ARCHITECTURE DOCUMENT

Customer Income Capture Microservice

CLIENT	**Enterprise banking client**
STACK	Java, Spring Boot, MongoDB, Apigee, Jenkins CI/CD, Hadoop/Hive, SQL, Kubernetes, PCF, AWS ECS, Lambda
AUTHOR	Akash Ramesh
STATUS	Live, in production for an enterprise banking client

1. System Overview

This service extends the client's core banking application to capture customer declared income directly at the point of credit card activation. Previously, income data came from credit application forms, often 3 months or more out of date, or from third party income surveys that could take up to 6 months to return results. Adding a direct, first party data entry point at the moment of activation closes that gap.

All external traffic reaches the service through the Apigee gateway. Apigee handles authentication, rate limiting, and routing before any request reaches the Spring Boot application, which keeps the service itself focused on business logic.

2. High Level Architecture

Component map

The system is composed of six layers, each with a distinct responsibility:

Layer	Component	Responsibility
Client	Card activation flow	Customer declares income during activation
Gateway	Apigee	Authentication, rate limiting, routing
Application	Spring Boot Service	Business logic, income capture and validation
Data	MongoDB	Transactional store for captured income
Batch	Hadoop/Hive Pipeline	Extended SQL queries feeding the enterprise datalake
Analytics	Tableau	Business analyst dashboards over datalake data

Request flow

The customer facing hop goes through Apigee only. Everything downstream of that, the database write and the batch pipeline, is internal to the backend and never reachable directly from the customer.

```
Customer
 |
 |-- Declares income at activation --> Apigee Gateway
 |<-- Activation confirmed --------------|
 |
 |-- Authenticated request ------------> Spring Boot Service
                    |
                    |-- Write income event ------> MongoDB
                    |-- Extended batch SQL ------> Hadoop/Hive Pipeline

Hadoop/Hive Pipeline
 |-- Load into datalake --> Enterprise Datalake
                    |
                    |-- Read --> Offer Targeting System
                    |-- Read --> Tableau
```

3. Income Capture Flow

- Customer declares income during credit card activation, using the new capture step added to the existing activation flow
- The declaration request reaches the Apigee gateway first
- Apigee authenticates the request, applies rate limiting, and routes it to the Spring Boot service
- The Spring Boot service validates the income fields for the customer's employment type and writes the event to MongoDB
- On the next batch run, the extended Hadoop/Hive pipeline picks up the new income records through the extended SQL queries
- The pipeline loads the data into the existing enterprise datalake
- From the datalake, the data reaches both the offer targeting system and the business analyst Tableau dashboards

4. Apigee Gateway Design

Authentication, rate limiting, and routing are all handled at the Apigee gateway layer, ahead of the Spring Boot service. Every request is authenticated and checked against rate limits before it is routed to the application.

This was a deliberate choice over building the same checks into the service itself. Keeping security policy at the gateway meant the microservice stayed focused on business logic, and meant policy changes, such as a new rate limit or an updated auth rule, could be applied at the gateway without a service redeployment.

5. Data Model

Income data at card activation is document shaped, not relational. Each activation event has a variable set of income fields depending on the customer's employment type, so MongoDB was chosen as the transactional store rather than a relational database.

For example, income document structure can vary by employment type: a salaried customer's record centers on employer and base salary fields, while a self employed customer's record instead centers on declared business revenue. This variability is what a fixed relational schema would struggle to accommodate without frequent migrations. MongoDB absorbs it directly, without a schema migration for each new income type the business wants to capture.

6. Pipeline Extension

Rather than build a parallel data pipeline for the new income data, the existing Hadoop/Hive batch pipeline was extended with new SQL queries. The extension reads the new MongoDB income collection and maps the relevant fields into the pipeline's existing table structures in Hive.

This kept the income data flowing through the same batch cadence and into the same enterprise datalake the business already relied on, with no separate pipeline to build, monitor, or hand off to a different team. The same existing Tableau dashboards picked up the new data without analyst retraining.

7. CI/CD Pipeline

The service was built and deployed through a Jenkins CI/CD pipeline, automating build and deployment on every merge.

The team's infrastructure migrated from Pivotal Cloud Foundry to a combination of Kubernetes for container orchestration and AWS ECS and Lambda for scalable workload hosting. The Jenkins pipeline was updated as part of this migration so builds continued to deploy automatically to the new targets.

8. Security Design

Concern	Approach	Where enforced
Authentication	Apigee API key and OAuth policies	Apigee gateway

Concern	Approach	Where enforced
Rate limiting	Apigee rate limiting policies	Apigee gateway
Routing	Apigee routing rules to the Spring Boot service	Apigee gateway
Data storage	Income data held in MongoDB with internal service credentials	MongoDB
Pipeline access	Hadoop/Hive pipeline and datalake access restricted to internal service credentials	Hadoop/Hive, enterprise datalake
Deployment access	AWS IAM roles for ECS and Lambda workloads	AWS

9. My Contributions

Within the engagement team, my specific ownership:

- Designed and implemented the Spring Boot microservice, including the income capture endpoint and MongoDB persistence layer
- Configured Apigee gateway policies for authentication, rate limiting, and routing
- Extended the Hadoop/Hive batch pipeline with new SQL queries to bring captured income data into the enterprise datalake
- Built the Jenkins CI/CD pipeline and carried it through the migration from Pivotal Cloud Foundry to Kubernetes and AWS
- Led code reviews for the service
- Ran stakeholder calls with client side leads

Source code is in a private client repository. Akash Ramesh is available to walk through any aspect of this architecture in detail during interview or technical review.