Raúl Homero Llasag Rosero

BANKING PLATFORM: SOFTWARE ARCHITECTURE

Proposal as candidate to Software Government Analyst - Financial/Banking Sector

September 2024

Abstract

ONLINE banking transactions must be communicated in real-time, while banking systems are required to maintain high availability and adopt fault-tolerant strategies. Developing applications that meet these demands can be challenging, especially for less experienced software engineers, due to the need for seamless real-time communication between banking systems. This document presents the software architecture for two platforms developed by BP S.A.: web and mobile applications. To streamline development through DevSecOps practices—focusing on development, security, and operations—this document first decouples the software architecture using the C4 model. After detailing the context, containers, components, and code diagrams of the C4 model, it outlines the integration of DevSecOps practices and cloud computing tools designed to ensure high availability, data privacy, and fault tolerance.

Keywords: C4 model, Banking, Software Architecture, DevSecOps

This document is publicly available at: github.com/rhllasag/BPSystem.

Contents

List of Figures

Acronyms

API	Application Pprotocol Interface
AWS	Amazon Web Sservices
BP	Bank Payments S.A.
C4	Context, Containers, Components, Code
CI/CD	Continuous Integration/ Continuous, Development
DB	Data Base
DevSecOps	Development Secure Operations
DR	Disaster Recovery
FR	Functional Requirement
FT	Fault Tolerance
HA	High Availability
HTTP	Hypertext Transport Pprotocol
HTTPS	Hypertext Transport Pprotocol Secure
NoSQL	No Structured Query Language
OAuth2	Open Authorization
OpenID	OpenID Connector
SAML	Security Assertion Markup Language
SPA	Single Page Application
SQL	Structured Query Language
TCP	Transmission-Control Protocol

1. Introduction

Contents

This document outlines the proposed software architecture for Bank Payments S.A. (BP). Both customers and bank accountants require access to BP's services through two highly available online platforms: web and mobile applications.

This document first addresses the primary challenges banks face in building online platforms. It then outlines the specific requirements for BP's web and mobile applications. To aid engineers in developing these platforms, the main components and data exchanges are described using the C4 model, adhering to current banking regulations and data privacy standards. Finally, strategies for development operations and monitoring are discussed to accelerate product delivery, along with approaches to ensure High Availability (HA) and Fault Tolerance (FT).

1.1 Problem

Real-time and secure fund transactions are vital for the growth of the global economy. Automatically updating bank accountants with the current balance after a transaction is crucial to the success of online banking platforms. This feature extends beyond accountants within a single bank to those across different banks, where online services must ensure both high availability and data consistency.

1.2 Functional Requirements

This section describes the main features of web and mobile applications of BP. The web application gives access to a logged bank accountant to the Functional Requirements (FRs):

FR1 Display basic information about the bank account: username and password:

FR2 Display historical fund transactions.

FR3 Display historical of used services such as interbank transference, transfer money internationally and loans.

The mobile application gives access to a logged bank accountant to the following FRs:

FR4 Visualise the information the personal bank account: username, password, fingerprint, photo, passport, identity card number, etc.

FR5 Transfer money with other bank accountants of BP.

FR6 Use external services such as inter bank transference, transfer money internationally and loans.

1.3 Non Functional Requirements

The web and mobile applications implement a standard authorization protocol, such as OAuth2. While OAuth2 is recommended, other widely accepted protocols in banking services include OpenID and SAML 2.0.

The web application is designed as a Single Page Application (SPA), where only specific elements of the page are refreshed during fund transactions. In contrast, the mobile application must be developed for multiple platforms, such as iOS and Android.

Among the key requirements, HA and FT are critical, and this proposal addresses both effectively.

1.4 Document Outline

This document details the components and communication of web and mobile applications of BP across five chapters. Below is an outline:

- Chapter 2 describes the high-level relationships between the BP system and the bank accountants.
- Chapter 3 describes the internal components of the BP system and how web and mobile applications interact with each other.
- Chapter 3 decompose each container further to identify their functionalities, structural building blocks and interactions.
- Chapter 4 decompose each container further to identify their functionalities, structural building blocks and interactions.
- Chapter 5 details the code structure of each component using data diagrams.
- Chapter 6 describes how the proposed architecture can adopt Continuous Integration/ Continuous, Development (CI/CD) and DevSecOps tools to sped up product delivery. Finally, this chapter describes operation strategies for ensuring High Availability (HA), data integrity and Fault Tolerance (FT).

2. Context

Contents

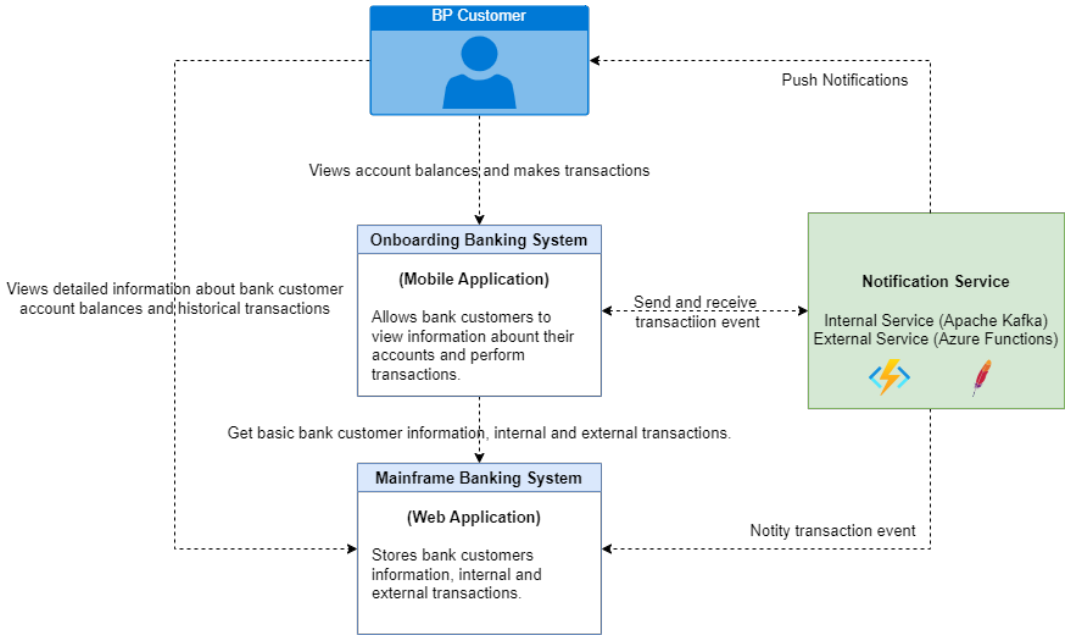
ONLINE Online bank transactions must be communicated immediately after they are completed. This communication is critical for the BP system, which consists of both web and mobile applications. This chapter outlines the high-level interaction between these applications and bank customers.

2.1 Introduction

A bank customer can view account balances through both the web and mobile applications. While the web application, currently envisioned as a mainframe-like system, provides detailed information on account holders and historical transactions, it does not support executing transactions. The mobile application, functioning as an onboarding system, guides customers step by step through internal and external transactions.

Figure 2.1 illustrates how a bank customer has different viewing and service permissions depending on whether they log in via the mobile or web application. The figure also shows the interaction between both applications through a notification service, which enables real-time communication when a money transfer is completed.

Figure 2.1: Context Diagram.



2.2 Onboarding and Mainframe Banking Systems

The onboarding mobile application displays basic customer information retrieved from the mainframe system. The mainframe provides the necessary methods to access internal and external services, such as through RESTful API endpoints.

All internal and external money transfers made through the mobile application are processed by the mainframe banking system, which updates the account balance and logs the transaction in the historical records.

2.3 Notification System

To automatically update the views of both web and mobile applications—especially the account balance field—a notification service allows these applications to subscribe to event occurrences. The need for an internal notification system was outlined in the requirements. Apache Kafka, a stream-processing platform, can be utilized for this purpose without the need to contract external notification services from cloud-computing vendors. Table **??** lists popular external services provided by cloud-computing vendors.

Table 2.1: Real-time data communication alternatives

Service	**Vendor**	**Pricing**	**Free-trial mode**
Lambda functions	AWS	$0.20 for 1st M per month	1M per month
Azure functions	Azure	$0.20 per 1M in a month	1M per month

2.4 Banking Regulations and Security Compliance

Banking regulations vary across different regions. Although on-device biometrics, such as facial recognition, were mentioned in the requirements, the European Banking Authority (EBA) did not recognize them as a valid method for a second authentication factor as of January 2023. Strong customer authentication, a security requirement established by the EBA[1], mandates that customers provide two or more of the following authentication factors:

- Something the customer knows (e.g., a password or PIN),
- Something the customer possesses (e.g., a cell phone or security token); and
- Something the customer is (e.g., biometric data such as the face, fingerprint, or voice recognition).

[1] https://www.eba.europa.eu/homepage

3. Containers

Contents

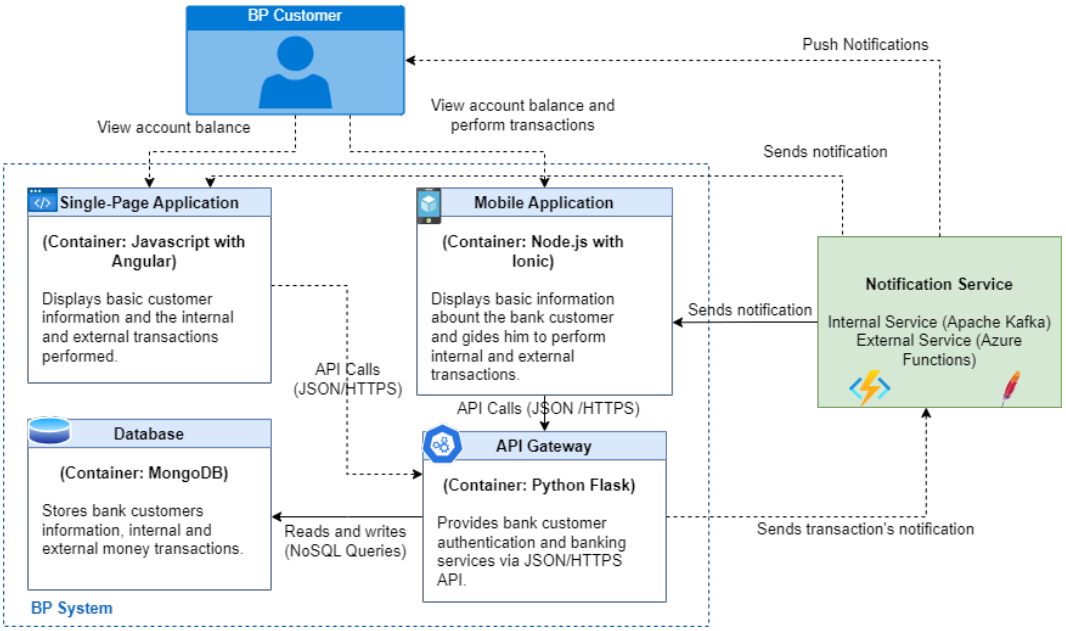
BANKING applications must interact with one another to ensure that customer account balances remain up to date. This chapter explains how different containers communicate with each other to implement the functionalities of the BP system.

3.1 Introduction

The banking system, represented by the blue dotted square in Figure 3.1, consists of four key containers: SPA, Mobile App, API, and Database.

At this stage, the mainframe banking system has been decoupled into SPA and an API gateway to facilitate engineers in developing the BP system. Additionally, the external container, shown in green, is designated for a specific notification service, though its implementation will depend on whether a cloud provider is selected.

Figure 3.1: Container Diagram.



3.2 Single-Page Application

A Single Page Application (SPA) is a web application that loads a single web document and updates the body content dynamically when money transactions occur. Several software tools can be used to implement a SPA, including Angular, Meteor, React, and Flask. In this case, Angular was chosen for its strong performance, scalability, and robust feature set.

While the SPA does not allow customers to perform transactions, it enables them to view their information and past transactions by requesting data from the API gateway via HTTP requests.

3.3 API Gateway

The API gateway allows subscribers to access internal and external transaction and login functionalities through specific endpoints. The use of HTTP methods such as POST, GET, PUT, and DELETE will depend on the required operations for creating, reading, updating, or deleting data.

The HTTPS protocol is utilized to enhance functionalities in production while adhering to privacy policy requirements. The OAuth2 standard protocol is recommended for implementing authorization, although OpenID and SAML 2.0 are also viable alternatives. Authentication through social media and email accounts, such as those provided by AWS Cognito, is not recommended due to stringent banking data privacy policies.

A container using the Python Flask framework was selected to implement the API gateway due to its seamless integration with artificial intelligence services, including generative neural networks.

3.4 Database

The API directly connects to the database, which performs CRUD operations through queries. There are numerous data storage alternatives available in the market, including MySQL, Oracle, DynamoDB, and InfluxDB. This proposal recommends adopting MongoDB to facilitate the integration of new information fields, such as images for face recognition, due to its flexibility as a NoSQL database.

3.5 Mobile Application

The BP mobile application must be compatible with multiple platforms. Developing native applications is not recommended; instead, cross-platform frameworks like Ionic and React Native can expedite product delivery. This proposal suggests using Ionic for less experienced mobile developers while noting that React Native offers superior performance.

The mobile application accesses BP functionalities through HTTPS requests to the API. When a money transaction is completed, the mobile application receives a push notification. This functionality is achieved by subscribing to notification event services.

3.6 Continuous Integration / Continuous Development

When defining containers, an experienced software developer typically considers the software development lifecycle and the necessary tools. This document encourages developers to adopt DevSecOps practices to mitigate the risk of releasing code with security vulnerabilities. By fostering collaboration, automation, and clear processes, developers can share responsibility for security throughout the development process, rather than addressing it only at the end when issues can be more challenging and costly to resolve.

Figure 3.2 illustrates the proposed inclusion of SonarQube and OpenVAS containers to ensure code quality and manage vulnerabilities. To empower developers to utilize various containers dedicated to CI/CD operations, the adoption of Azure services such as Pipelines, Container Registry, Key Vault, and Active Directory is recommended.

Figure 3.2: Proposed tools for DevSecOps based on Azure services.

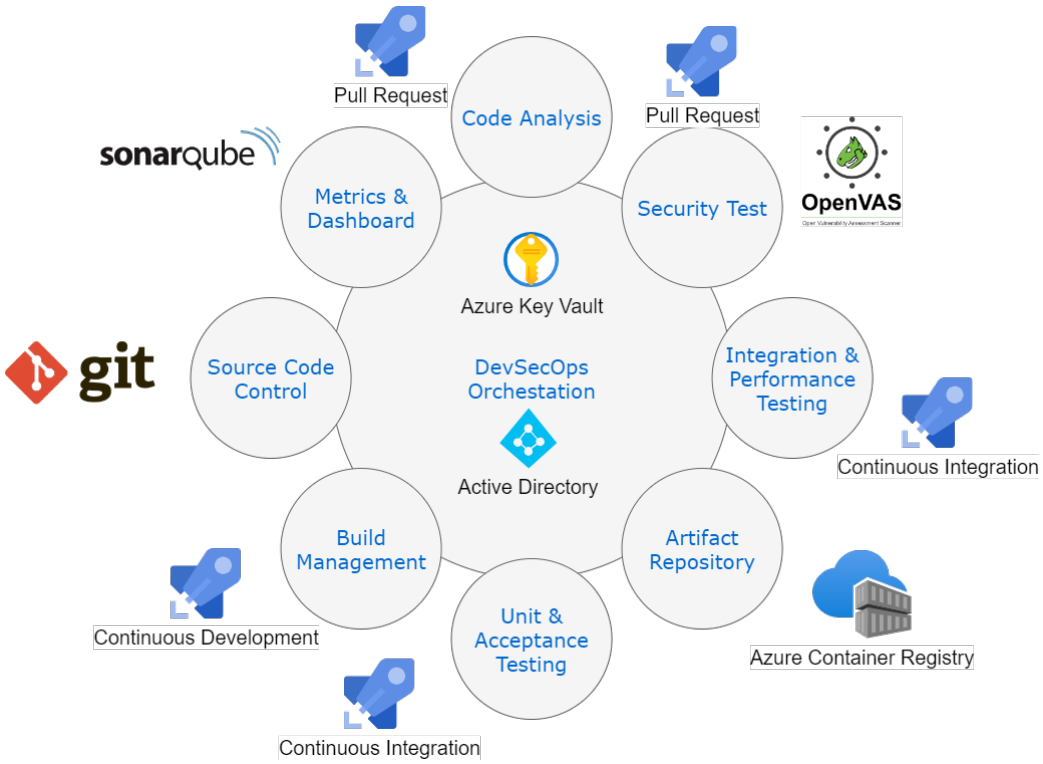


Figure 3.3 illustrates how a software developer and an operator can adopt CI/CD using Azure services. Git repositories are continuously evaluated using Azure Pipelines in pull request, CI and CD stages. Once a product version is approved, it passes from staging to production. Of course, the product put in production is monitored to guarantee its availability.

Figure 3.3: Continuous Integration and Continous Development.

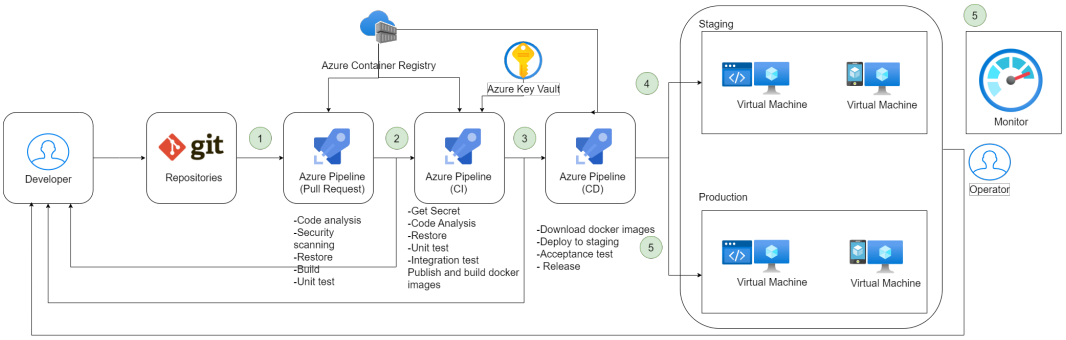


Table 3.1 describes the cost of using the proposed Azure services.

Table 3.1: DevSecOps costs in Azure

Service	**Pricing**
Azure Pipelines	$40 per every additional hosted CI/CD
Azure Artifacts	$2 per GiB
Azure Key vault	$5 per key per month
Azure Container Registry	$1,667 per day
Azure Active Directory	Premium P1 plan is priced at $6 user/month
Monitoring	$0.10 per GB per month

4. Components

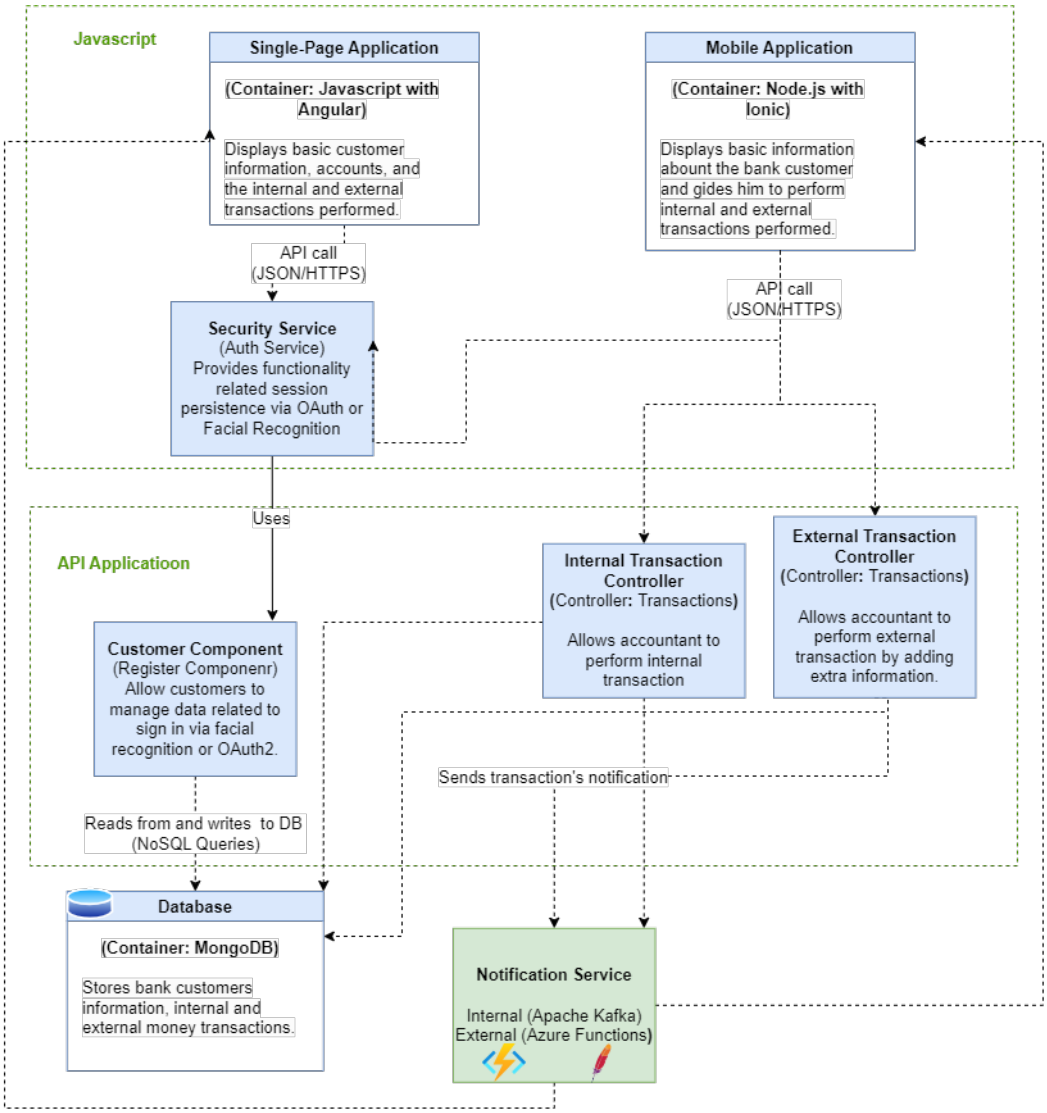
Contents

IDENTIFYING the major structural building blocks of a software solution and their interactions is essential for starting the development of any system. This chapter focuses on explaining the key components of the BP system, specifically the application pprotocol interface (API) gateway, as well as the web and mobile applications

4.1 Introduction

Figure 4.1 illustrates the main components of online platforms and the API gateway. The top green box illustrates a common component named security service implemented separately in SPA and mobile application.

Figure 4.1: Components Diagram.



The bottom green box illustrates three main components of the API gateway.

The customer component is dedicated to managing customers' data. On the other hand, internal and external transactions components manage money transference data.

4.2 Security Component

Web and mobile applications utilize a security service to authenticate customers. When implementing the OAuth2 service, an authentication service typically first checks if a customer is already logged in before requesting their credentials. For applications built with JavaScript or TypeScript frameworks, the authentication functionality accessed via HTTPS requests operates similarly. The choice between using local storage or session storage is at the discretion of the developers.

4.3 Customer Component

When authentication issues arise due to missing credentials or unregistered customers, the customer's information must be stored. This functionality is implemented by a customer component that directly interacts with the database to perform CRUD operations.

4.4 Internal and External Transaction Controller

A money transfer occurs when a customer transfers funds from their account to another customer's account. When both customers hold accounts within the same bank, the transaction does not require external data that the bank does not have access to. Conversely, an external transaction necessitates additional information from an external financial entity.

To accommodate the need for storing extra information from external customers, a dynamic transaction data structure is required. This challenge can be addressed by adopting a document-oriented database.

4.5 Notifications

Internal and external transactions must be communicated to the involved customers in real-time. While the components dedicated to money transfers do not directly implement notification methods, they trigger events in an external notification system. This allows subscribers to those events, such as web and mobile applications, to be easily notified of any new available information.

5. Code

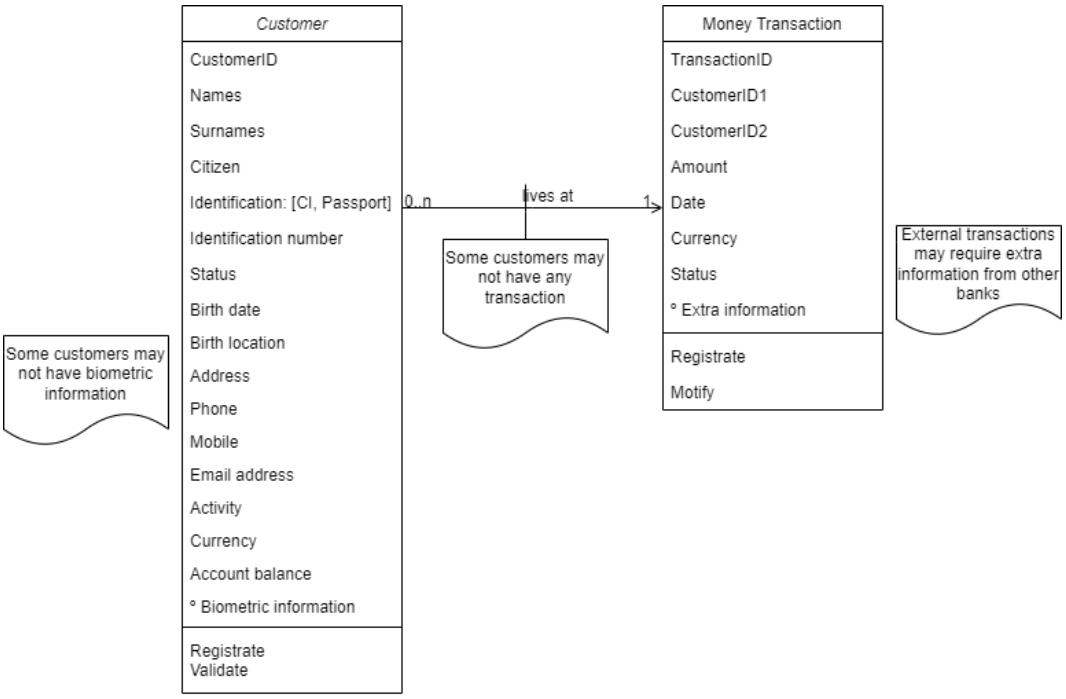
Contents

Low Low-level diagrams assist software developers in coding the solution, although detailed diagrams are not typically recommended when adopting agile methodologies. This chapter provides a brief overview of a class model that outlines the information required to open bank accounts for a legal entity, in accordance with the "Resolución de la Junta de Política Monetaria y Financiera 353".

5.1 Class Diagram

Figure 5.1 illustrates the interaction between two main objects: customers and their money transactions. Most Ecuadorian banks incorporate the fields listed in the customer class in compliance with current financial regulations. Additional fields, such as fingerprint codes and facial images, may be included if biometric authentication methods are adopted. By using a NoSQL database, these fields can be easily added without the need to update the entire data table.

Figure 5.1: Class Diagram.



Money transfers occur between two customers; however, some customers may not engage in any transactions. The current diagram illustrates a scenario where a transaction is performed between two BP customers. Additionally, it allows for the registration of external transactions by simply adding extra fields to the NoSQL database structure. This diagram can be utilized to code and deliver at least a minimum viable product.

6. Operations

Contents

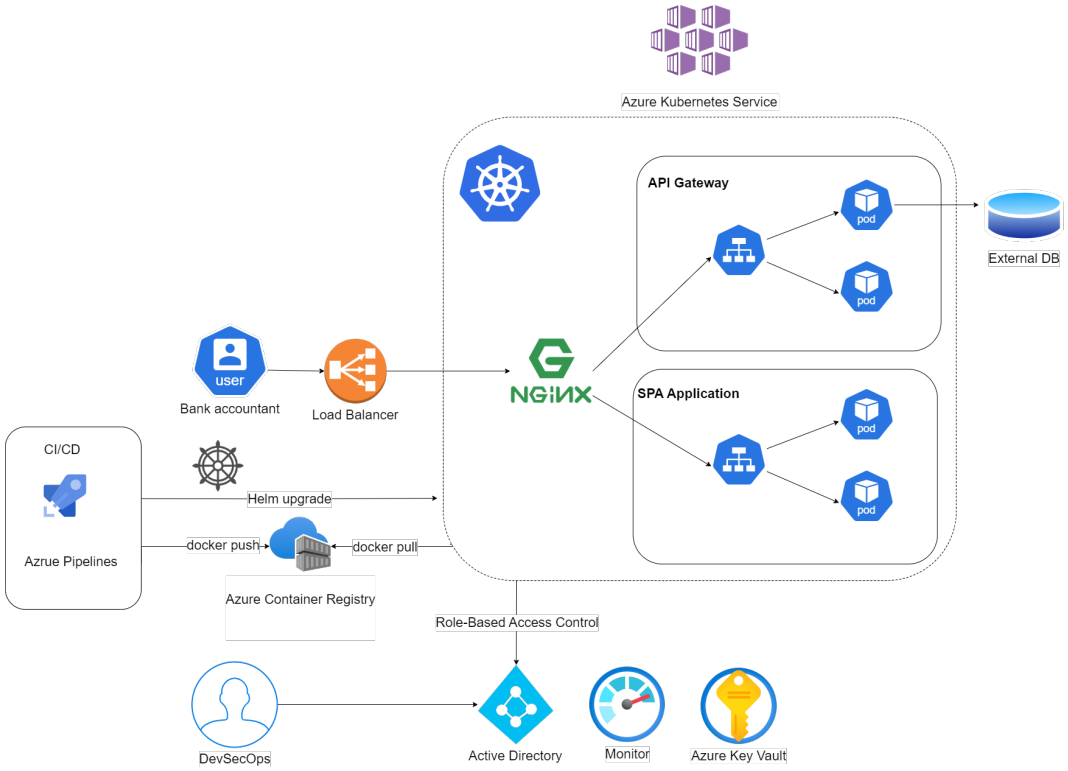
HIGH availability and fault tolerance are critical requirements in banking systems. While software developers can deliver Docker images to production, implementing highly available systems often requires additional effort. This chapter presents a proposed architecture that utilizes Kubernetes, an open-source container orchestration system, to automate software deployment.

6.1 Architecture

The proposed architecture, illustrated in Figure 6.1, introduces additional components to the previously defined DevSecOps architecture from earlier chapters. The adoption of Kubernetes technology is primarily aimed at enhancing a high availability (HA) architecture in the cloud.

Kubernetes facilitates the migration from local environments, typically hosted on personal computers, to Azure Kubernetes Services. Starting development with tools like kind or kubectl is free and ensures a smooth transition to production environments.

Figure 6.1: High Availability Architecture



Not all applications developed require high availability (HA) compliance. The large dotted box in Figure 6.1 illustrates a cluster in which an NGINX web server provides HA for the web and API applications.

HA is achieved when NGINX can select from multiple replicas of the same application, each located in different pods. If one replica fails, NGINX can

seamlessly switch to another replica. In practice, a load balancer performs this by selecting from various IP addresses. This entire process is abstracted from the end user, such as the bank accountant.

Figure 6.1 also highlights Azure components dedicated to CI/CD and emphasizes that monitoring operations are essential for mitigating faults.

6.2 Conclusion

This document guides developers and operators in implementing a robust architecture while encouraging the adoption of DevSecOps practices and cloud computing tools. Although this proposal is based on Azure, a similar architecture can also be constructed using other cloud computing providers or open-source tools.