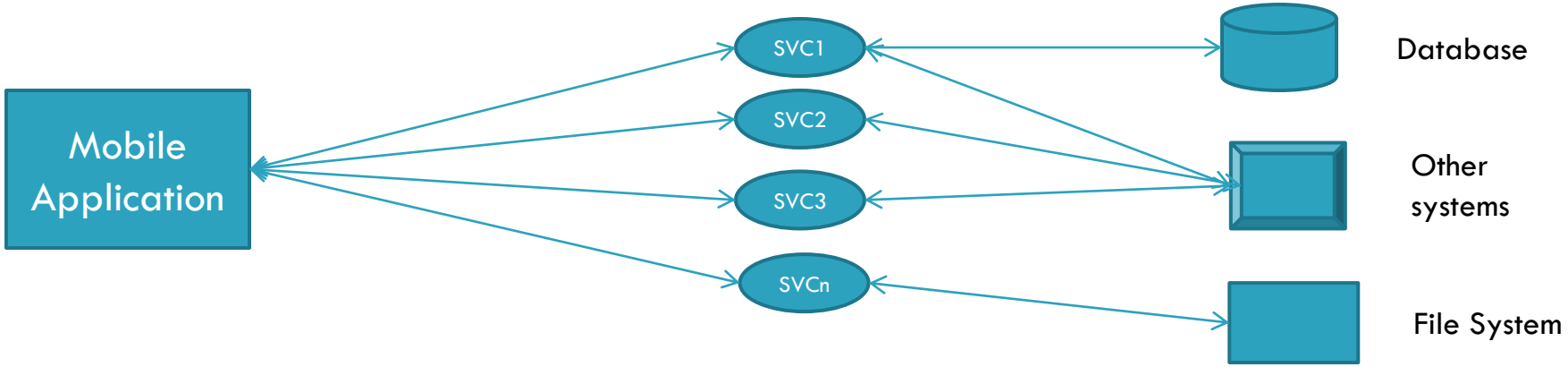
DANSKE IT – CASE STUDY

CS20180505 – DEF Mobile Bank

Existing Design



- Services are exposed as SOAP based services
- Direct point to point integration between mobile application and the services
- Technical services and hence complexity in understanding and consumption
- Architecture is not scalable to handle additional load

Business Requirement

Functional Requirements

- Enable internal mobile applications to consume APIs for the business needs
- Exposure of APIs to partner community for consumption
- Enable extension of mobile application usage from existing two countries to five more countries and from two million user base to six million

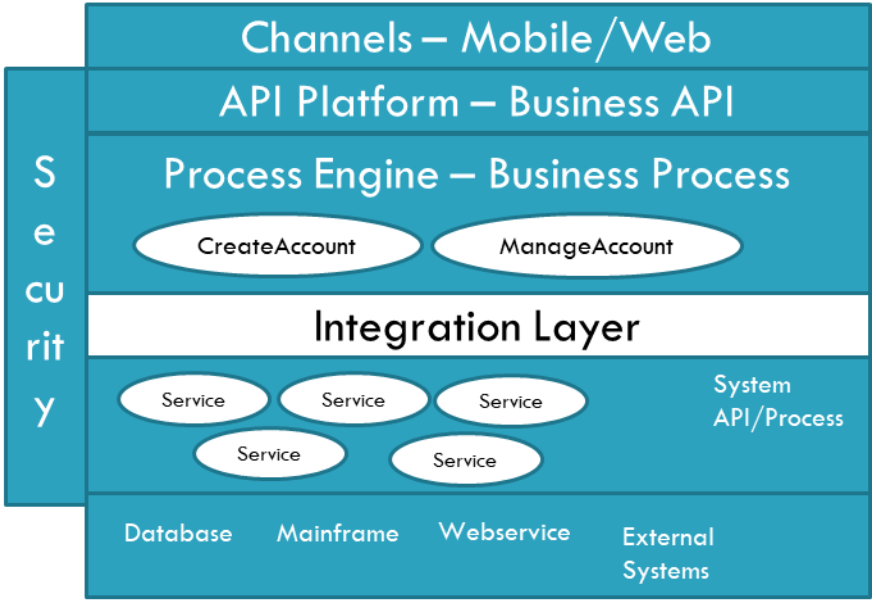
Non functional Requirements

- Scalable Architecture
- High availability, low latency and resiliency
- Additional Security and Data handling requirements as per compliance and data protection laws

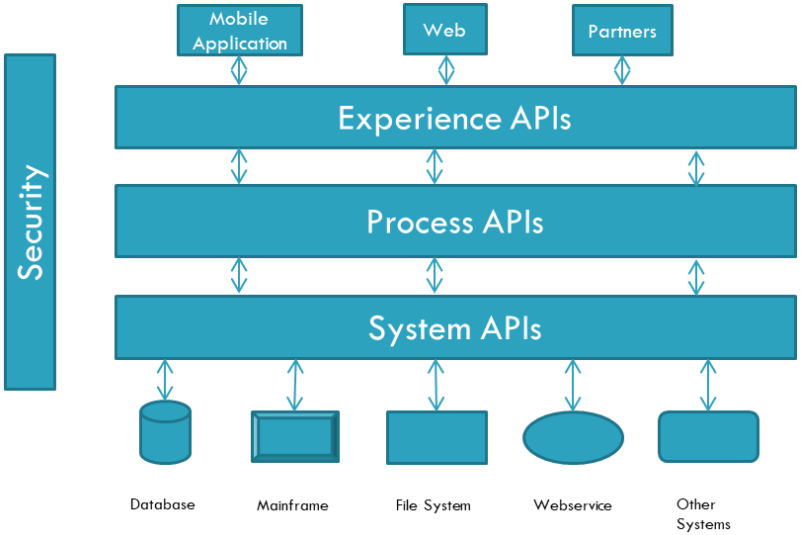
Assumptions

- Existing services can be re-engineered
- Migration of existing applications to the new architecture is not in scope
- Number of users using the application for viewing details are more than the number of users for creating new account

High Level Architecture/Design

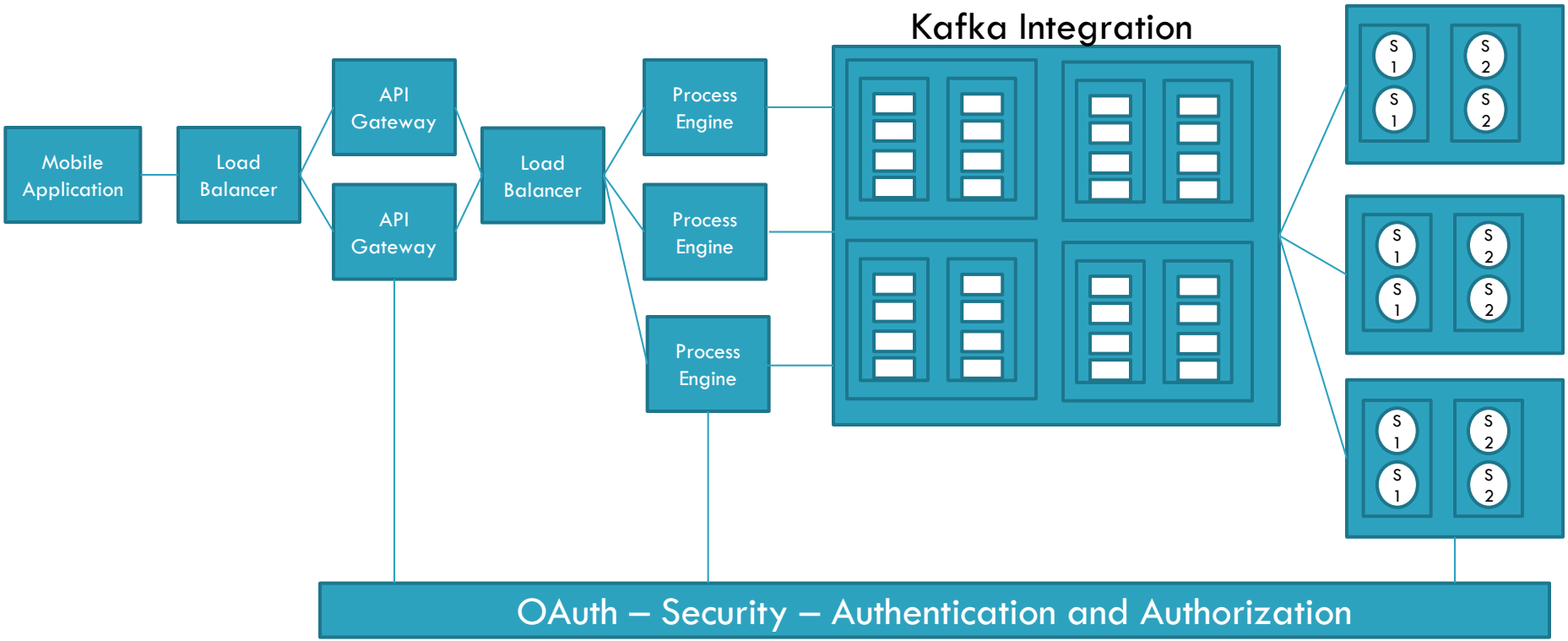


Reference Architecture



Layered Architecture

Low level Architecture – Option 1



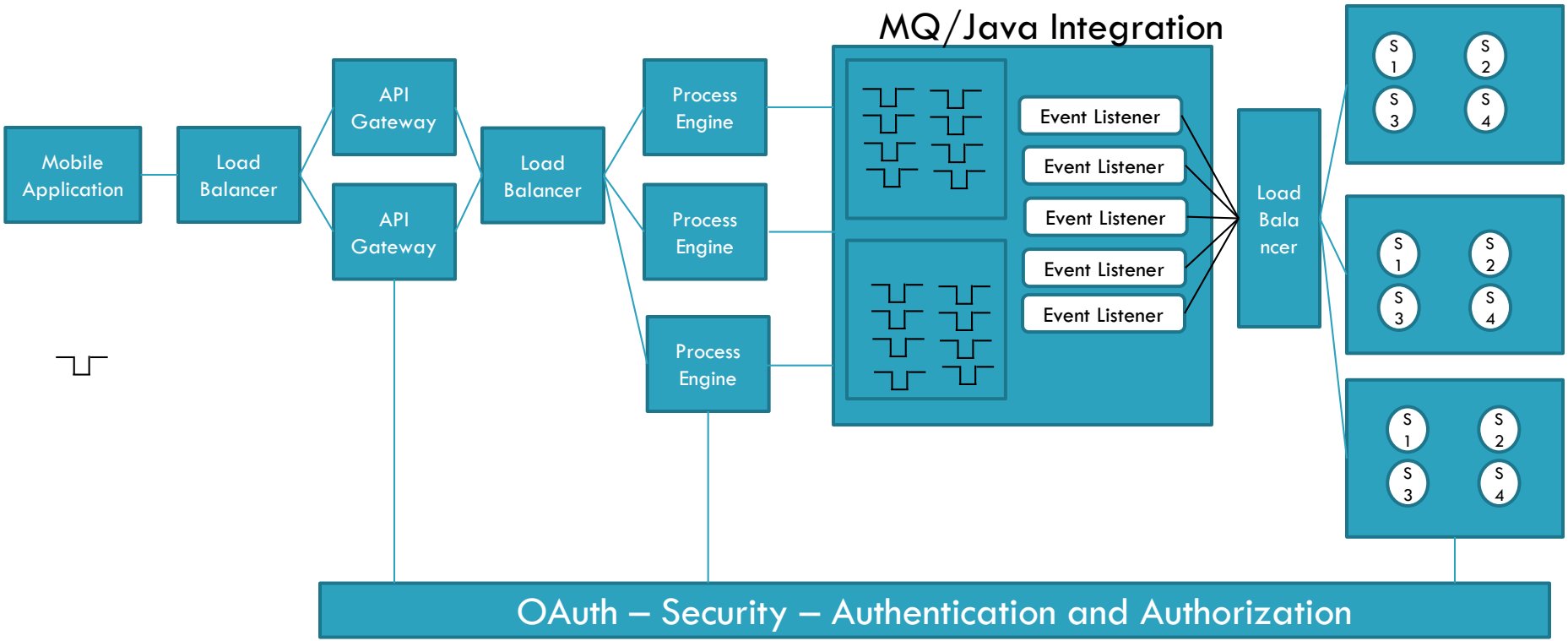
Low level Architecture – Option 1 - Overview

Details:

1. API Gateway – primarily responsible for exposing APIs to external world – partners. This layer will also be used for throttling and monetization. Products that can be considered – APIGEE or WSO2
2. Process Engine – All the business processing and orchestration happens on this layer. This layer will primarily be hosting Process API wherein most of the business data will be processing. Any additional security (Message Encryption) or Data Handling (Message Enrichment) will be handled on this layer. Products that can be considered – Java Spring Integration Framework or COTS
3. Integration – Kafka will be used as the integration/messaging layer. The high processing speed and easy scaling will help us to decouple the backend atomic services. Each Kafka server will have:
    - Brokers – one each for the processing country – this will help in segregating the traffic and also separation of concerns
    - Topics – Each broker will have multiple topics – one topic per atomic service
    - Partitions – helps in storing messages – can be scaled based on the traffic expectation
4. Atomic Services – Java services, developed on Spring boot framework. Each atomic services will have its own server instance and multiple instance can be running on the same server. These atomic services can be group under Kafka Consumer Group and hence there is no need to load balance. Kafka ensures of redistributing the load even if one of the instances is down.
5. As Kafka consumer group helps in balancing the load, there is no need to have a load balancer between Kafka layer and atomic services.
6. OAuth – Ensuring security across the flow of data through each of the layers. Token based authentication at each layer to authenticate and authorize the traffic.

Compliant to Microservices Architecture

Low level Architecture – Option 2



Low level Architecture – Option 2 - Overview

Details:

1. API Gateway – primarily responsible for exposing APIs to external world – partners. This layer will also be used for throttling and monetization. Products that can be considered – APIGEE or WSO2
2. Process Engine – All the business processing and orchestration happens on this layer. This layer will primarily be hosting Process API wherein most of the business data will be processing. Any additional security (Message Encryption) or Data Handling (Message Enrichment) will be handled on this layer. Products that can be considered – Java Spring Integration Framework or COTS
3. Integration – IBM MQ can be used as the messaging layer to ensure that there is loose coupling between the process engine and atomic services. Different event listeners will be listening to the queue to ensure that any message that is put on the queue is immediately serviced. This layer will have:
    - Dedicated Queue Managers for each country
    - Dedicated Queues for each service
    - One event listener per queue to read the message from the queue and invoke the backend atomic service
4. Load balancer between listeners and the atomic services to distribute the traffic as well as to manage the traffic incase of any downtime on the backend atomic services server.
5. Atomic Services – Java services, developed on Spring boot framework. Each atomic services will have its own server instance and multiple instance can be running on the same server. Atomic services to de deployed across multiple instances and scaled as per the requirement.
6. OAuth – Ensuring security across the flow of data through each of the layers. Token based authentication at each layer to authenticate and authorize the traffic.

DANSKE IT – CASE STUDY

Business Deliverables

List of APIs

Business function	Business API	Process API	System API
CustomerManagement	CustomerAPI	CreateCustomerAPI	ValidateDetailsService VerifyAddressService IDVerificationService CustomerCreationService
		ManageCustomerAPI	ViewCustomerService UpdateCustomerService DeleteCustomerService
Account Management	AccountAPI	CreateAccountAPI	ViewCustomerService(Reuse) AccountCreationService NotificationService (Reuse)
		ManageAccountAPI	ViewAccountService VerifyAccountService UpdateAccountService DeleteAccountService
		TransactionManagementAPI	ViewTransactionsService
Payments	PaymentsAPI	AddAccountAPI	ValidateToAccountService AddToAccountService
		TransferMoneyAPI	VerifyAccountService (Reuse) ViewAccountService (Reuse) MoneyTransferService NotificationService (Reuse)
Miscellaneous	NotificationAPI	NotificationAPI	EmailNotificationService SMSNotificationService

API Details - Rationale

CreateCustomerAPI

- API for creating a new customer
- This API will encompass all the steps that is required for a customer to be onboarded

AccountAPI

- API for onboarding a new customer – one customer can have multiple accounts
- Some of the services that will be created can be reused by other APIs as required
- From a business perspective, everything to do with the account creation and view of the account will be part of this service
- Once the account is created, management of the account is also part of this API
- All aspects associated with the account will be part of this API

PaymentsAPI

- This API primarily serves the purpose of transferring money from one account to another account
- This API will make use of some of the atomic services that are already available as part of AccountAPI

NotificationAPI

- This API primarily helps in sending different kind of notifications to the user – email and SMS notifications

All the services have been designed based on the Business Domain and Business Functionality. **The rationale behind the design of the services is using Business Driven API Design.**

API implementation – CustomerAPI – Non-fuctional and Security

Availability

- Load balancer before API Gateway ensures that the request is routed to one of the available Gateways. Further, the load balancer before the process engine as well continuously checks on the health of process engine and routes the request to one of the process Engines.

Security and Data Integrity

- API Gateway – processes application registration – OAuth to be used for registering the application and the services available for the application
- API Gateway – Validates the application ID and if found valid, will generate a temporary token for the service invocation. Process Engine and Service APIs validate the token ID before executing the service

Maintainability and Serviceability

- Integration with any tool like Splunk to ensure that the transaction flow is captured end to end for maintenance purposes.
- Deployment of APIs across each of the layers of API Gateway, Process Engine and System API and across multiple servers enables to make changes on the APIs in a way not to disrupt the traffic

Scalability

- Prepare for the worst – ensure that we understand the maximum traffic and have the appropriate number of instances available. The System APIs can be scaled by spawning a new instance of API based on the alerts on increased spike in traffic. The integration layer too can follow the same.

Regulatory

- Process Engine and System API – any regulatory requirement like auditing and encryption can be taken care at both Process Engine and System API layers

Logical Design and Implementation

Option 1 - Kafka	Option 2 – MQ/EventListener
Open Source – Stability can be a question	Established/Stable product
Comparatively maintenance is difficult	Easy Maintenance
High Performance	Compared to Kafka – performance is slow
Automated Load balancing	Load balancing using cluster mechanism
Message retained even after read	Messages cannot be retained once read
Easy replication - Automatic failover and High Availability	Do not have message replication

Based on the above factors, considering performance, failover and high availability recommend Option 1 for the new Integration Architecture

Strengths and Drawbacks

Strengths

- API based architecture
  - clear distinction of concerns providing high resilience in case of any failures
  - Loosely couple architecture to ensure that process can be taken to completion offline even in case of a failure on the backend
  - Load Balanced – ensure that there is proper load balancing of messages to split traffic across available servers
  - Scalable – system APIs can be scaled easily by replicating instances of the services as required
  - Broker based separation to process messages based on the country of origin
  - Security – highly secured – can create some latency- but is a trade off between security and latency
  - Easy maintenance of services

Drawbacks

- Multiple hops introducing multiple points of failure – however loosely coupled architecture addresses this concern to a certain extent
- Skill – Kafka – sparsely available talent pool – need to build the pool through training sessions