	GitLab	Provides a Git-repository management for Application Source Code/Properties
2	Git Client	Used to make connection between GitLab and Jenkins for Application Source Code/Properties Checkout to Jenkins
3	Jenkins	Jenkins is an open source automation tool written in Java with plugins built for Continuous Integration purpose.
4	Groovy	Used in Jenkins to write JenkinsFile for Pipeline type of Jenkins Job
5	Maven	Maven is a build automation tool used primarily for Java projects.
6	Junit /Cucumber	JUnit is a unit testing framework for Java programming language. A cucumber is a tool based on Behavior Driven Development (BDD) framework which is used to write acceptance tests for the web application.
7	SonarQube	Continuous inspection of code quality to perform automatic reviews with static analysis of code to detect bugs, code smells, and security vulnerabilities.
8	Checkmark	CxSAST is a powerful tool with a lot of room to customize scan rules and integrate into different build processes.
9	Twistlocker	Provides vulnerability management and compliance across the application lifecycle by scanning images and serverless functions to prevent security and compliance issues from progressing through the development pipeline, and continuously monitoring all registries and environments
10	Jfrog Artifactory	A Repository Manager that functions as a single access point organizing all of your binary resources including proprietary libraries, remote artifacts and other 3rd party resources.
11	Docker	Building, shipping, and running container applications
Continuous Deployment CD		
14	AWS Code Build Pipeline	AWS Code Pipeline to use on deploying containers to AWS Fargate ECS Cluster
15	CD: AWS: EKS	AWS EKS Elastic Kubernetes Service is set of node machines for running containerized applications, fully managed by AWS
16	Ansible / Helm	Ansible used to deploy build result on Helm chart on EKS cluster in AWS.

8.7. Automation of test suites

Below frameworks are used to provide the test automation

Unit Tests : JUnit, Mokito and PowerMock should be used to carry out unit test automation.
Integration Test : RestAssured Framework should be used to carry out Integration test automation.
End To End Testing : Selenium Driver should be used to carry out end to end test automation.
Contract Testing : Spring Cloud Contract framework should be used to carry out Contract test automation.
Performance Test : Open Source tools like JMeter or commercial tool HP LoadRunner should be used to perform the load test automation.
Endurance Test : Open Source tools lie JMeter or commercial tool HP LoadRunner should be used to perform the endurance test automation.
RegressionTesting : Selenium Driver should be used to carry out end to end test automation.
Test automation suites developed should be integrated with CI pipelines. Performance or Endurance test should be carried out in a Production like environment with Production peak load and production identical load for prolonged duration.

8.8. Support PI/PCI Data. Application should be PCI compliant.

Follow PCI Compliance Guidelines as below
1: Install and maintain a firewall configuration to protect cardholder data.
2: Do not use vendor-supplied defaults for system passwords and other security parameters.
3: Protect stored cardholder data.
- Enable Encryption for the physical storage devices
- Enable Encryption for Datastores

4: Encrypt transmission of cardholder data across open, public networks.
- Configure HTTPS (TLS1.3) for all public facing network access to encrypt data in transit

5: Protect all systems against malware and regularly update anti-virus software or programs. Customer Shared Provider
- Regularly Patch Servers
- Provide regular software updates and critical patches

6: Develop and maintain secure systems and applications.
- Follow secure coding practices
- Provide protect for OWASP Top 10 vulnerability

7: Restrict access to cardholder data by business need to know.
- Production cardholder data /environment should have limited access

- Cardholder data should be encrypted
- Databases should be encrypted
- Ensure that PI data should not be logged, if required then it should be masked

8: Identify and authenticate access to system components.
- Setup Authentication and Authorisation.

9: Restrict physical access to cardholder data.

10: Track and monitor all access to network resources and cardholder data.

11: Regularly test security systems and processes.
-Carry out regular automated scans of all application endpoints and infrastructure components

12: Maintain a policy that addresses information security for all personnel.