EPAY APPLICATION ARCHITECTURE

November 2024

Document Control Sheet	
Current Version	2.0
Project Code	NS_SBI_04_08_2024
Project Name	SBI ePay Implementation
Document Type	ePay Application Architecture
Author	Sarita Kulkarni
First Reviewed By	Harshal Kirtane
Final Reviewed By	Ranu Jain
Approved By	
Contact Person	Sarita Kulkarni
Frequency of Document Review	Next Revision of Standard Documents
Document Creation Date	16-11-2024
Last Updated Date	27-11-2024

Revision History

Version	Date	Name	Comments
2.0	27-11-2024	Sarita Kulkarni	Draft Version

CONTENTS

Document Purpose....4

Document Scope....4

Audience of the Document....4

1. Overview of Application Architecture....6

2. Processing of Internal Application Architecture....7

3. SBI ePay Application Architecture....12

3.1 SBI ePay 2.0 Integration Architecture....12

3.2 SBI ePay 2.0 Information Architecture....14

3.2.1 Working of Transaction Service and Payment Service APIs....14

3.2.2 Working of Business Service APIs....16

3.2.3 Working of Security Service APIs....17

3.2.4 Working of Audit Service APIs....18

3.3 SBI ePay 2.0 Architecture Diagram....19

3.4 SBI ePay 2.0 Technical Architecture....23

3.4.1 Data Encryption and Decryption....24

3.4.2 Authentication and Logging....25

3.4.3 SCM Integration and Continuous Delivery Pipeline....27

3.5 SBI ePay 2.0 Application Architecture....29

4. SBI ePay 2.0 Network Architecture....34

4.1 High-Level Network Diagram (Old and New OCP Environments)....35

4.2 SBI ePay 2.0 OCP Network Diagram....39

4.3 IT Infrastructure Components Architecture....42

4.4 SBIePay 2.0 Outward Communication from Open Shift Cluster....43

4.5 SBIePay 2.0 Surround System to Outward Communication from Open Shift Cluster 44

Document Purpose

This document is designed to provide a comprehensive understanding of the network architecture integrated with various technical components. It aims to help team members gain a clear insight into the functionalities of each component, ensuring they can effectively contribute to the project. By detailing the design, setup, and interactions of the network, this guide serves as a valuable resource for understanding how the different elements work together to achieve the desired outcomes.

Document Scope

This document offers a comprehensive overview of the API integration and network architecture, detailing the integrity and interactions of all technical components involved in the project.

Audience of the Document

This document is intended for software professionals—including programmers, testers, system architects, and IT support staff—who are involved in the design, development, maintenance, and testing of the SBI ePay 2.0 system. The purpose of this document is to provide a comprehensive overview of the network architecture underlying SBI ePay 2.0, a leading payment gateway solution that facilitates secure and seamless digital transactions.

1. Overview of Application Architecture

The document explains the key components of the network architecture, the roles they play in ensuring the system's scalability, security, and availability, and how these components interact to support high-performance payment processing. Whether you are working on the back-end integration, testing payment flows, or analyzing the system's performance, this document will serve as a valuable resource to help you understand the technical intricacies of SBI ePay 2.0's network structure.

By understanding the network architecture of SBI ePay 2.0, software professionals can effectively contribute to the system's development, integration, testing, and maintenance. The modular design, coupled with a focus on security and scalability, ensures that the platform can meet the growing demands of online payment processing in a secure and efficient manner.

2. Workflow of Architectural APIs

This version is structured to highlight the flow of events during a transaction, with clear details about each service's function and how they interact with one another and the Oracle Database. It also adds clarity by grouping services into logical categories, which helps in understanding how each part of the system contributes to the overall architecture.

Overview of ePay 2.0 Payment Process

1. User Initiates Payment via UI Web Application
2. Authentication & Authorization of the user
3. Payment Information is securely entered and encrypted
4. Transaction Initiation with third-party payment gateways
5. Transaction Validation and payment authorization
6. Payment Confirmation and status update
7. Settlement and Fund Transfer to merchant
8. Refund (if applicable)
9. Reporting and transaction record-keeping
10. Post-Transaction Notifications sent to user and merchant
11. Logging & Auditing for system monitoring and compliance
12. Ongoing Data Security and Regulatory Compliance

The ePay 2.0 payment process is typically designed to handle secure, seamless, and efficient payment transactions between users, merchants, and financial institutions. Here's a step-by-step breakdown of how the ePay 2.0 payment process works:

2.1 User Initiates Payment (Frontend Interaction)

2.1.1 Action

The user (either merchant or customer) selects products or services on the Merchant Portal or ePay Checkout page and proceeds to the checkout.

2.1.2 Technology

The UI Web Application (which can be accessed through browsers like Chrome, Firefox, etc.) is used for initiating the payment.

2.2. Authentication and Authorization

2.2.1 Action

Before any payment is processed, the Authentication Service verifies the identity of the user or merchant through login credentials, session tokens, or multi-factor authentication (MFA).

2.2.2 API Call

The Authorization Service API generates a security token, validates it, and checks the referrer and IP address (IP whitelisting).

2.2.3 Technology

User authentication may use OAuth tokens or similar mechanisms for secure access.

2.3. Payment Information Submission

2.3.1 Action

The user enters payment details (credit card info, bank account details, UPI credentials, etc.) into the ePay Checkout page.

2.3.2 Encryption

The data entered is encrypted using a Shared Secret (Key) to ensure secure transmission.

2.4 Encryption Process

Plain text payment data is encrypted into cipher text.

2.4.1 API Call

The Encoding/Decoding Service may be used to ensure that sensitive data is securely encoded.

2.5 Transaction Initiation

2.5.1 Action

Once the payment details are entered, the Transaction Service API initiates the transaction by sending the payment request to the relevant payment gateway.

2.5.2 API Call

The ePay Transaction Initiation API is invoked to start the payment process, which communicates with external payment gateways (like UPI, Wibmo, or banks).

2.5.3 Payment Gateway

The transaction is passed to the Payment Gateway (e.g., UPI, Wibmo, etc.) for further processing.

4.6 Transaction Validation

2.6.1 Action

The ePay Transaction Validation process checks the payment details and ensures the transaction is valid.

Validation can include checking account balance, card validity, and anti-fraud measures (such as 3D Secure for card payments).

2.6.2 API Call

The Transaction Service API communicates with third-party services like UPI or Wibmo for validation and authorization.

2.6.3 Action

If the payment is valid, the transaction proceeds. If invalid, an error or failure message is returned.

2.7 Payment Confirmation

2.7.1 Action

Once the payment gateway authorizes the transaction, the status of the payment is updated.

2.7.2 API Call

The ePay Transaction Status Management API updates the transaction status (e.g., Success, Failed, and Pending) and syncs the data with Report DB and Ops DB for reporting and operational monitoring.

2.7.3 Notification

The Notification Service sends updates to the user and merchant via SMS or Email Gateway, informing them of the payment status.

2.8 Settlement and Fund Transfer

2.8.1 Action

After the payment is authorized, the system initiates the Settlement Process.

2.8.2 Settlement

Funds are transferred from the payer's account to the merchant's account via the payment gateway or financial institution.

2.8.3 Reconciliation

The transaction details are reconciled to ensure that all payments and transfers are correctly processed.

2.8.4 API Call

The Ops Service handles the settlement and reconciliation process.

2.9 Refund/Dispute (If Applicable)

2.9.1 Action

If the user requests a refund or there is a transaction dispute, the Refund Process is triggered.

2.9.2 Refund

The payment gateway processes the refund, and the transaction status is updated accordingly.

2.9.3 Reconciliation

The Ops Service reconciles the transaction to reflect the refund.

2.9.4 API Call

The Ops Service API manages the refund or dispute resolution, ensuring funds are returned to the customer and proper records are maintained.

2.10 Reporting and Record Keeping

2.10.1 Action

All transaction details are stored and reported for business analysis, auditing, and regulatory compliance.

2.10.2 Transaction Reports

The Report Service generates transaction-related reports, which include payment success rates, merchant performance, revenue, etc.

2.10.3 Database

Transaction data is stored in the Oracle Database, including transaction status, payment details, merchant information, and settlement records.

2.10.4 API Call

The Report Service API retrieves data from the database and generates reports for internal and external stakeholders.

2.11. Post-Transaction Notifications

2.11.1 Action

After the transaction completes (whether successful, failed, or refunded), the system sends final notifications to the user and merchant.

2.11.2 Email & SMS

Notifications are sent via SMS Gateway and Email Gateway, keeping the user informed of the transaction status, confirmation, or failure.

2.12. Logging and Auditing

2.12.1 Action

All activities within the payment process are logged for security, auditing, and troubleshooting purposes.

2.12.2 API Call

The Logging Service stores detailed logs of each action, including transaction initiation, validation, status updates, settlement, and refunds.

Logs are essential for system health monitoring, fraud detection, and compliance audits.

2.13 Data Security and Compliance

2.13.1 Action

Throughout the entire payment process, sensitive data such as payment information is securely handled according to relevant data protection regulations (e.g., PCI DSS).

2.13.2 Technology

- Encryption and Key Management Service (KMS) are used to protect sensitive data during both transmission and storage.

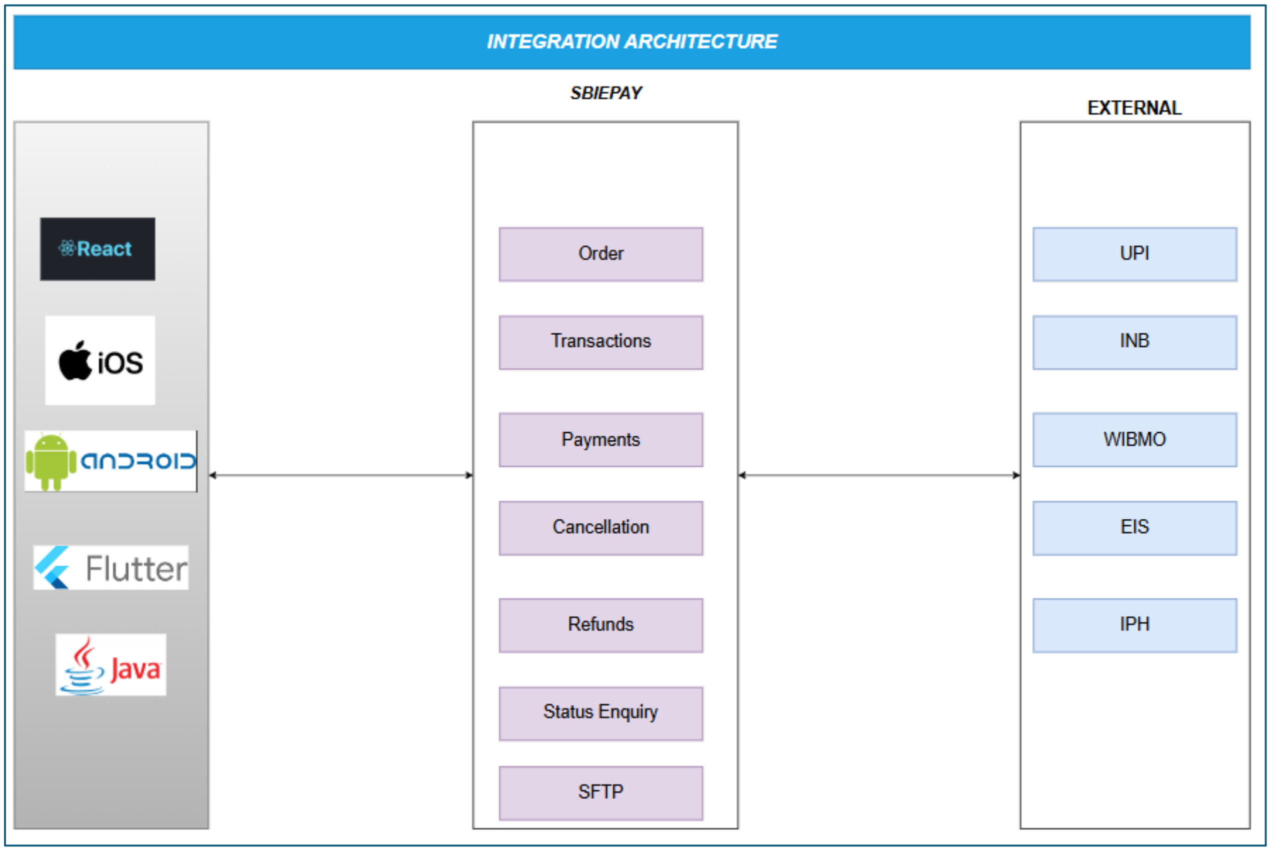
This breakdown outlines the various steps involved in the ePay 2.0 payment process, detailing how different service APIs and systems work together to handle transactions securely and efficiently. The process ensures that each transaction is validated, processed, settled, and reported, while keeping users and merchants informed at every stage.

3. SBI ePay Application Architecture

Application Architecture diagram Descriptions are as follows:

3.1 SBI ePay 2.0 Integration Architecture

Kindly relate each diagram with the architectural diagram *integration_architeture.drawio.html*



In any transaction, the first interaction originates from the frontend UI, which then calls the relevant backend service to process the request. Depending on the platform and technology being used, the respective service API will be called.

For example, as illustrated in the diagram, the frontend UI utilizes various technologies to interact with the SBIEPAY services:

3.1.1 Java and React

For users on web platforms, the frontend UI built with Java and React will communicate with the SBIEPAY services as required. The Java-based backend and React frontend seamlessly call the respective service APIs to complete the transaction flow.

3.1.2 iOS and Flutter

For mobile users on iOS devices, the frontend UI developed with Flutter (or native iOS components) will invoke the SBIEPAY services to carry out transactions. The service calls are designed to work with the iOS platform using the appropriate API integrations.

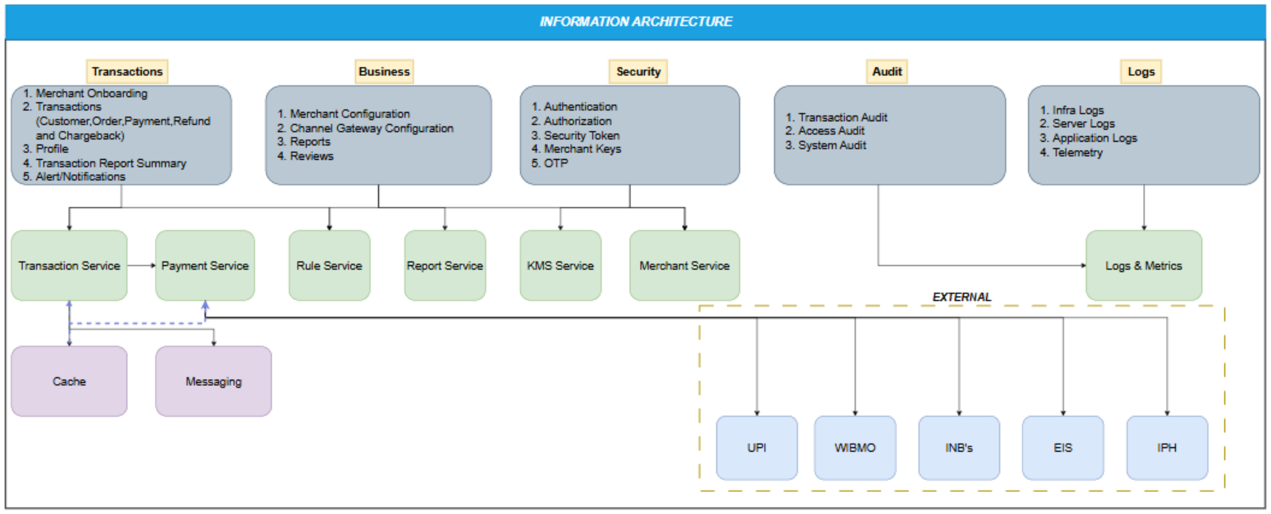
3.1.3 Android

Android users will interact with the SBIEPAY services through the frontend UI built for the Android platform, utilizing the Android-specific technologies to initiate service calls.

Each of these platforms — Java/React for web, Flutter/IOS for iOS, and native Android — uses the respective service calls tailored to their respective technologies, ensuring seamless integration with SBIEPAY services across multiple environments.

Once the SBIEPAY services have initiated communication with the frontend, external third-party services are engaged to handle the subsequent stages of transaction processing. These third-party services may include payment gateways, fraud detection systems, banks, or other financial institutions that provide specialized functionality required to complete the transaction. The SBIEPAY system acts as an intermediary, ensuring seamless integration and data flow between the frontend, SBIEPAY services, and the external services, facilitating a smooth and secure transaction process from start to finish.

3.2 SBI ePay 2.0 Information Architecture



The diagram above outlines the key business functionalities and their respective services involved in transaction processing. Each functionality utilizes specific service APIs to ensure smooth execution and seamless interaction across systems.

3.2.1 Working of Transaction and Payment Service APIs

The following functionalities will interact with the Transaction Service API and Payment Service API during processing:

3.2.1.1 Merchant Onboarding

This process involves registering and verifying merchants who will be accepting payments through the platform.

3.2.1.2 Transactions

The core transaction processing functionality, which includes initiation, authorization, and settlement of payments.

3.2.1.3 Profile Management

This includes the creation, updating, and management of user or merchant profiles for transaction-related activities.

3.2.1.4 Transaction Report Summary

This generates reports summarizing transaction activities, statuses, and other relevant metrics for analysis and auditing.

3.2.1.5 Alert/Notifications

This covers real-time notifications and alerts for users or merchants regarding transaction updates, approvals, and statuses.

During the transaction processing, both the Transaction Service and Payment Service APIs will interact with the Cache Server for data retrieval and performance optimization, as well as with Notification Services to ensure timely communication of transaction updates to users.

3.2.1.6 Role of Third-party Services

Additionally, the Payment Service APIs will call various third-party external services as part of the payment processing workflow. These third-party services include:

- UPI (Unified Payments Interface): For processing UPI-based transactions
- WIBMO: For payment gateway services or other integrations
- INB (Internet Banking): For transactions involving online banking services
- EIS (Enterprise Integration Services): For enterprise-level integrations supporting financial transaction flows
- IPH (Integrated Payment Hub): For connecting to different payment networks and ensuring interoperability

These third-party services play a crucial role in the payment ecosystem, enabling SBIEPAY to support a wide range of transaction types and payment methods.

3.2.2 Working of Business Service APIs

The following functionalities will interact with the Business Service API to perform their respective processing tasks:

3.2.2.1 Merchant Configuration

This functionality manages the setup and configuration of merchant accounts, including parameters such as payment methods, fees, and service preferences.

3.2.2.2 Channel Gateway Configuration

This involves configuring the various payment channels and gateways through which transactions will be processed, ensuring integration with multiple payment providers.

3.2.2.3 Reports

This generates detailed reports on various aspects of the business, including transaction summaries, revenue, and other key performance indicators.

3.2.2.4 Reviews

This manages customer or merchant reviews, feedback, and ratings, typically linked to transactions or service quality.

During transaction processing, the Business Service API will communicate with the Rule Service to apply business rules and validations, ensuring compliance with policies and ensuring transactions are processed according to predefined criteria.

Additionally, it will interact with the Report Service to generate or update reports based on transaction data, configurations, and other relevant business activities.

This integration ensures that the business logic is applied consistently across different processes, while also enabling effective reporting and compliance through interaction with the appropriate services.

3.2.3 Working of Security Service APIs

The following functionalities will interact with the Security Service API to ensure secure transaction processing:

3.2.3.1 Authentication

Verifies the identity of users or merchants to ensure they are who they claim to be before granting access to the system.

3.2.3.2 Authorization

Determines the level of access and permissions a user or merchant has within the system, ensuring that they can only perform actions within their allowed scope.

3.2.3.3 Security Token

Issues and manages security tokens used for secure session handling and to authenticate API requests.

3.2.3.4 Merchant Keys

Manages and secures encryption keys used by merchants for secure communication and transaction processing.

3.2.3.5 OTP (One-Time Password)

Generates and validates time-sensitive one-time passwords for additional security during login or transaction authorization.

During transaction processing, the Security Service API will interact with the KMS (Key Management Service) to securely manage encryption keys and other sensitive information, ensuring data protection and compliance with security standards.

Additionally, the API will interact with the Merchant Report Service to ensure that any security-related events, such as failed authentication or authorization attempts, are logged and reported for auditing and compliance purposes.

This integration of security functionalities provides a robust layer of protection, safeguarding user and merchant data throughout the transaction process.

3.2.4 Working of Audit Service APIs

The following functionalities will interact with the Audit Service API to ensure comprehensive auditing and tracking of activities within the system:

3.2.4.1 Transaction Audit

Tracks and records all transaction-related activities to ensure compliance, traceability, and transparency in payment processing.

3.2.4.2 Access Audit

Monitors and logs user and merchant access to the system, tracking who accessed what information and when, to ensure proper access control and security.

3.2.4.3 System Audit

Logs system-level activities, such as configuration changes, service interactions, and other administrative actions, to maintain accountability and support troubleshooting.

3.2.5 Working of Logs and Services

The Audit Service API will integrate with the Logs and Metrics Services during transaction processing to collect detailed information and generate audit trails. These services ensure that relevant data is captured and available for monitoring, analysis, and reporting.

Additionally, the following functionality will also leverage the Audit Service API for the management and tracking of system logs:

3.2.5.1 Infra Logs

Captures infrastructure-level logs, including network, server, and database activities, ensuring system stability and identifying potential issues.

3.2.5.2 Server Logs

Monitors and logs events at the server level, tracking system performance, errors, and other critical operational events.

3.2.5.3 Application Logs

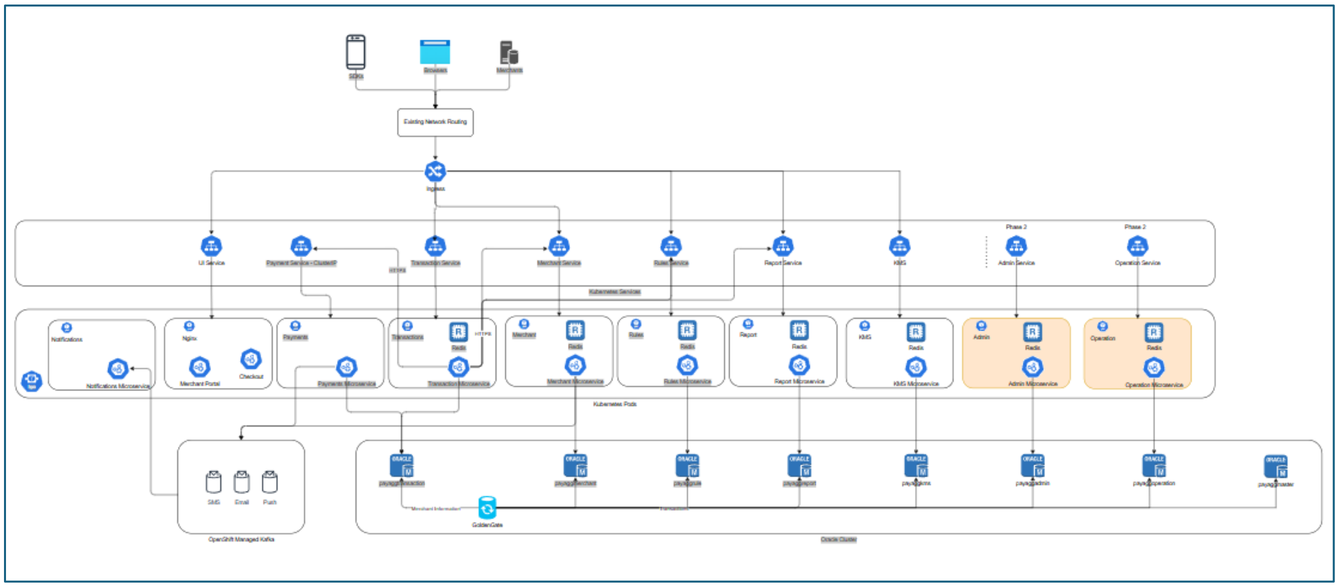
Records logs related to application behavior, performance, errors, and interactions with other services.

3.2.5.4 Telemetry

Collects and transmits real-time metrics, performance data, and usage statistics, providing insights into system health and user behavior.

These services interact with the Logs and Metrics Services to provide comprehensive visibility into system activities and ensure that all critical events are logged for auditing, analysis, and troubleshooting.

3.3 SBI ePay 2.0 Architecture Diagram



There are three primary types of users interacting with the system:

- Merchants
- Users using Browsers
- SDK Users

Each of these users interacts with the various service APIs through the existing Network Routing Ingress, ensuring a smooth flow of data across the platform.

3.3.1 Service Interactions and Microservices

The following outlines the key interactions between different microservices, their databases, and how each service communicates within the system:

3.3.1.1 Merchant Portal Interaction

Merchants access the UI Service API via the Merchant portal. During this interaction, the Notification Micro-service API is triggered to send alerts and updates to the merchant.

3.3.1.2 Payment Microservices

The Payment Microservices are accessed through the Payment Service ClusterIP.
Both Payment Microservices and Transaction Microservices interact with the Oracle Cluster for transaction and payment data. Specifically, the payment-related data is stored in the payaggtransaction database.

3.3.1.3 Transaction Microservices

The Transaction Microservice, provided by Redis, is responsible for processing transactions. It interacts with the Transaction Service and the Oracle Cluster, specifically the payaggtransaction database, to store transaction data.

3.3.1.4 Merchant Microservices

The Merchant Microservice, also provided by Redis, is used by the Merchant Service to handle merchant-specific operations and data.
The Merchant Microservice interacts with the Oracle Cluster, accessing data in the payaggmerchant database.

3.3.1.5 Payment and Merchant Service Interaction

The Payment Service and Merchant Service communicate with each other using the HTTP protocol, enabling seamless data exchange between the two services.

3.3.1.6 Rules Microservices

The Rules Microservice, provided by Redis, is responsible for enforcing business logic and rules within the system.

The Rules Microservice interacts with the Oracle Cluster, where the rule data is stored in the payaggrule database.

3.3.1.7 Report Microservices

The Report Microservices, provided by Redis, are used by the Report Service to generate reports based on transaction and payment data.

3.3.1.8 Report Microservices

The Report Microservice interacts with the Transaction Microservice to obtain transaction references when generating reports.

The report data is stored in the Oracle Cluster within the payaggreport database.

3.3.1.9 KMS (Key Management Service)

The KMS Microservices, provided by Redis, are used by the KMS service for managing encryption keys and securing data within the platform.

The KMS Microservice accesses the Oracle Cluster for key storage, specifically in the payaggkms database.

3.3.2 Admin and Operations Services

In this new version, the Admin Service will interact with the Admin Microservices, provided by Redis, to manage administrative functions. This service will access the Oracle Cluster and the payaggadmin database to retrieve necessary data.

Similarly, the Operations Service will use the Operations Microservices provided by Redis to interact with the Oracle databases (payaggoperation and payaggmaster) to fetch operational data.

3.3.2.1 Data Migration to New Database

Finally, in the data migration phase, all relevant data from the existing databases will be transferred to the new database environment using GoldenGate, ensuring seamless data migration without system downtime.

3.3.3 Interaction between Micro-services and Database

3.3.3.1 Redis Microservices

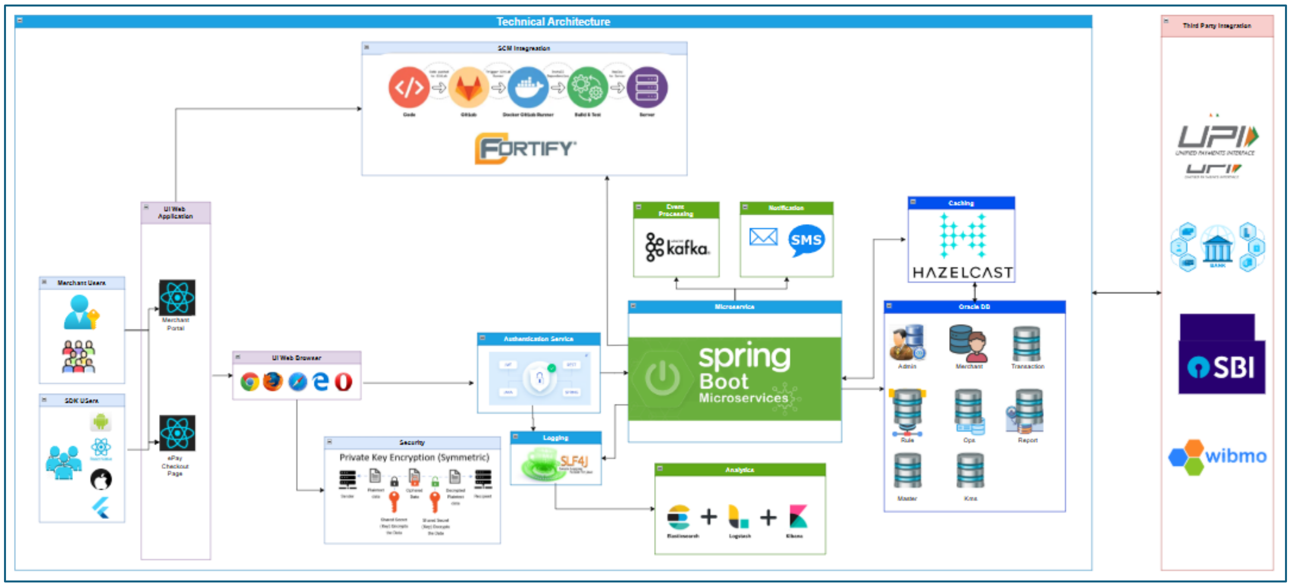
- Merchant Microservices: Interacts with payaggmerchant
- Transaction Microservices: Interacts with payaggtransaction
- Rules Microservices: Interacts with payaggrule
- Report Microservices: Interacts with payaggreport
- KMS Microservices: Interacts with payaggkms
- Admin Microservices: Interacts with payaggadmin
- Operations Microservices: Interacts with payaggoperation and payaggmaster

3.3.3.2 Oracle Cluster Databases:

Each service communicates with its designated database in the Oracle cluster, ensuring data is consistently updated and retrieved.

This revised version presents a clearer and more structured breakdown of the interactions and workflows between users, microservices, and databases. It also emphasizes how the different services interact through Redis and Oracle clusters, along with the key migration process to the new database using GoldenGate.

3.4 SBI ePay 2.0 Technical Architecture



Merchant and SDK Users interact with the UI Merchant Portal and ePay Checkout page through the UI Web Application, which supports a variety of web browsers, including:

- Google Chrome
- Mozilla Firefox
- Internet Explorer
- Opera

These web browsers will continuously interact with the Security Services for authentication purposes, ensuring that each user is properly verified before accessing sensitive resources.

3.4.1 Data Encryption and Decryption

When a sender enters data in plain text format, the data is encrypted using a Shared Secret (Key).

The encrypted data, or cipher text, is then transmitted securely and, upon reaching the recipient, is decrypted back into its original plain text format before being shared with the recipient.

This encryption and decryption process ensures data security and confidentiality during transmission between users, systems, and services.

3.4.2 Authentication and Logging

The Authentication Services play a critical role in verifying user identities. These services communicate with:

3.4.2.1 SLF4J (Logging)

For logging authentication activities, user interactions, and system events

3.4.2.2 CLK (Analytics)

For gathering and processing user and system analytics to track authentication patterns and behaviors

3.4.2.3 Spring Boot Microservices

Other backend services that rely on authentication to grant access to various resources and ensure secure communication

3.4.2.4 Event Processing and Notification Services

The Spring Boot Microservices ecosystem utilizes Apache Kafka for efficient event processing, ensuring real-time data streaming and communication between different system components

3.4.2.5 Email and SMS Notification Services

Email and SMS Notification Services are integrated to send alerts and updates to users or merchants when specific actions occur (e.g., transaction status updates, and account notifications)

3.4.2.6 Cache and Database Interactions

The Spring Boot Microservices will interact with both the Cache Server (Hazelcast) and the Oracle Database to handle various entities and provide fast data retrieval. The services will interact with the following data stores:

3.4.2.6.1 Admin

For managing administrative users and settings

3.4.2.6.2 Merchant

For storing merchant-specific data, such as payment configurations and profiles

3.4.2.6.3 Transaction

For recording transaction data, including payment details and status

3.4.2.6.4 Rule

For managing business rules and validation logic

3.4.2.6.5 Ops

For operational data related to the system's health and performance

3.4.2.6.6 Report

For storing and retrieving report data, including transaction summaries and analytics

3.4.2.6.6.1 Master

For storing foundational data that supports core system operations

3.4.2.6.6.2 KMS (Key Management Service)

For securely managing encryption keys and other sensitive information

3.4.3 SCM Integration and Continuous Delivery Pipeline

Following information provides SCM Integration and its continuous support for the delivery components.

3.4.3.1 Microservices Supporting Internal Processes

The Spring Boot Microservices leverage Fortify for Source Code Management (SCM) Integration, supporting the following internal processes:

3.4.3.1.1 Code Push

Developers push the latest code to the GitLab Server.

3.4.3.1.2 GitLab Runner Trigger

A trigger in GitLab Runner activates the process.

3.4.3.1.3 Docker GitLab Runner

The build process is executed within a Docker GitLab Runner environment.

3.4.3.1.4 Internal Build & Testing

The system performs automated builds and application testing to ensure code quality and functionality.

3.4.3.1.5 Deployment

After successful testing, the application is deployed on the production server.

This continuous integration/continuous delivery (CI/CD) pipeline ensures that updates to the system are reliable, tested, and quickly deployed.

3.4.3.2 Integration with Third-Party Services

SBIEPAY 2.0 is designed to seamlessly integrate with a range of third-party services, enabling expanded functionality and support for various payment methods. These integrations include:

3.4.3.2.1 UPI (Unified Payments Interface)

For enabling fast, secure UPI-based transactions.

3.4.3.2.2 Wibmo

A third-party payment gateway service that facilitates card payments and other digital payment methods.

3.4.3.2.3 SBI (State Bank of India) and Other Banks:

For facilitating bank-level transactions and ensuring compatibility with various banking systems.

3.4.3.2.4 Financial Institutes

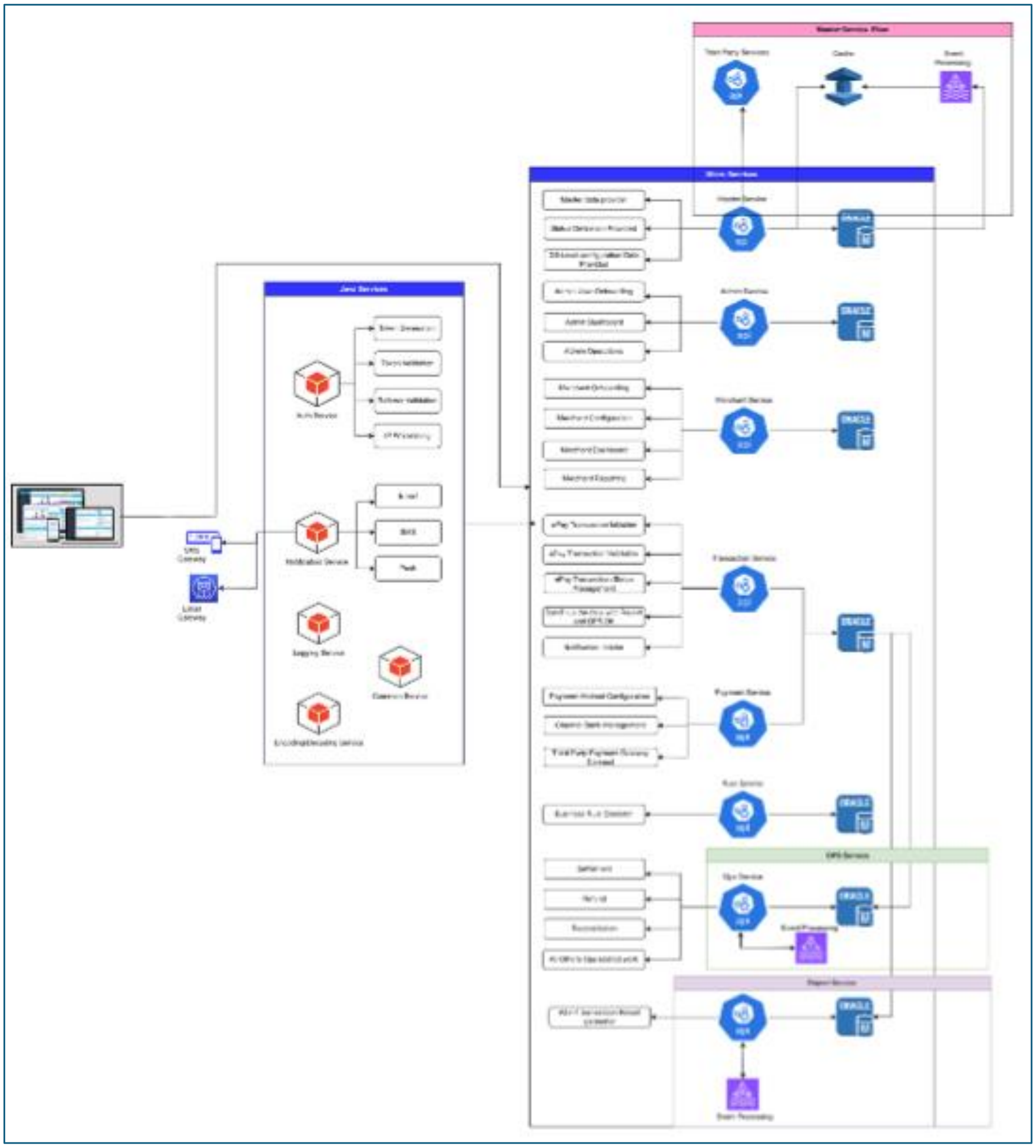
Integration with additional financial institutions to support diverse payment systems and enhance cross-platform transaction processing.

This flexible integration framework ensures that SBIEPAY 2.0 can provide a robust, scalable, and future-proof payment solution, supporting a wide variety of payment methods, financial services, and partners.

SBIEPAY 2.0 incorporates a comprehensive architecture designed for secure, efficient, and scalable transaction processing. From user authentication and data encryption to real-time event processing and integration with financial institutions, the platform leverages modern technologies, including Spring Boot Microservices, Hazelcast, Oracle DB, and Apache Kafka. Furthermore, the continuous delivery pipeline powered by Fortify and GitLab ensures that updates are seamlessly delivered while maintaining high standards of code quality. Integration with third-party services, such as UPI, Wibmo, and various banks, enables SBIEPAY 2.0 to offer a wide range of payment solutions for merchants and end-users alike.

This revised version is more structured, providing a clearer explanation of each component's role in the architecture and how they work together to deliver secure, reliable, and efficient services.

3.5 SBI ePay 2.0 Application Architecture



The following sequence of events occurs whenever a transaction is initiated. Each step involves a specific service API being called to perform a function, ensuring smooth transaction processing. The integration of these services and their respective APIs forms the core architecture of the application, facilitating efficient and secure transaction management.

3.5.1 Master Data Configuration and Update Activity

3.5.1.1 Master Data Provider

Responsible for supplying foundational data used across various modules in the system

3.5.1.2 Status Definition

Provides status definitions to categorize and track the state of various transactions and entities

3.5.1.3 Database-Level Configuration

Handles configuration at the database level, ensuring that the system's core configurations are accurately reflected in the data layer

3.5.2 Merchant Configuration

3.5.2.1 Merchant Onboarding

Involves registering and verifying merchants who wish to use the platform to accept payments.

3.5.2.2 Merchant Configuration

Defines the settings and parameters for each merchant, such as payment methods, fees, and preferences.

3.5.2.3 Merchant Dashboard

Provides merchants with a user interface to view transaction data, account details, and performance metrics.

3.5.2.4 Merchant Reporting

Generates reports on transaction activity, revenue, and other key performance indicators for the merchant.

3.5.3 Admin Management

3.5.3.1 Admin User Onboarding

Allows the onboarding and registration of administrators who will have access to manage the platform’s settings and operations.

3.5.3.2 Admin Dashboard

Provides administrators with a centralized view of system health, user activity, and key metrics.

3.5.3.3 Admin Operations

Facilitates operational tasks such as managing users, configurations, and system settings, ensuring smooth platform governance.

3.5.4 Authorization Service

3.5.4.1 Token Generation

Creates authentication tokens to verify the identity of users or systems interacting with the platform.

3.5.4.2 Token Validation

Validates tokens to ensure they are legitimate and have not expired, preventing unauthorized access.

3.5.4.3 Referrer Validation

Checks the referrer to ensure that the request comes from a valid and trusted source.

3.5.4.4 IP Whitelisting

Restricts access to the platform based on trusted IP addresses, enhancing security and preventing unauthorized access.

3.5.5 Transaction Service

3.5.5.1 ePay Transaction Initiation

Begins the process of a payment transaction by receiving payment details and initiating the payment flow.

3.5.5.2 ePay Transaction Validation

Validates the payment details, ensuring that the transaction is legitimate and complies with required rules.

3.5.5.3 ePay Transaction Status Management

Tracks and manages the various stages of the transaction, updating the status as the payment progresses.

3.5.5.4 Sync Data with Report and Ops DB:

Ensures that transaction data is synchronized with the Report and Operations databases, providing accurate and up-to-date records for reporting and operational tasks.

3.5.5.5 Notification Initiator

Triggers notifications to users and merchants regarding transaction status, updates, and other relevant events.

3.5.6 Ops Service with Event Processing

3.5.6.1 Settlement

Processes the final settlement of funds between the payer and payee.

3.5.6.2 Refund

Handles refund requests, ensuring that funds are returned to the payer if necessary.

3.5.6.3 Reconciliation

Ensures that transaction records match between various systems (e.g., payment gateways, banks) and identifies any discrepancies.

3.5.6.4 Other Operations-Related Tasks:

Covers all other operational activities related to transaction processing, including dispute resolution, payment adjustments, and system monitoring.

3.5.7 Notification Service

3.5.7.1 SMS Gateway

Sends transaction-related alerts and notifications to users via SMS.

3.5.7.2 Email Gateway

Sends transaction-related alerts and notifications to users via email.

3.5.8 Common Service

3.5.9 Logging Service

Tracks all interactions with the system, logging transaction details, errors, and system events to support troubleshooting and auditing.

3.5.10 Report Service

Transaction-Related Report Generation: Generates reports related to transaction activity, merchant performance, user behavior, and financial summaries, providing actionable insights to both merchants and administrators.

3.5.11 Rule Service

3.5.11.1 Business Rules and Validation:

Ensures that transactions and operations adhere to predefined business rules and validation criteria, ensuring compliance and consistency in the system's operations.

3.5.11.2 Encoding/Decoding Service

3.5.11.2.1 Data Security

Handles the encoding and decoding of sensitive data, such as payment information, ensuring that it is securely transmitted and stored.-

3.5.12 Data Storage and Oracle Integration

All of the above services interact with the Oracle Database for data storage, retrieval, and synchronization.

The following entities are managed and stored in the database:

3.5.12.1 Admin

Stores admin user data, roles, and system configuration

3.5.12.2 Merchant

Stores merchant data, including registration details, payment configurations, and transaction history

3.5.12.3 Transaction

Stores transaction data, including payment details, transaction status, and associated records

3.5.12.4 Rule

Stores business rules, validations, and configurations

3.5.12.5 Ops

Stores operational data related to system health, performance, and issue tracking

3.5.12.6 Report

Stores generated reports, transaction summaries, and analytics data

3.5.12.7 Master

Stores core configuration data used across the platform

3.5.12.8 KMS

Stores encryption keys and other sensitive security data

The seamless interaction between these service APIs and their respective functions ensures a robust and efficient transaction processing ecosystem. Each service is responsible for a specific set of tasks, whether it's managing merchant onboarding, validating transactions, processing payments, or generating reports. The entire architecture is integrated with the Oracle Database to store and manage data, ensuring that the system remains consistent, secure, and scalable. By leveraging a combination of core services, APIs, and databases, the platform provides a reliable and comprehensive solution for merchants and end-users alike.

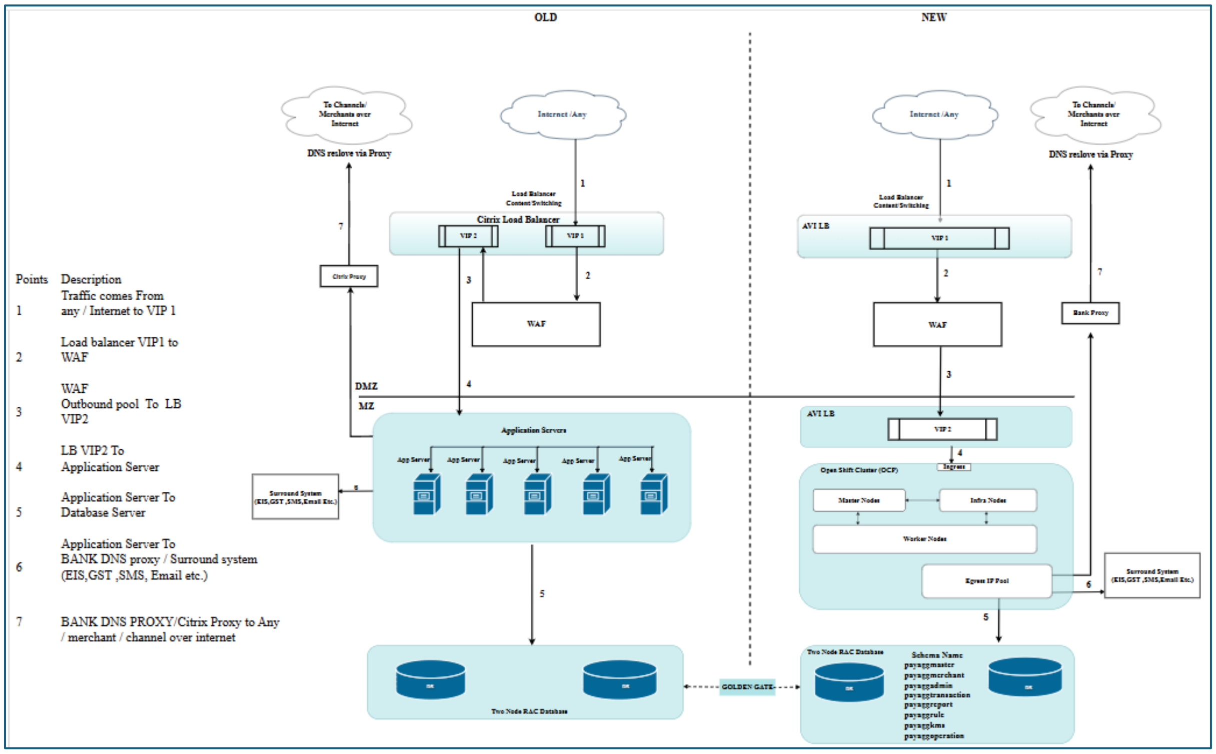
4. SBI ePay 2.0 Network Architecture

SBI ePay 2.0's network architecture is built to provide a robust, scalable, and secure platform for processing online payments. The system follows a multi-layered approach, utilizing cutting-edge technologies and frameworks to manage the various demands of modern payment processing, including real-time transactions, load balancing, and data security.

Architectural, OCP, and Network diagram Descriptions are as follows:

Kindly relate each diagram with the *High-Level Network Diagram (Old and New OCP Environments)*

4.1 High-Level Network Diagram (Old and New OCP Environments)



Points	Description
1	Traffic comes From any / Internet to VIP 1
2	Load balancer VIP1 to WAF
3	WAF Outbound pool To LB VIP2
4	LB VIP2 To Application Server
5	Application Server To Database Server
6	Application Server To BANK DNS proxy / Surround system (EIS,GST ,SMS, Email etc.)
7	BANK DNS PROXY/Citrix Proxy to Any / merchant / channel over internet

Please note that the following diagrams will serve as detailed illustrations of each component described in the cluster architecture. These diagrams will visually represent the roles, interactions, and responsibilities of each element within the system, helping to clarify the structure and flow of traffic across the cluster.

4.1.1.1 Overview of Traffic Flow and Network Architecture in OLD OCP Environment

Traffic originating from the internet is directed towards physical devices, specifically the **Citrix Load Balancer**, which plays a pivotal role in managing and distributing incoming requests across various systems.

4.1.1.1 VIP1 (Virtual IP 1)

VIP1 is designated for **incoming internet traffic**. This IP address is used primarily for managing and filtering external traffic before it reaches the internal network. VIP1 ensures that only legitimate traffic enters the environment, serving as a point of entry where security policies are enforced.

4.1.1.2 VIP2 (Virtual IP 2)

VIP2 acts as a **forward proxy**, specifically a **Citrix proxy**. It is used to facilitate communication between internal systems and external services. VIP2 ensures that internal devices do not directly expose their identities to the outside world, offering an additional layer of security.

4.1.1.3 Bank Proxy

The **Bank Proxy** functions as a **firewall** in this architecture, effectively managing and filtering all traffic between the internal network and external sources. This device ensures that only authorized and secure traffic is allowed through, preventing potential threats from reaching the sensitive systems inside the organization.

4.1.1.4 Internal Application Server

The **Internal Application Server** serves as the core engine for processing business logic and interacting with other internal services. It communicates directly with **databases** and surrounding systems that facilitate data storage and processing. These systems are responsible for distributing the traffic further to various channels and applications based on the requirements of the organization.

4.1.2 Traffic Flow and Network Architecture in the New OCP Environment:

In the **New OCP Environment**, traffic from the external internet is directly routed to the **WAF (Web Application Firewall)**, which is the bank's security device. Unlike the previous setup, there are no Citrix devices involved in this flow. The WAF acts as the first line of defense, filtering and securing incoming traffic.

4.1.2.1 AVI Load Balancer

From the WAF, traffic is forwarded to the **Software Load Balancer (AVI LB)**. The AVI Load Balancer operates in a **two-layer architecture**:

- The **upper layer**, known as the **DMZ (Demilitarized Zone)**, handles traffic that interacts with external sources.
- The **lower layer**, called the **MZ (Management Zone)**, is where traffic is directed after passing through the DMZ for further processing.

4.1.2.2 Layers in Server Traffic

Once traffic reaches the AVI LB, it is directed to an **internal application server**. This server processes traffic across multiple layers:

- **Master Nodes**: Responsible for orchestrating tasks
- **Infra Nodes**: Handle infrastructure management
- **Worker Nodes**: Perform the actual processing and task execution

From the internal application server, traffic is sent to the **Database** and **Surround Systems** for storage and processing. This communication ensures that internal systems are continuously updated and can perform required operations efficiently.

In contrast to the old environment, the **traffic and data** from the internal systems are now forwarded by the **Bank Proxy Server** to various channels for further distribution to their respective destinations. This improves the scalability and management of traffic as it moves through different layers of the architecture.

4.1.3 Key Differences between Old and New Environments:

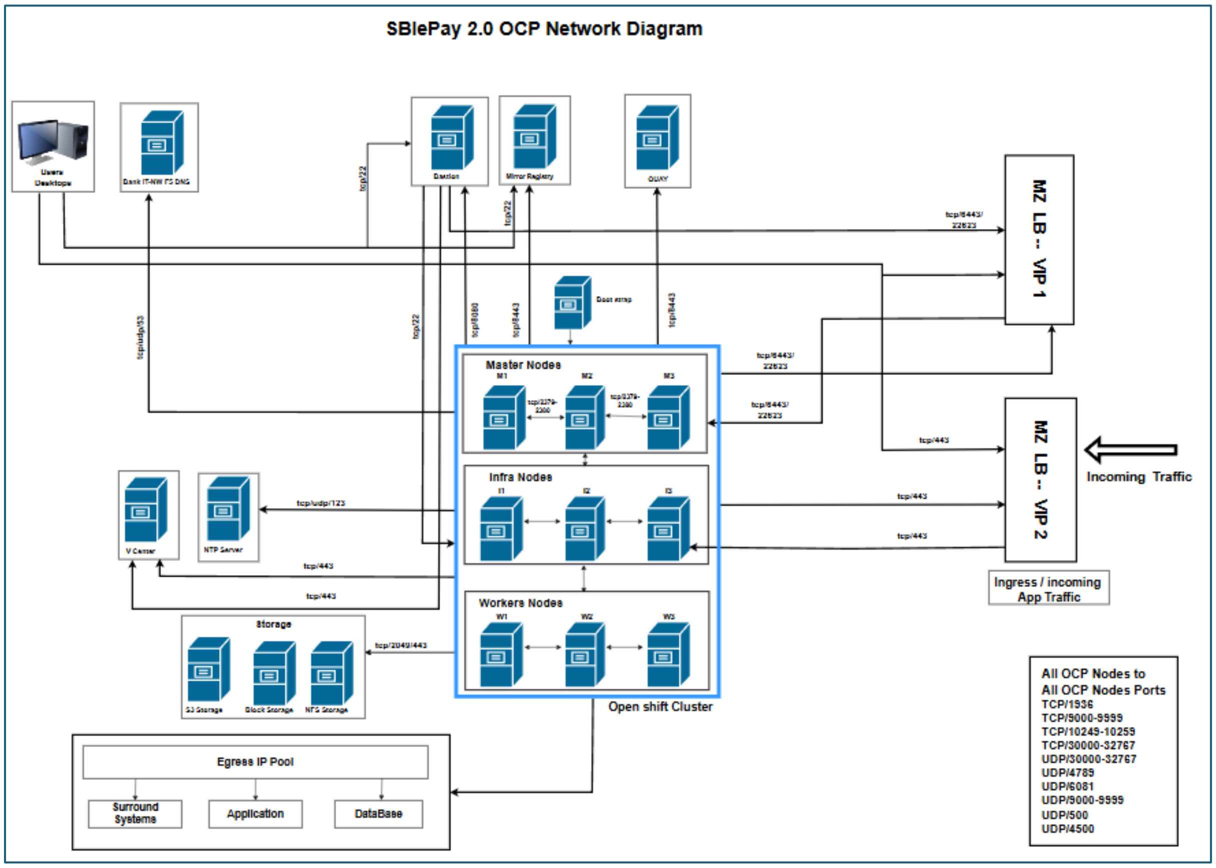
- In the **old environment**, the setup included **2-node clusters** for handling traffic and processing.
- In the **new environment**, the architecture is more scalable and segmented, with **3-node clusters** and **8 database instances** for improved performance, redundancy, and fault tolerance.

The **Old** and **New environments** are connected by a **bridge called Golden Gate**, which enables data migration and synchronization between the two. The **Golden Gate** ensures that data from the old environment is seamlessly transferred to the new one, ensuring continuity and minimal disruption during the migration process.

This version is more concise, with clear breakdowns of each component's function and differences between the old and new environments. It also improves readability and flow, making it easier to understand the transition and system architecture.

4.2 SBI ePay 2.0 OCP Network Diagram

Kindly relate the following diagram with the *High-Level Network Diagram (Old and New OCP Environments)*



In the segregated cluster configuration, there are three distinct types of nodes, namely **Master Nodes**, **Infra Nodes**, and **Worker Nodes**, with three instances of each type for redundancy and load balancing. The cluster architecture utilizes several key components working together, each with dedicated responsibilities to ensure optimal performance, security, and reliability.

4.2.1 Key Components and Responsibilities

4.2.1.1 Web Application Firewall (WAF):

- The WAF serves as the first line of defense against external internet traffic, ensuring that malicious or unauthorized traffic is filtered before reaching the internal network.
- External traffic is routed directly to the WAF through **VIP1** (Virtual IP 1), which is responsible for handling all management traffic related to cluster operations.

4.2.1.2 VIP1 and VIP2 (Virtual IPs)

- **VIP1** handles traffic that is related to the **management of the cluster**, including communication with the Master Nodes. It ensures that management functions, such as configuration changes and monitoring, are performed securely and efficiently.
- **VIP2** is responsible for handling **transactional and incoming traffic**, which includes processing data and managing service requests within the cluster.

4.2.1.3 Egress IP Pool

- The **Egress IP Pool** is used for **internal processing and outbound communication**. It is specifically used by the Infra Nodes to manage internet-bound traffic, ensuring that external services are reachable for internal processing tasks.
- Internal cluster operations, such as **communication with NTP servers** (Network Time Protocol) for time synchronization and the **V Center** for cluster management and monitoring, are facilitated by the Egress IP Pool.

4.2.1.4 Infra Nodes

- The **Infra Nodes** play a crucial role in managing the cluster infrastructure, including network connectivity, monitoring, and coordination of services. These nodes are responsible for managing virtual machines and hosting critical services.

- The Infra Nodes interact with the **Egress services** to enable internet-bound traffic and support internal processing tasks like time synchronization and communication with other external systems.

4.2.1.5 Database Servers

- Database servers are deployed as part of the infrastructure to handle persistent data storage for the cluster's operations. These servers ensure that data is securely stored and accessible for cluster applications.

4.2.1.6 V Center

- The **V Center** provides a graphical interface for monitoring and managing the entire cluster environment. It enables administrators to visually manage the nodes, services, and overall health of the cluster, ensuring smooth operations and timely troubleshooting when necessary.

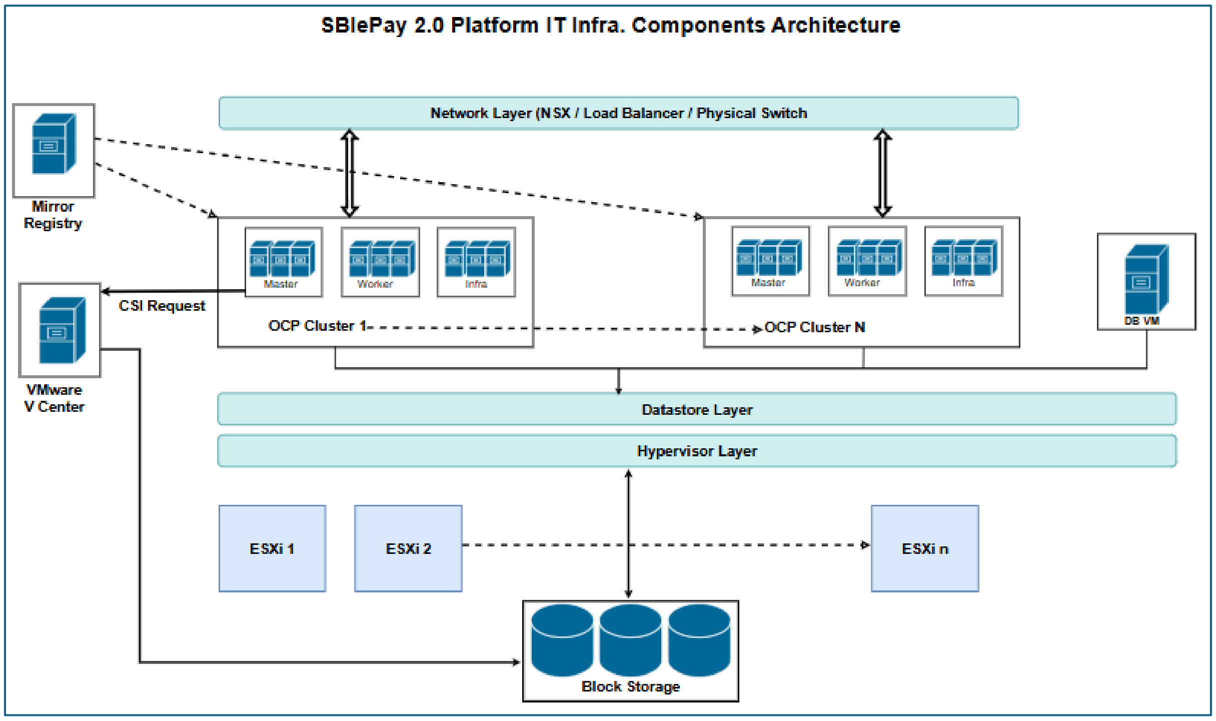
4.2.1.7 Internal Cluster Communication:

- Internal communication between the nodes and other services, such as the **NTP server** for time synchronization and **V Center** for cluster management, is facilitated by the Egress IP Pool. This ensures that all cluster nodes are properly synchronized and maintained.

This clustered architecture leverages a multi-tier approach where different components serve distinct purposes to ensure optimal traffic flow, security, and performance. VIP1 handles management tasks, while VIP2 focuses on transaction-related traffic. The Egress IP Pool enables secure internal processing and interaction with external services, while the Infra Nodes manage the infrastructure and maintain communication with critical services like the NTP server and V Center for cluster management.

4.3 IT Infrastructure Components Architecture

Kindly relate the following diagram with the *High-Level Network Diagram (Old and New OCP Environments)*



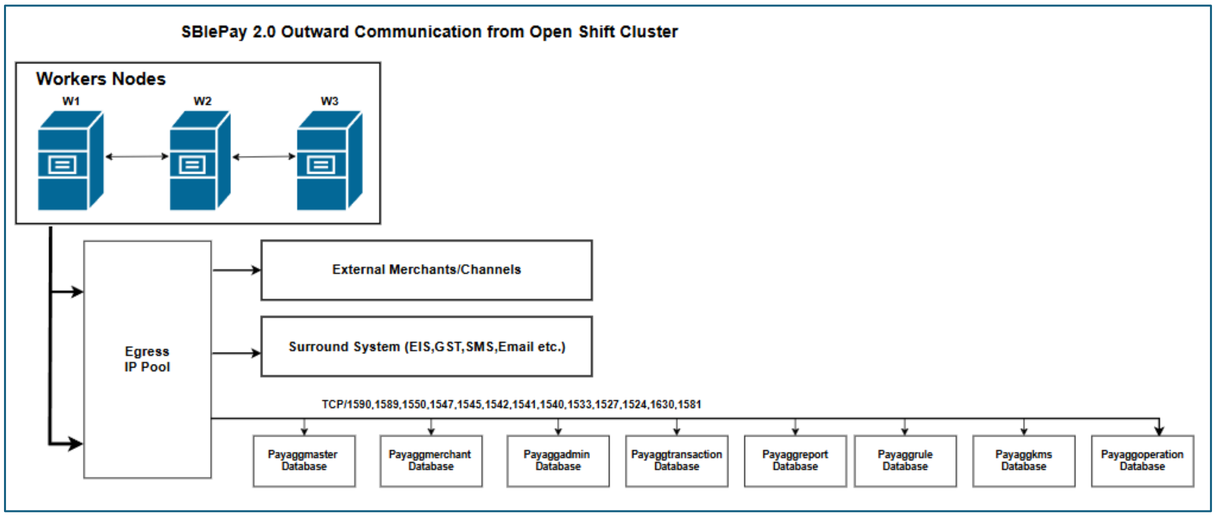
The incoming data traffic is processed across multiple layers within the system, including the Network layer, Datastore layer, and Hypervisor layer, through a series of OCP clusters. The number of virtual machines is configured and managed through the Datastore and Hypervisor layers, with the process being overseen by Meghdoot.

These virtual machines, along with the associated block storage, are effectively managed by VMware vCenter, ensuring seamless operation and monitoring of the virtualized environment.

A set of clusters (denoted as c1, c2, c3, ..., Cn) are configured and managed by the Mirror Registry. The Mirror Registry is responsible for overseeing the entire system's graphical interface, as well as managing the images and configurations associated with the clusters, providing a centralized control point for the infrastructure.

4.4 SBIePay 2.0 Outward Communication from Open Shift Cluster

Kindly relate the following diagram with the High-Level Network Diagram (Old and New OCP Environments)

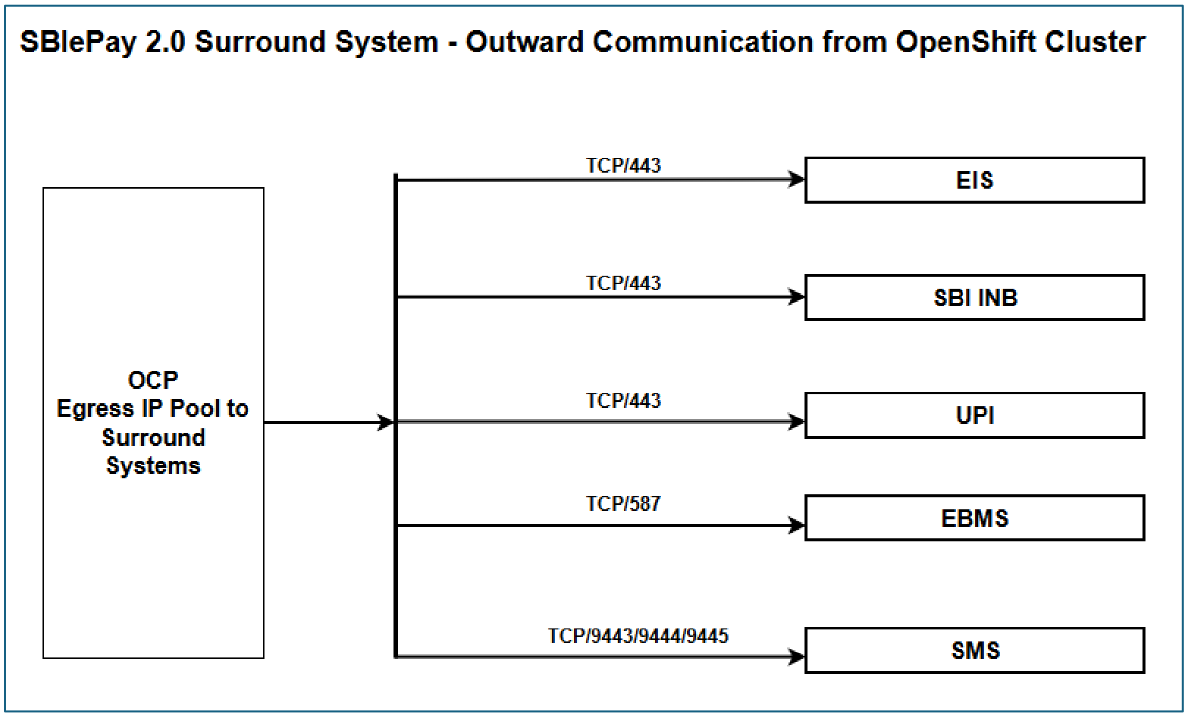

SBIePay 2.0 Outward Communication from Open Shift Cluster
Workers Nodes
W1
W2
W3
Egress IP Pool
External Merchants/Channels
Surround System (EIS,GST,SMS,Email etc.)
TCP/1590,1589,1550,1547,1545,1542,1541,1540,1533,1527,1524,1630,1581
Payaggmaster Database
Payaggmerchant Database
Payaggadmin Database
Payaggtransaction Database
Payaggreport Database
Payaggrule Database
Payaggkms Database
Payaggoperation Database

In the segregated cluster, there are three types of nodes: the Master Node, Infra Node, and Worker Node, with each type having a total of three instances. During internal processing, the Worker Nodes establish communication with the Egress IP Pool.

This Egress IP Pool acts as an intermediary, facilitating communication between the internal cluster and various external entities, including Merchants/Channels, Surround Systems (such as EIS, GST, SMS, Email), and a total of eight dedicated databases. Each database is specifically allocated to its respective storage, ensuring efficient and organized data management.2

4.5 SBIePay 2.0 Surround System to Outward Communication from Open Shift Cluster

Kindly relate the following diagram with the *High-Level Network Diagram (Old and New OCP Environments)*


SBIePay 2.0 Surround System - Outward Communication from OpenShift Cluster
OCP Egress IP Pool to Surround Systems
TCP/443
EIS
TCP/443
SBI INB
TCP/443
UPI
TCP/587
EBMS
TCP/9443/9444/9445
SMS

When communication occurs from an Egress IP Pool to surrounding systems (typically other network services or devices), it can involve several TCP ports. These ports often depend on the specific type of system or service being communicated with. Below are some common TCP ports that might be involved in such communication:

TCP Port Number	**Channel / Service**
TCP/443	Dedicated to EIS for the Loan purpose
TCP/443	Dedicated to SBI INB for SBI Internet Banking
TCP/443	Dedicated UPI for SBI UPI
TCP/587	Dedicated to EBMS for SBI Email services
TCP/9443/9444/9445	Dedicated to SMS for SBI SMS services