Implementation plan for SMS Gateway Solution for Nagad Ltd.

Contents

1. Introduction

This SDP outlines the design and implementation approach for the Nagad SMS Gateway solution, supporting both VM-based and Kubernetes containerized deployments. The solution aims to provide a reliable, scalable, and secure messaging service tailored for diverse business purposes.

2. System Overview

The SMS Gateway will enable reliable messaging services, including promotional, transactional, and alert notifications. It will support push and pull SMS services, event-based and event action-based notifications, and SMS campaign management. The solution will integrate with the Nagad DFS system and provide comprehensive reporting, logging, and monitoring capabilities.

3. Functional Requirement

Functional Requirements for the NAGAD SMS Gateway Solution

3.1 SMS Delivery Capabilities

1. **API Support for Various Protocols**
   The SMS Gateway should support SMS delivery via HTTPS, HTTP, JSON, and SMPP APIs.
2. **Blacklist/Whitelist/DND Management**
   The solution must include options to manage blacklisted, whitelisted, and DND (Do Not Disturb) numbers.
3. **Unicode SMS Option**
   The system should allow Unicode SMS delivery.
4. **Session/Channel Segregation**
   The system must support session/channel segregation based on SMS type (e.g., transaction SMS vs. promotional SMS).
5. **Push & OTP Message Delivery**
   The gateway should seamlessly transmit real-time notifications and OTP (One-Time Password) messages via an API interface for secure user interactions.
6. **Bulk SMS Distribution**
   - **Personalized Bulk Messaging**: Ability to send customized SMS to multiple recipients simultaneously using file uploads (XLS, CSV) or JSON arrays (many-to-many).
   - **Predefined Bulk Messaging**: Support for distributing predefined messages to recipients using file uploads or JSON requests (one-to-many).
7. **Scheduled SMS**
   Users should be able to schedule SMS delivery for future dates.
8. **Retry Mechanism for Failed SMS**
   The system must include a configurable retry mechanism for undelivered SMS.

9. **Configurable Masking/Short Code by SMS Type**
   The system should enable configuration of masking/short code based on SMS/user type.

10. **MNP Checking Capabilities**
    The system should support Mobile Number Portability (MNP) checking capabilities, which may be required at a later stage.

3.2 Push-Pull SMS Service

1. **Automated Information Retrieval**
   The system should automatically collect and process data from incoming messages, delivering template-based responses to users.

2. **Dynamic Response System**
   Depending on the content of user requests, the system must deliver predefined template messages in response.

3.3 SMS Channel Management

1. **Multiple Delivery Channels per MNO**
   The SMS Gateway should connect to multiple delivery channels for each MNO (e.g., HTTP/HTTPS API and SMPP for Banglalink).

2. **Configurable GUI for Channel Management**
   A GUI interface must be available to manage SMS delivery channels based on transaction types and MNO configurations. This includes segregating TPS for different types of priority SMS.

3. **Automatic Rerouting of SMS**
   The system should automatically reroute SMS messages to alternative channels in case a primary SMSC or delivery channel fails (e.g., fallback from HTTP API to SMPP for a particular MNO).

4. **Heartbeat Mechanism**
   A heartbeat mechanism is needed to continuously monitor channel availability and automatically switch to a secondary channel if the primary one fails.

3.4 Store-and-Dispatch Capability for Priority SMS

1. **Priority SMS Storage**
   The system must store priority-type SMS (e.g., SMS with priority level 50) and dispatch them based on user triggers at a later time.

3.5 Queue Mechanism for Priority-Based SMS

1. **SMS Priority Queue**
   The system must maintain a priority queue for SMS delivery, based on the priority level defined by the DFS system.

2. **Validation of MNO and Channel Combination**
   Upon receiving SMS data, the system must validate the MNO and the corresponding delivery channel.

3. **Priority Routing**
   The system should route SMS through appropriate gateways and utilize the defined TPS allocation based on the priority level.

4. **Priority Selection Table**
   The system must allow configuration of SMS priority levels (e.g., OTP as highest, Marketing as low).

3.6 Prefix-based MNO Selection in API Requests

1. **MNO Prefix Configuration**
   The system should allow the configuration of the first three digits (prefix) to determine the destination MNO when the MNO name is not provided.

3.7 SMS Identification Requirements

1. **Unique SMS Identification**
   The system should assign a unique identification code to each SMS to enable precise tracking.

2. **Application-Wise Identification**
   Each SMS should be attributed to its originating application, allowing clear visibility of the message flow.

3.8 Web GUI Interface

1. **Role-Based Access Control (RBAC)**
   The system must provide a web portal with role management (Admin, Super Admin, User) to control configuration activities and access MIS reports.

2. **Maker-Checker Mechanism**
   A maker-checker mechanism should be implemented for configuration management and operations.

3. **Real-Time Message Status**
   The web interface should provide real-time updates on the status of messages.

4. **Failure Notifications for Undelivered SMS**
   Failure notifications should be displayed in the web interface, including error descriptions.

5. **SMS Submission Check Status**
   The web GUI should allow users to check the status of SMS submissions to MNOs and the delivery status to the handset.

3.9 Reporting Module

1. **MIS Reporting**
   The system must generate detailed and summary reports on SMS activity, based on criteria like MNO, user, channel, and time-period.

2. **Customizable Reports**
   Reports should be customizable to include MNO-wise, user-wise, and channel-wise views, as well as periodic summaries (e.g., monthly, weekly).

3.10 Compliance and Security

1. **Data Protection**
   Sensitive data (e.g., OTPs) should be encrypted both in transit and at rest. Logs should implement data masking where necessary.

2. **Access Control**
   Implement role-based access control (RBAC), multi-factor authentication (MFA), and IP whitelisting for portal access.

3. **API Security**
   The APIs should support OAuth 2.0 or API key authentication and use HTTPS/TLS for secure communication.

4. **System Logging and Auditing**
   Detailed logs for SMS processing and system activities should be maintained for auditing purposes.

5. **High Availability (HA)**
   The system architecture must support high availability (HA), with zero downtime for updates or upgrades.

6. **Vulnerability Management**
   The solution must address security vulnerabilities identified during Nagad's security screenings.

7. **Data Retention**
   Define retention periods for SMS logs and user data, ensuring compliance with applicable regulations.

3.11 Technical Support and Logging

1. **24/7 Technical Support**
   Provide continuous technical support, with a dedicated Single Point of Contact (SPOC) for resolving issues.

2. **SMS Logging**
   Maintain detailed logs for all SMS requests, including timestamps for sending and delivery.

3. **System Monitoring and Performance Tracking**
   Implement robust monitoring to track system performance, message delivery, and integration points.

4. **Troubleshooting and Audit Logging**
   Enable detailed logging to facilitate troubleshooting and maintain audit trails.

3.12 Comprehensive Testing Infrastructure

1. **Sandbox Testing Environment**
   Provide a sandbox environment for comprehensive end-to-end testing of integrations.

3.13 Redundancy & Load Balancing

1. **Dynamic Message Routing**
   Enable dynamic routing based on load balancing, network availability, and cost optimization.

2. **Load Balancing and Redundancy**
   Implement load balancing and redundancy at the application and database layers to ensure high availability.

3.14 Backup and Recovery

1. **Backup and Recovery Plan**
   Implement a comprehensive backup and recovery plan to ensure data integrity.

2. **Disaster Recovery Plan**
   Provide a disaster recovery plan to ensure system continuity in case of catastrophic failure.

3.15 Maintenance & Monitoring

1. **Web Server Monitoring**
   Monitor API response status and alert if APIs are not responding with the expected HTTP codes.

2. **Application-Level Monitoring**
   Track issues related to database connections, cache errors, and other application-level disruptions.

3. **VM Monitoring**
   Monitor system resource utilization, including CPU, memory, disk space, and IOPS.

4. **MNO Failure Monitoring**
   Track failures in SMPP/HTTP accounts and message rejections by MNO accounts.

5. **Queue Monitoring**
   Monitor the processing of message queues to ensure smooth operation.

6. **Archiving and Purging**
   Implement archiving and purging mechanisms for efficient database management.

3.16 Control Process & Reporting

1. **Data Masking**
   Mask sensitive data (PIN/OTP/Password) in transit and at rest.

2. **SMS Request Logging**
   Log all SMS sending requests with details such as Source System, Module/API from DFS, and Unique Request Identifier.

3. **Fraud Management**
   Implement a fraud management system that prevents fraudulent activities based on SMS type, user, or account number.

4. **Approval Workflow for Bulk SMS**
   Implement an approval workflow for bulk SMS requests.

5. **Auto-Purge for OTP SMS**
   Implement an auto-purge mechanism for OTP SMS to prevent sending expired messages.

3.17 Data Feeds

1. **Near Real-Time Data Feeds**
   Provide data feeds for external systems in near real-time (every 5 minutes or 10MB file).

4. High Level Architecture

- **Presentation Layer:** API and GUI-based interfaces for MIS management, bulk message distribution, and DFS integration.
- **Application Layer:** SMS Gateway services, message processing, and campaign management.
- **Data Layer:** Database and storage services for message data, templates, and campaign information.
- **Infrastructure Layer:** Virtualized environment (VMware Cloud) or containerized environment (Kubernetes) with auto-scaling capabilities.

5. Data Flow Diagram

The **Data Flow Diagram (DFD)** for the **SMS Gateway Solution** for Nagad Ltd represents the flow of data through the system and how different components interact. It is broken down into **level 1** and **level 2** DFDs to provide clarity.

Level 0: Context Diagram

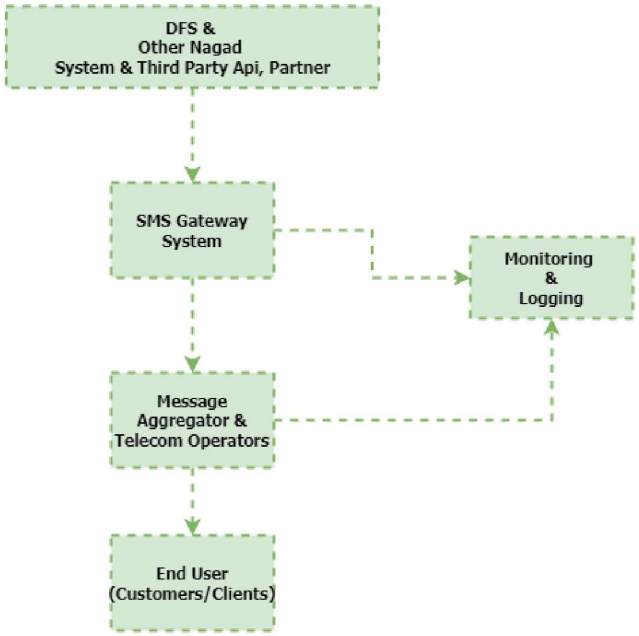

Fig: DFD-Context Diagram

This high-level diagram shows the entire system as a single process and outlines the major external entities interacting with the system.

- **External Systems**: These are third-party systems that interact with the SMS Gateway, like partner organizations (e.g., banks, telcos) or external APIs for sending/receiving SMS messages.
- **SMS Gateway System**: This is the core of the solution, which processes and routes SMS messages based on business logic.
- **Monitoring & Logging**: This system collects logs, monitors system health, and tracks the status of SMS deliveries.
- **Message Aggregator & Telecom Operator**: Interfaces with the external SMS network to deliver messages to recipients.
- **End Users**: Represents customers who either receive SMS notifications (push) or send messages to the business (pull).

Level 1: System Components Interaction

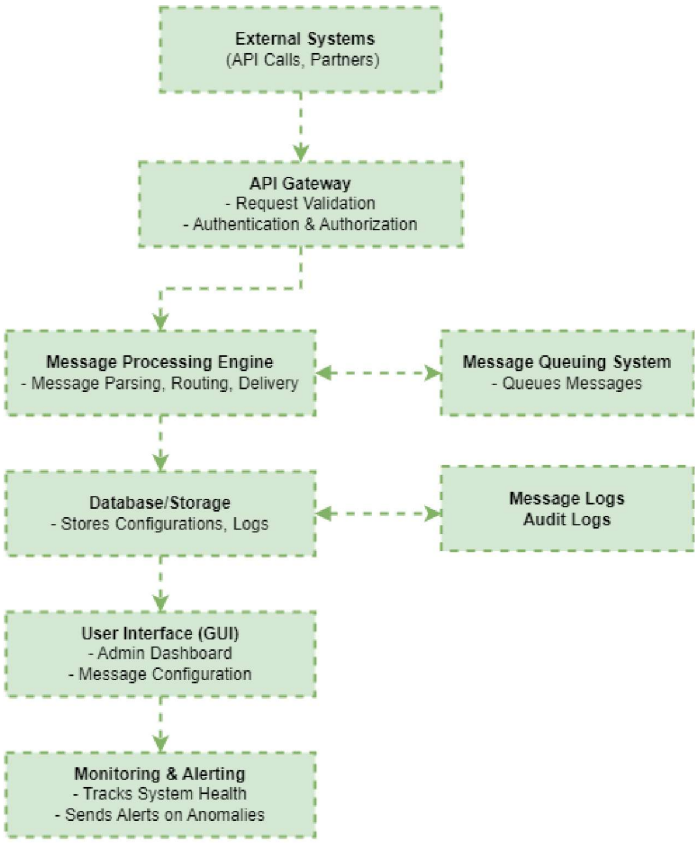

Fig: DFD - System Components Interaction

This diagram delves deeper into the core components of the system and how they interact with each other.

- **API Gateway**: Handles incoming API requests from external systems. It authenticates, validates, and routes requests to the appropriate service (Message Processing Engine).
- **Message Processing Engine**: Core logic that parses the message, applies business rules for routing and prioritization, and sends it to the Message Queuing System.
- **Message Queuing System**: Ensures reliable message delivery by queuing messages for processing by the appropriate delivery channel.
- **Database/Storage**: Stores all system configurations, user data, audit logs, and message metadata.
- **User Interface (GUI)**: Provides administrators with the ability to configure, monitor, and manage message operations. This is where the user can define message templates, set priorities, and review delivery statistics.
- **Monitoring & Alerting**: Tracks system performance, health, and logs. Sends alerts when certain thresholds or anomalies are detected.

Level 2: Detailed Process Flow of the Message

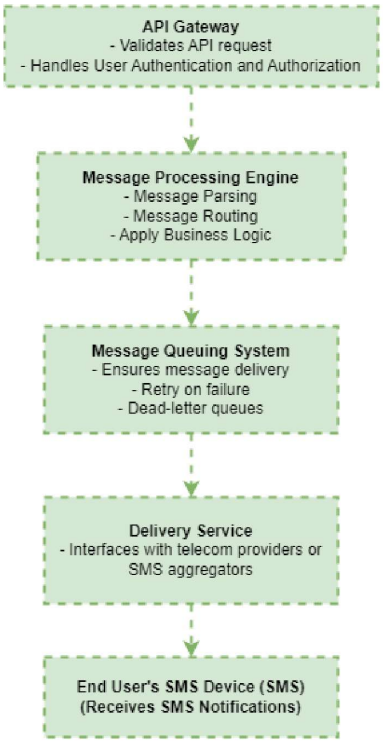

Fig: DFD - Detailed Process Flow of the Message

- **API Gateway** receives incoming requests (either push or pull SMS) and authenticates them.
- **The Message Processing Engine** handles the business logic, parsing, and routing of messages based on user configuration (such as priorities).
- Messages are then sent to the **Message Queuing System** for reliable delivery. If a message cannot be sent immediately, it will be retried or placed in a dead-letter queue for later investigation.
- The **Delivery Service** interfaces with external systems (e.g., telecom operators) to send the SMS messages to the end users.
- End Users' SMS Devices (mobile phones) receive the notifications and responses as part of the business process.

6. Microservices Descriptions

The API Gateway Service is the single-entry point for incoming requests, handling routing, authentication, and access control for all interactions with the system. The Message Validation Service ensures that all incoming messages conform to required formats and business rules before processing, preventing errors and ensuring compliance. The Message Routing Service determines the best delivery route for each message, based on factors such as sender, recipient, and message type, optimizing the use of available channels. The Message Delivery Service processes and delivers messages, supporting different protocols (HTTP, SMPP) and providing features like retries, delivery confirmation, and performance optimization. Additional microservices like the Template Management Service, Campaign Management Service, and Priority Queue Management Service offer specialized functionality such as SMS template storage, campaign scheduling, and handling high-priority message queues, ensuring efficient and targeted message delivery.

- **API Gateway Service**

  Single entry point for all incoming messages and requests. Handles authentication, authorization, and routing.

- **Message Validation Service**

  Validates incoming messages for format, content, and compliance.

- **Message Routing Service**

  Determines the appropriate route for message delivery based on message type, sender, and other criteria.

- **Message Delivery Service**

  Handles the actual delivery of messages to the intended recipients.

  Supports various messaging types and protocols.

  1. **Message Queuing:**

- Receives messages from the Message Routing Service and places them in a queue for delivery.

2. **Protocol Handling:**
    - Supports various messaging types and protocols (e.g., SMPP, HTTP, SMTP) for message delivery.
3. **Delivery Attempts:**
    - Manages multiple delivery attempts in case of failures, with configurable retry logic.
4. **Delivery Confirmation:**
    - Handles delivery receipts and confirmation messages from the recipient's network.
5. **Error Handling:**
    - Manages errors and exceptions during message delivery, logging them for further analysis.
6. **Performance Optimization:**
    - Optimizes message delivery based on priority queues and message types.
7. **Scalability:**
    - Scales horizontally to handle large volumes of messages efficiently.
8. **Integration:**
    - Integrates with external systems and third-party services for message delivery.

- **Template Management Service**
    - Manages predefined templates for common message types.
    - Supports creating, updating, and deleting templates through a web interface.
- **Campaign Management Service**
    - Manages the creation, scheduling, and automation of SMS campaigns.
    - Supports detailed reporting on campaign performance.
- **Event Notification Service**
    - Handles event-based notifications triggered by specific events in the system.
- **Event Action Notification Service**

- Handles event action-based notifications triggered by user actions or system processes.

- **Priority Queue Management Service**
  - Manages dynamic priority queues for optimized message delivery.
  - Supports configurable priority levels for different types of messages.
- **Push-Pull Service**
  - Supports both push (business to customer) and pull (customer to business) SMS services.
  - Handles large volumes of push and pull messages efficiently.
- **Reporting and Logging Service**
  - Provides comprehensive reporting, logging, and monitoring capabilities.
  - Collects and maintains logs for audit purposes.
- **Security and Compliance Service**
  - Ensures secure and compliant handling of sensitive data.
  - Manages encryption, authentication, and authorization.
- **Integration Service**
  - Facilitates seamless integration with existing business systems and third-party services.
  - Handles data exchange and API interactions.
- **Monitoring and Alerting Service**
  - Monitors system performance, health, and security.
  - Provides alerts and notifications for critical events.
- **Backup and Restore Service**
  - Manages backup and restore of all data, configuration, and logs.
  - Ensures compliance with RTO, RPO, and retention policies.

7. Detailed System Architecture – (VM & K8S)

7.1 VM-based Architecture

1. **Nagad Consumer Manager Application:**
   - Interfaces with the Nagad Push-Pull Service, Nagad SMS Push Portal, and Nagad DFS System.

- Communicates via API Clients and Web Browsers.

2. **System Monitoring (Prometheus, Grafana):**
    - Tools used for monitoring the system's performance and health.
3. **Load Balancer 1:**
    - Distributes incoming traffic to multiple App Server VMs (Virtual Machines).
    - Ensures high availability and reliability by balancing the load.
4. **App Server VM1 and App Server VM2:**
    - Each VM hosts multiple services essential for the application's functionality:
        - API Gateway Service: Manages API requests.
        - Campaign Management: Handles campaign-related operations.
        - UI & Core Services: Core functionalities and user interface services.
        - Message Validation Service: Validates messages.
        - Event Notification Service: Manages event notifications.
        - Event Action Notification Service: Handles actions based on events.
        - Reporting & Logging: Generates reports and logs activities.
        - Integration Service: Integrates with other systems.
        - Template Management Service: Manages message templates.
        - Monitoring and Alerting Service: Monitors the system and sends alerts.
        - Backup & Restore Service: Manages data backup and restoration.
        - Notification Service: Sends notifications.
        - Database Service: Manages database operations.
        - Message Routing Service: Routes messages to the correct destination.

- Priority Queue Management Service: Manages priority queues.

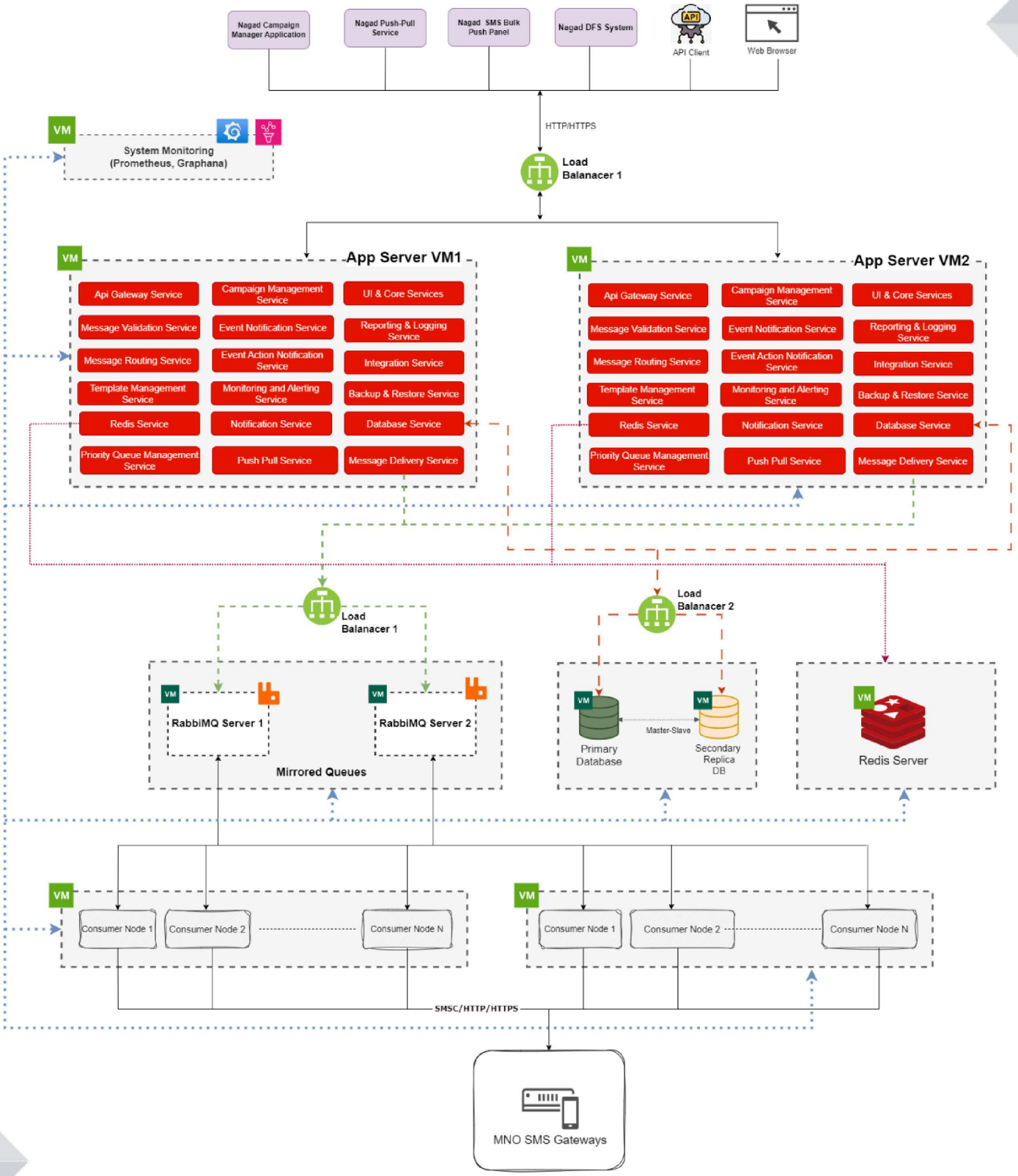

Fig: Solution Architecture Diagram

- Push Pull Service: Handles push and pull operations.
- Message Delivery Service: Ensures message delivery.

5. **Load Balancer 2:**
    - Distributes traffic to RabbitMQ Servers, Databases, and Radis Server.
6. **RabbitMQ Server 1 and RabbitMQ Server 2:**
    - Message brokers that handle message queuing.
    - Use mirrored queues for redundancy and reliability.
7. **Primary Database and Secondary Database:**
    - Master-Slave configuration for data storage.
    - Ensures data consistency and availability.
8. **Radis Server:**
    - In-memory data structure store, used for caching and real-time data processing.
9. **Consumer Nodes:**
    - Multiple nodes (Consumer Node 1 to Consumer Node N) that process messages.
    - Communicate with RabbitMQ Servers and other services.
10. **MNO SMS Gateways:**
    - Interfaces with Mobile Network Operator (MNO) SMS Gateways for sending and receiving SMS messages.
    - Communicates via SMPP/HTTP/HTTPS protocols.

Communication Flow:

- HTTP/HTTPS: Used for communication between API Clients, Web Browsers, Load Balancers, and App Servers.
- SMPP/HTTP/HTTPS: Used for communication between Consumer Nodes and MNO SMS Gateways.

7.2 Container Orchestration Based Architecture

This solution architecture diagram represents a Kubernetes-based system designed to manage SMS Gateway (SMSGW) operations efficiently. Below is an explanation of each component and its role:

1. **External Systems**
    - **Sources:** External entities like **Nagad DFS**, Partner APIs, Telecom Providers, and others interact with the system using APIs or webhooks.

- These systems initiate requests for SMS services, which are then routed through the architecture.

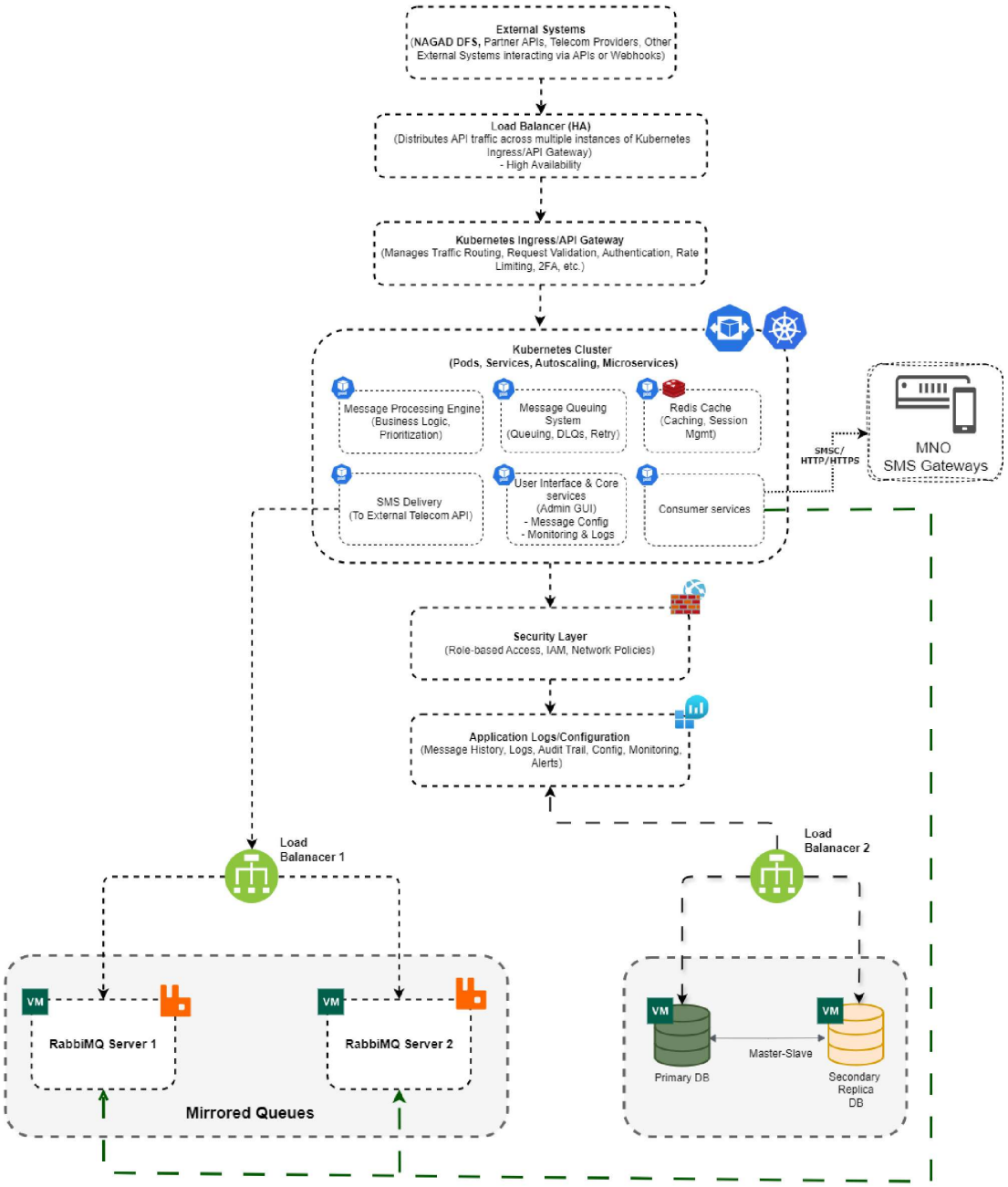

Fig: Solution Architecture using K8s

2. Load Balancer (HA)

- Distributes incoming traffic to multiple Kubernetes ingress instances.

- Ensures **High Availability** and prevents bottlenecks by spreading the load evenly across services.
- Works with Kubernetes Ingress or an API Gateway for scalability and fault tolerance.

3. Kubernetes Ingress/API Gateway

- Handles:
    - **Traffic Routing:** Directs requests to appropriate services within the Kubernetes cluster.
    - **Request Validation:** Ensures incoming requests are valid.
    - **Authentication:** Verifies the source of the request.
    - **Rate Limiting:** Controls the number of requests to prevent overloading.
    - **2FA:** Adds an additional layer of security for sensitive operations.

4. Kubernetes Cluster

- The heart of the system, consisting of various **Pods**, **Services**, **Autoscaling**, and **Microservices** for different functionalities:
- **(a) Message Processing Engine:**
    - Implements business logic, prioritizes SMS processing, and manages delivery tasks.
- **(b) Message Queuing System:**
    - Uses RabbitMQ to manage message queues.
    - Supports retry mechanisms, Dead Letter Queues (DLQs), and ensures message durability.
- **(c) Redis Cache:**
    - Acts as a caching layer for fast session management and temporary data storage.
- **(d) SMS Delivery Services:**
    - Interfaces with external telecom APIs to send SMS messages.
- **(e) User Interface & Core Services:**
    - Admin GUI for managing configurations.
    - Includes monitoring, logs, and message configuration tools.
- **(f) Consumer Services:**
    - Dynamically scales consumer instances based on traffic requirements to handle high throughput efficiently.

5. Security Layer

- Ensures robust security with:
    - **Role-Based Access Control (RBAC):** Limits user access based on roles.
    - **Identity and Access Management (IAM):** Manages user permissions and authentication.
    - **Network Policies:** Enforces secure communication between services within the cluster.

6. Application Logs/Configuration

- Maintains logs for:
    - **Message History:** Tracks SMS sent/received.
    - **Audit Trails:** Logs system changes for accountability.
    - **Monitoring and Alerts:** Monitors the system's health and sends alerts on anomalies.
- Stores configuration details for system operations.

7. RabbitMQ (Message Queuing Backend)

- Two RabbitMQ servers are set up with **Mirrored Queues** for high availability and failover support.
- The **Load Balancer 1** ensures traffic distribution across the RabbitMQ servers.

8. Database System

- **Primary DB (Write):** Handles write operations and stores critical data.
- **Secondary Replica DB (Read):** Provides a replica for read operations to reduce load on the primary database.
- Database replication ensures data redundancy and fault tolerance.

9. Integration with Telecom Gateways

- The SMS delivery services interact with **MNO SMS Gateways** via **SMPP**, **HTTP**, or **HTTPS** protocols to send SMS to end-users.

8. Network Architecture Diagram – (VM & K8S)

1. Network Architecture Using Virtual Machines (VMs)

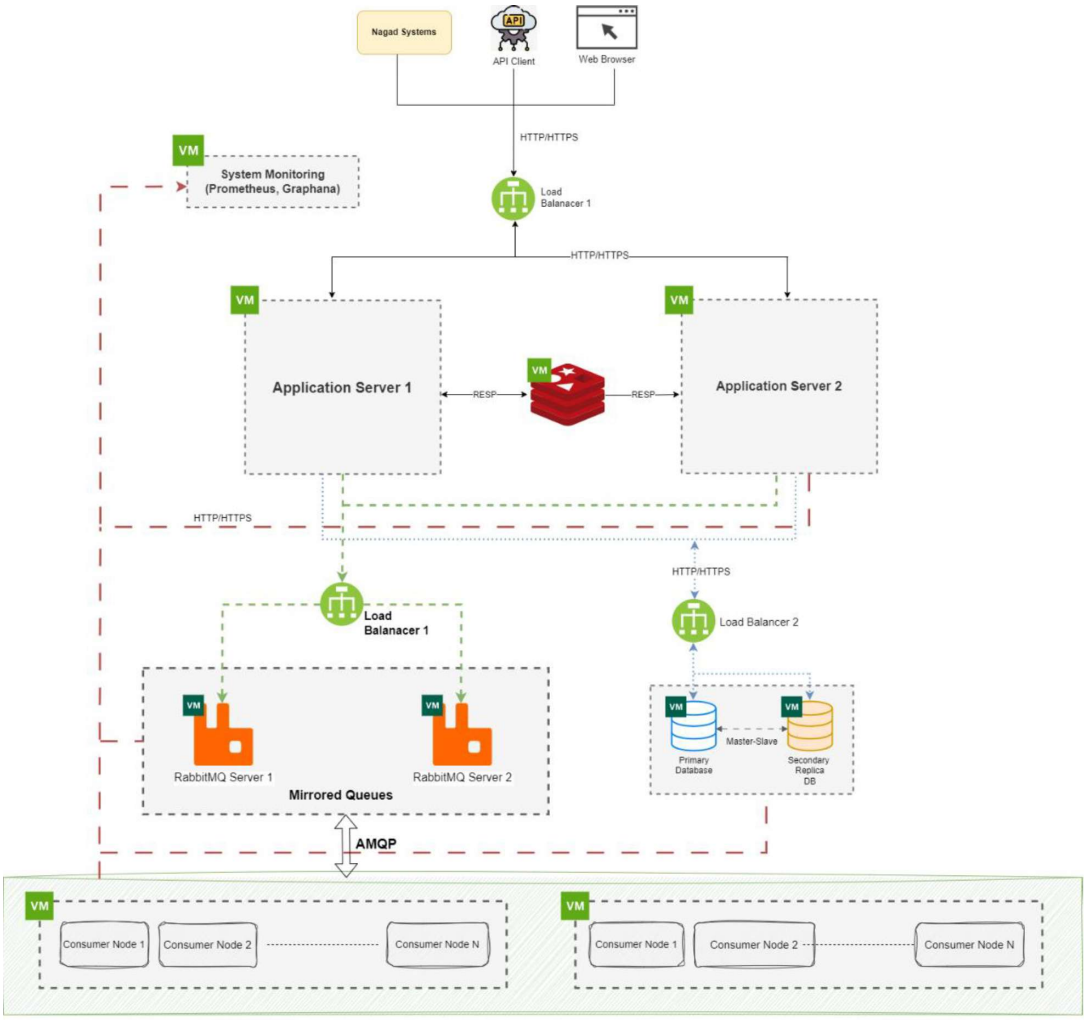

Fig: Network Architecture Diagram using VM

This architecture relies on traditional VM instances for deploying and scaling the system.

- **External Systems:**
    - Interact with the architecture through HTTP/HTTPS protocols via APIs or web browsers.
    - Systems like Nagad Systems use the APIs for SMS-related operations.
- **Load Balancer 1:**
    - Distributes traffic across multiple application servers.
    - Ensures high availability and handles load management.

- **Application Servers:**
    - Two application servers handle the core logic for SMS operations.
    - Connected to a Redis instance for session management and caching to ensure quick data access.
- **Load Balancer 2:**
    - Balances traffic between the primary database and the secondary replica database (used for read operations).
    - A master-slave configuration ensures data redundancy and fault tolerance.
- **RabbitMQ Servers:**
    - Two RabbitMQ servers are configured with mirrored queues for high availability.
    - AMQP (Advanced Message Queuing Protocol) is used for communication between application servers and RabbitMQ.
- **Consumer Nodes:**
    - A set of consumer nodes handles message consumption and delivery tasks.
    - Consumers are scaled manually or based on queue length to handle high throughput.
- **System Monitoring:**
    - Tools like Prometheus and Grafana are used for monitoring performance and providing insights.

Advantages of VM Architecture:

- Simpler to set up for teams already familiar with VM environments.
- Reliable for smaller or static-scale deployments.

Disadvantages:

- Less dynamic scaling compared to Kubernetes.
- Higher resource overhead due to fixed VM allocation.
- Manual intervention required for scaling and fault recovery.

2. Network Architecture Using Kubernetes (K8s)

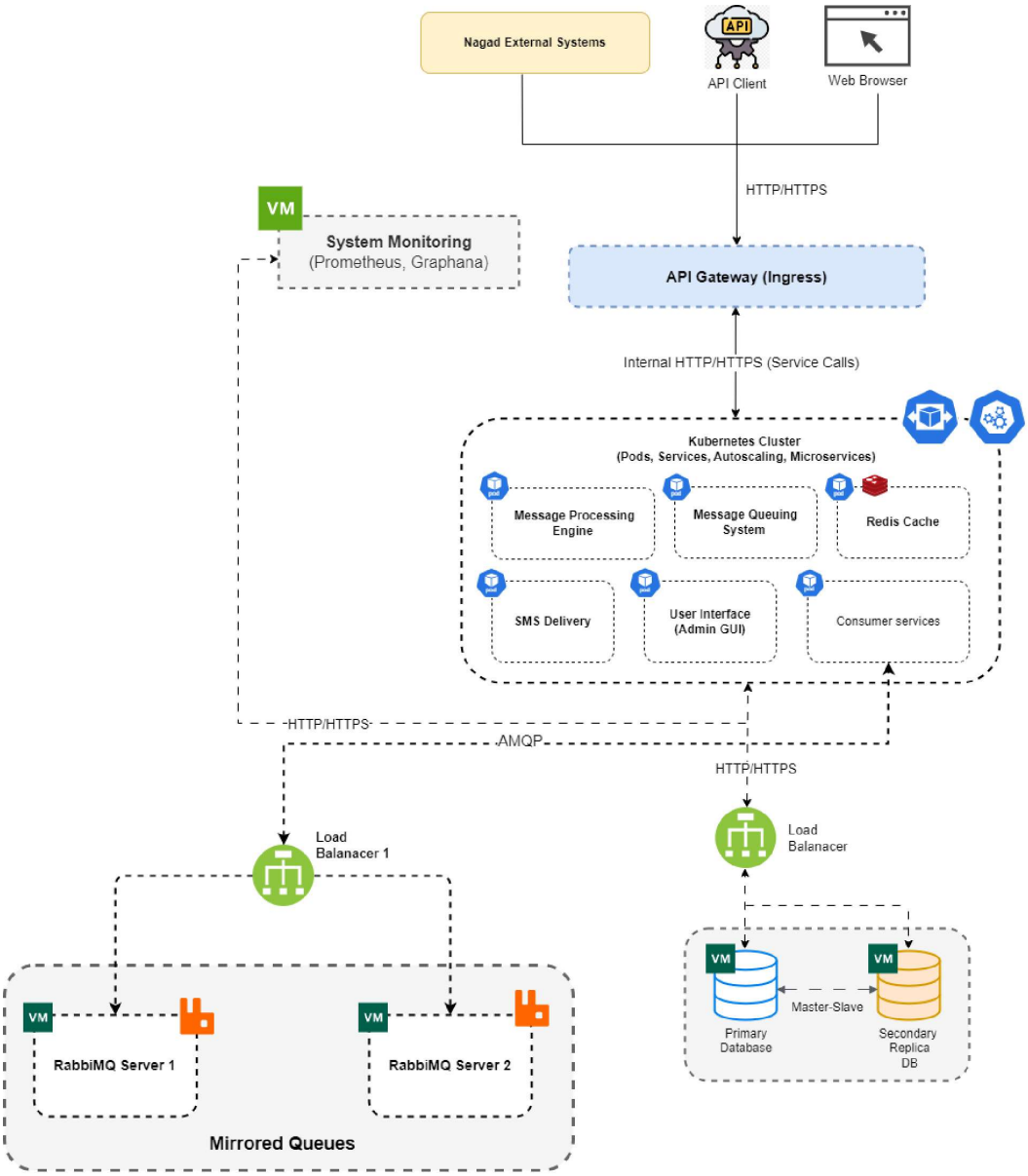

Fig: Network Architecture Diagram usign K8S

This architecture leverages Kubernetes for a more dynamic, scalable, and containerized system deployment.

- **External Systems:**
    - Similar to the VM architecture, external systems interact via APIs or web browsers.
- **API Gateway (Ingress):**
    - Replaces the traditional load balancer with a Kubernetes Ingress Controller.

- Manages traffic routing, authentication, and rate-limiting for services within the cluster.

- **Kubernetes Cluster:**
  - Hosts a set of containerized microservices for:
    - Message Processing Engine (handling business logic and prioritization).
    - Message Queuing System (RabbitMQ for message queuing).
    - Redis Cache (for caching and session management).
    - SMS Delivery (interacting with external telecom APIs).
    - Admin GUI and Core Services (for configuration, monitoring, and user interface).
  - Kubernetes automatically scales pods based on workload.
- **RabbitMQ Servers:**
  - Similar to the VM setup, RabbitMQ servers have mirrored queues, but they are containerized and managed within the Kubernetes cluster.
- **Load Balancer 2:**
  - Distributes traffic to the primary database and its replica.
  - Kubernetes can scale database connections dynamically if configured.
- **System Monitoring:**
  - Uses the same tools (Prometheus and Grafana) for monitoring, deployed as Kubernetes services.

Advantages of Kubernetes Architecture:

- Dynamic Scaling: Pods scale automatically based on traffic and resource usage.
- Fault Tolerance: Kubernetes self-heals by restarting failed containers.
- Resource Efficiency: Better utilization of resources with containerization.
- CI/CD Integration: Easier integration with modern development pipelines for automated deployments.

Disadvantages:

- Requires more expertise to manage.
- Initial setup complexity can be higher compared to VMs.

9. E-R Diagram

Please refer to E-R Diagram.pdf for the E-R Diagram.

10. High Availability – (VM & K8S)

1. High Availability in VM-Based Architecture

In a virtual machine setup, HA is achieved by leveraging redundancy and fault-tolerant components.

Key HA Components in VM Architecture:

1. **Load Balancers:**
    - Load Balancer 1 distributes API requests between multiple application servers.
    - If one server fails, the load balancer reroutes traffic to other healthy servers.
2. **Redundant Application Servers:**
    - Multiple application servers (e.g., Application Server 1 and 2) ensure that if one server goes down, another takes over.
    - These servers are configured to handle the same workload.
3. **Mirrored RabbitMQ Queues:**
    - RabbitMQ servers (e.g., RabbitMQ Server 1 and 2) use mirrored queues, where message queues are replicated across servers.
    - If one RabbitMQ server fails, the other seamlessly takes over without data loss.
4. **Primary and Replica Databases:**
    - The database layer has a master-slave (primary-replica) setup.
    - If the primary database becomes unavailable, the replica can take over for read operations until recovery.
5. **Manual Recovery:**
    - In the VM architecture, scaling and failover (e.g., adding a new application server) often require manual intervention or automation scripts like Ansible, Terraform, or cloud-provider features (e.g., AWS Auto Scaling Groups).

Limitations of HA in VM-Based Systems:

- Scaling is slower and less automated compared to Kubernetes.
- Manual effort may be needed to restart failed services or VMs.
- The system may experience some downtime before the failover occurs.

2. High Availability in Kubernetes-Based Architecture

Kubernetes natively provides features for HA, making it more robust and automated compared to VMs.

Key HA Components in Kubernetes Architecture:

1. **Kubernetes Ingress (API Gateway):**
    - The Ingress controller acts as a load balancer, distributing traffic across pods (microservices).
    - It automatically detects unhealthy pods and reroutes traffic to healthy ones.
2. **Replica Sets for Microservices:**
    - Kubernetes ensures that a specified number of replicas (pods) for each microservice (e.g., SMS Delivery, Message Processing Engine) are always running.
    - If a pod fails, Kubernetes automatically restarts it on a healthy node.
3. **Cluster-Level Redundancy:**
    - Kubernetes runs on a cluster of nodes. If one node fails, the workloads are automatically moved to other healthy nodes in the cluster.
    - This ensures zero downtime for services.
4. **RabbitMQ in Kubernetes:**
    - RabbitMQ servers are containerized and managed as pods. The mirrored queues feature of RabbitMQ remains intact, with Kubernetes ensuring the RabbitMQ pod's availability across the cluster.
5. **Database HA:**
    - The database is still set up with a primary-replica (master-slave) configuration.
    - Kubernetes monitors and can restart database pods or scale replicas automatically.
6. **Health Checks & Self-Healing:**
    - Kubernetes performs liveness and readiness probes to detect unhealthy pods.
    - It can replace or restart failed pods automatically, ensuring continuous service.
7. **Autoscaling:**
    - Kubernetes uses Horizontal Pod Autoscaling (HPA) to scale up pods when traffic spikes or resource utilization increases (CPU, memory, etc.).
    - This ensures that services remain available under heavy load.

Advantages of HA in Kubernetes:

- Automated Recovery: No manual intervention is needed for failover or recovery.
- Zero Downtime: Workloads are redistributed dynamically across healthy nodes.
- Proactive Scaling: Auto-scaling ensures resources are added dynamically based on demand.
- Rolling Updates: Updates to microservices are deployed without downtime, maintaining HA during deployments.

11. Scalability – (VM & K8S)

1. Scalability in VM-Based Architecture

Horizontal Scalability

- **Adding More Application Servers:**
  - New application servers (VMs) can be added behind the load balancer to handle increased API traffic or computational demands.
  - The load balancer automatically distributes traffic among the available servers.
- **Scaling RabbitMQ Servers:**
  - RabbitMQ supports clustering, which allows you to add more nodes to the RabbitMQ cluster for increased throughput and resilience.
- **Database Scaling:**
  - Read replicas can be added to distribute read traffic.
  - A load balancer can route read requests to replicas while write requests go to the primary database.
- **Consumer Nodes:**
  - More consumer nodes can be added to handle increased message-processing loads from RabbitMQ queues.

Vertical Scalability

- **Increasing VM Resources:**
  - CPU, memory, or storage of individual VMs (application servers, RabbitMQ servers, database servers) can be increased as needed.
  - Virtualization platforms (like VMware, Hyper-V, or AWS EC2) make it easy to resize VMs.
- **Improving Database Performance:**

- The primary database server can be upgraded with more CPU, memory, or SSD storage to handle larger write workloads.

Challenges with VM-Based Scaling:

- Horizontal scaling requires more manual effort (e.g., provisioning and configuring new VMs).
- Vertical scaling is limited by hardware constraints (e.g., a single VM cannot exceed the physical host's resources).

---

2. Scalability in Kubernetes-Based Architecture

Horizontal Scalability

- **Pods and Services:**
  - Kubernetes allows horizontal scaling of pods for each microservice (e.g., Message Processing Engine, SMS Delivery, etc.).
  - Horizontal Pod Autoscaler (HPA) automatically adjusts the number of pods based on resource usage or custom metrics.
- **Cluster Nodes:**
  - New nodes can be added to the Kubernetes cluster, and the scheduler automatically distributes pods across the nodes.
- **Database and RabbitMQ:**
  - **Similar to VM-based architecture, RabbitMQ servers and databases can be** scaled horizontally with additional replicas or nodes.
  - Kubernetes can manage these as stateful sets, ensuring proper replication and distribution.

Vertical Scalability

- **Pod Resource Allocation:**
  - Resources (CPU, memory) allocated to each pod can be dynamically adjusted using Kubernetes' resource requests and limits.
  - For example, if the Message Processing Engine needs more CPU during peak hours, its pod can be allocated additional resources.

12. Infrastructure Requirement – (VM & K8S)

Below is a approximate infrastructure requirement for the years 2025–2028.

➢ **Using VM:**

	Quantity	CPU	Memory	Storage	Network
2025					
App Server	2	32 vCPUs	64 GB	1 TB	1Gbps
Consumer Node	>10	32 vCPUs	64 GB	150 GB	1Gbps
Database Server	2	32 vCPUs	64 GB	2TB	500Mbps
RabbitMQ Server	2	32 vCPUs	64 GB	1 TB (SSD)	1Gbps
Redis Server	1	16 vCPUs	32GB	500GB (SSD)	500Mbps
Load Balancer	3	16 vCPUs	32 GB	200GB	1Gbps
Hybrid SAN	1	-	-	2TB	-
System Monitoring Server	1	16 vCPUs	24GB	500GB	30Mbps

- **Using Kubernetes:**

	Quantity	CPU	Memory	Storage	Network
2025					
App & Consumer Node	3	>128 vCPUs	256 GB	1.5 TB	1Gbps
Database Server	2	32 vCPUs	64 GB	2TB	500Mbps
RabbitMQ Server	2	32 vCPUs	64 GB	1TB (SSD)	1Gbps
Load Balancer	2	16 vCPUs	32 GB	200GB	1Gbps
Hybrid SAN	1	-	-	2TB	-
System Monitoring Server	1	16 vCPUs	24GB	500GB	30Mbps

Note: Please refer to Infrastructure **Projection Report (2025-2028).pdf** for the storage project for the year 2025 to 2028

13. Monitoring & Logging

1. Monitoring Using Prometheus and Grafana

Prometheus:

Prometheus is a time-series database designed for monitoring and alerting. It collects metrics from applications and systems, stores them, and allows querying to identify trends and issues.

Grafana:

Grafana is a visualization and analytics tool that integrates with Prometheus to display metrics in customizable dashboards. It makes complex data easier to interpret by presenting it graphically.

How will it work:

1. **Data Collection:**
    - Prometheus collects metrics from various sources via its scraping mechanism. Metrics are exposed by applications using the /metrics endpoint (usually via libraries like prom-client in Node.js or prometheus-client in Python).

- Example metrics include:
    - CPU usage, memory consumption, disk IO, etc. (from servers and VMs).
    - Number of active RabbitMQ consumers.
    - Application request rates and error rates.

2. **Data Storage:**
    - Prometheus stores metrics in its time-series database for querying and analysis.
3. **Alerts:**
    - Prometheus uses Alertmanager to send alerts (via email, Slack, PagerDuty, etc.) based on predefined thresholds (e.g., CPU > 80% for more than 5 minutes).
4. **Visualization:**
    - Grafana connects to Prometheus and provides dashboards with various widgets (graphs, gauges, heatmaps, etc.) to visualize system performance.
    - Examples:
        - RabbitMQ queue length over time.
        - CPU and memory usage trends.
        - Error rates in application APIs.

---

2. Logging Using ELK (Elasticsearch, Logstash, Kibana)

ELK Stack Overview:

The ELK stack is a popular solution for centralized logging:

1. **Elasticsearch:** A distributed search and analytics engine where logs are stored and indexed for fast querying.
2. **Logstash:** A data-processing pipeline that ingests logs from multiple sources, processes them (e.g., parsing, filtering, enriching), and sends them to Elasticsearch.
3. **Kibana:** A visualization tool for logs, integrated with Elasticsearch. It provides search and filtering capabilities and customizable dashboards for log analysis.

How will it work:

1. **Log Generation:**
    - Applications, RabbitMQ, and other components generate logs (e.g., HTTP request logs, error logs, system logs).
    - These logs can be written to files, stdout/stderr streams, or log aggregation systems.

2. **Log Collection with Logstash:**
    - Logstash collects logs from various sources:
        - Application log files.
        - Syslog from servers.
        - Docker or Kubernetes logs.
    - It processes the logs (e.g., parses JSON, removes noise, adds metadata) and sends them to Elasticsearch.
3. **Log Indexing in Elasticsearch:**
    - Logs are stored in Elasticsearch as searchable, structured data.
    - Elasticsearch supports full-text search, allowing users to query logs based on criteria like time, log level, application ID, or specific error messages.
4. **Visualization in Kibana:**
    - Kibana provides a graphical interface to explore, search, and analyze logs.
    - Example Use Cases:
        - Filter logs by campaign ID or consumer ID.
        - Analyze error patterns over time.
        - Visualize system-wide log activity across multiple nodes.

14. Project Phases, Milestone & Deliverables

➢ **Project Phases, Milestones, and Deliverables (13-Week Plan)**

Phase 1: Planning and Requirements Gathering (Weeks 1–2)

- **Activities:**
    - Conduct project kickoff meeting with stakeholders.
    - Gather requirements for the SMS Gateway, including security protocols and compliance standards.
    - Finalize architecture and technology stack (VM/K8s).
    - Create a detailed project plan, including resource allocation and timeline.
- **Milestones:**
    - Requirements document signed off.
    - Initial system architecture approval.

- **Deliverables:**
    - DFD and ER Diagrams: Provide Data Flow Diagrams and Entity-Relationship diagrams outlining the SMS Gateway system.

Phase 2: System Design and Environment Setup (Weeks 3–4)

- **Activities:**
    - Finalize infrastructure setup for VM and Kubernetes environments.
    - Set up Prometheus, Grafana (monitoring), and ELK stack (logging).
    - Design and document API architecture, database schema, and RabbitMQ configurations.
- **Milestones:**
    - Infrastructure ready for development and testing.
    - Finalized architecture for API Gateway and message delivery systems.
- **Deliverables:**
    - Updated DFD/ER diagrams reflecting system design changes.
    - Initial setup of monitoring and logging tools.

Phase 3: Development and Implementation (Weeks 5–8)

- **Activities:**
    - Develop SMS Gateway components:
        - Message Processing Engine.
        - Message Queuing System with RabbitMQ.
        - Redis cache integration.
        - Health-check endpoints for the API Gateway.
    - Implement security protocols, such as HTTPS/TLS, role-based access control, and secure APIs.
    - Implement database systems for primary and replica configurations.
    - Develop APIs for message submission, delivery tracking, and reporting.
    - Configure autoscaling in Kubernetes and load balancing for high availability.
- **Milestones:**

- Core functionalities of the SMS Gateway completed.
- Secure API Gateway implemented.

- **Deliverables:**
  - Fully implemented and functional SMS Gateway.

Phase 4: Testing and Quality Assurance (Weeks 9–10)

- **Activities:**
  - Perform unit, integration, and system testing.
  - Conduct performance and stress testing to validate scalability and high availability.
  - Fix issues identified during testing.
- **Milestones:**
  - Testing completed with all critical issues resolved.
- **Deliverables:**
  - Health Check Report: Provide a detailed health check analysis report for application and database components.

Phase 5: Training, Documentation, and Handover (Weeks 11–12)

- **Activities:**
  - Create user manuals and operational documentation for the system.
  - Train NAGAD's team on system operations, including monitoring tools, logging, and troubleshooting.
- **Milestones:**
  - Documentation and training completed.
- **Deliverables:**
  - Training & Documentation: Operational documents and hands-on training provided to NAGAD's team.
  - Finalized system documentation (DFD/ER diagrams, API guides, etc.).

Phase 6: Deployment and Project Closure (Week 13)

- **Activities:**

- Deploy the SMS Gateway to production.
- Conduct final health checks and validations post-deployment.
- Deliver project summary and final report.

- **Milestones:**
  - Successful system deployment to production.
  - Project closure meeting conducted.
- **Deliverables:**
  - Fully deployed SMS Gateway system.
  - Final project report with all system details and a health check summary.

Summary of Key Deliverables:

1. Fully implemented and tested SMS Gateway with compliance to standard security protocols.
2. Health Check Report for application and database components.
3. Data Flow Diagrams (DFD) and ER Diagrams for the solution.
4. Regular status reports and a final project report.
5. Training and operational documentation for system maintenance by NAGAD's team.

Project Timeline Summary:

Phase	Weeks	Key Deliverables
Planning & Requirements	Weeks 1–2	DFD/ER diagrams.
System Design & Setup	Weeks 3–4	Initial setup of infrastructure, monitoring, and logging tools.
Development	Weeks 5–8	Fully implemented SMS Gateway.
Testing	Weeks 9–10	Health Check Report for application and database.
Training & Documentation	Weeks 11–12	Operational documentation and hands-on training.
Deployment & Closure	Week 13	Fully deployed system and final project report.

15. Project Team

Team Structure and Allocation

Role	No. of Members	Time Commitment
Project Manager	1	Full-time throughout the project.
Solution Architect	1	Full-time during design phases.
Backend Developers	2–3	Full-time during development.
Frontend Developer	1	Full-time during development.
DevOps Engineer	1	Full-time during setup and deployment.
Database Administrator	1	Part-time during setup and optimization.
QA Engineer	1	Full-time during testing phases.
Technical Writer	1	Part-time during documentation phases.
Trainer	1	Part-time during training phases.

16. Testing Strategy

1. API Security

Use OAuth 2.0 for secure API authentication and authorization.

Implement rate limiting to prevent API abuse.

Encrypt API traffic using HTTPS/TLS.

2. Data Security

Encrypt sensitive data (e.g., user information, SMS content) at rest using AES-256 encryption.

Ensure all communication between components (e.g., RabbitMQ, database) is encrypted using SSL/TLS.

3. Access Control

Apply Role-Based Access Control (RBAC) to limit access to sensitive data and administrative functionality.

Implement two-factor authentication (2FA) for admin-level accounts.

4. Secure Logging

Ensure logs do not contain sensitive data like passwords or SMS content.

Secure logs using encryption and limit access to authorized personnel only.

5. Protection Against Attacks

Use firewalls and WAFs (Web Application Firewalls) to block malicious traffic.

Implement input validation and output sanitization to protect against SQL injection, XSS, and other vulnerabilities.

Conduct regular penetration testing to identify and mitigate vulnerabilities.

6. Compliance

Ensure the system adheres to industry security standards and local compliance regulations (e.g., GDPR, ISO 27001).

17. Detailed System Architecture

A comprehensive testing strategy will ensures that the system is reliable, scalable, and secure. Below are the testing stages and methodologies:

1. Unit Testing

Objective: Test individual modules (e.g., API endpoints, RabbitMQ consumers) for correct functionality.

Tools: Mocha, Jest, or JUnit.

2. Integration Testing

Objective: Verify that components (e.g., API Gateway, database, RabbitMQ) interact correctly.

Tools: Postman, Newman, or custom integration scripts.

3. System Testing

Objective: Validate the entire system, including end-to-end workflows such as SMS sending, queuing, and delivery.

Tools: Selenium, JMeter.

4. Performance Testing

Objective: Test system scalability under high TPS (Transactions Per Second).

Tools: Apache JMeter, Locust.

5. Security Testing

Objective: Identify vulnerabilities in the system.

Tools: OWASP ZAP, Burp Suite, Nessus.

6. User Acceptance Testing (UAT)

Objective: Ensure the system meets the business requirements and is user-friendly.

Stakeholders: Conducted with Nagad's operational team.

7. Regression Testing

Objective: Verify that new changes do not break existing functionality.

Tools: Automated regression testing scripts.

8. Monitoring and Health Check Validation

Simulate system failures to validate the monitoring tools (Prometheus, Grafana) and ensure alert mechanisms work as intended.

18. Conclusion

The SMS Gateway project is designed to deliver a highly scalable, secure, and efficient communication platform for Nagad. By leveraging modern technologies such as Kubernetes, RabbitMQ, and Redis, the system ensures high availability, fault tolerance, and seamless scalability for both VMs and Kubernetes environments.

The project will:

1. Meet all functional and non-functional requirements, including scalability and security.
2. Provide detailed documentation, training, and support for operational sustainability.
3. Maintain compliance with industry standards and regulatory requirements.

Through well-defined phases, milestones, and a skilled project team, the project will be delivered within the 13-week timeframe. The comprehensive testing strategy and robust security measures will ensure a reliable and secure system for Nagad's communication needs.

Signature	:
Name of the signatory	: Mohammad Tanin Islam
Desiginataion of the Signatory	: Managing Director
Name of the Company	: Divergent Technologies Limited
Date	: 26.11.2024