1. High Level Data Flow Diagrams:

These diagrams and explanations illustrate the flow of Inbound, Outbound, Push, and Pull SMS processes within the SMS Gateway System for Nagad Ltd.

Inbound SMS Process:

In the Inbound SMS process, a user sends a message to a specific number provided by Nagad, such as a query about their account balance or a request for a service. The Mobile Network Operator (MNO) receives this SMS and forwards it to the SMS Gateway System. The API Gateway within the SMS Gateway System validates and routes the message to the Message Processing Engine. The engine processes the SMS, applies the necessary business logic (e.g., identifying the type of request), and forwards it to the Nagad DFS System. The DFS System then processes the request and takes the appropriate action, such as sending a response back to the user or updating the user's account information.

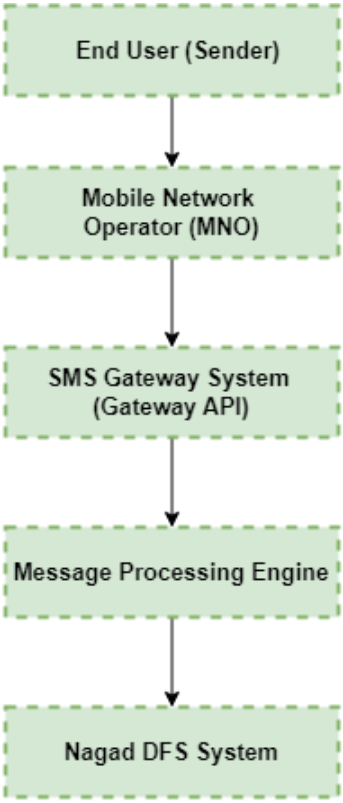

Fig: SMSGW - Inbound SMS Process

Explanation:

1. **End User (Sender):** The user sends an SMS to a designated number.

2. **Mobile Network Operator (MNO):** The MNO receives the SMS and forwards it to the SMS Gateway System.
3. **SMS Gateway System (API Gateway):** The API Gateway receives the SMS, validates it, and routes it to the Message Processing Engine.
4. **Message Processing Engine:** The engine processes the SMS, applies business logic, and forwards it to the Nagad DFS System.
5. **Nagad DFS System:** The DFS System receives the SMS and processes it according to the business rules (e.g., transactional messages, queries).

Outbound SMS Process:

In the Outbound SMS process, the Nagad DFS System generates an SMS to be sent to a user, such as a transactional alert (e.g., a confirmation of a money transfer) or a promotional message (e.g., a discount offer). The Message Processing Engine receives this SMS, applies the necessary business logic (e.g., prioritizing the message based on its type), and forwards it to the SMS Gateway System. The API Gateway within the SMS Gateway System validates the message and routes it to the Mobile Network Operator (MNO). The MNO then delivers the SMS to the end user's mobile device, ensuring that the user receives the important notification or offer.

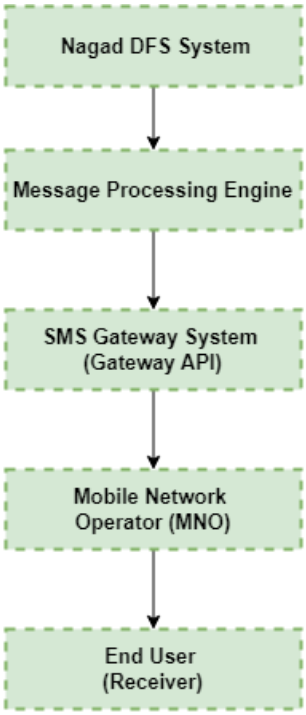

Fig: SMSGW - Outbound SMS Process

Explanation:

1. **Nagad DFS System:** The DFS System generates an SMS (e.g., transactional alert, promotional message).

2. **Message Processing Engine:** The engine processes the SMS, applies business logic, and forwards it to the SMS Gateway System.
3. **SMS Gateway System (API Gateway):** The API Gateway receives the SMS, validates it, and routes it to the Mobile Network Operator (MNO).
4. **Mobile Network Operator (MNO):** The MNO receives the SMS and forwards it to the end user.
5. **End User (Receiver):** The user receives the SMS on their mobile device.

Push SMS Process

In the Push SMS process, the Nagad DFS System generates a push SMS to be sent to a user, such as a promotional message (e.g., a new product announcement) or an alert (e.g., a security notification). The Message Processing Engine receives this SMS, applies the necessary business logic (e.g., prioritizing the message based on its urgency), and forwards it to the SMS Gateway System. The API Gateway within the SMS Gateway System validates the message and routes it to the Mobile Network Operator (MNO). The MNO then delivers the push SMS to the end user's mobile device, ensuring that the user receives the timely and relevant information.

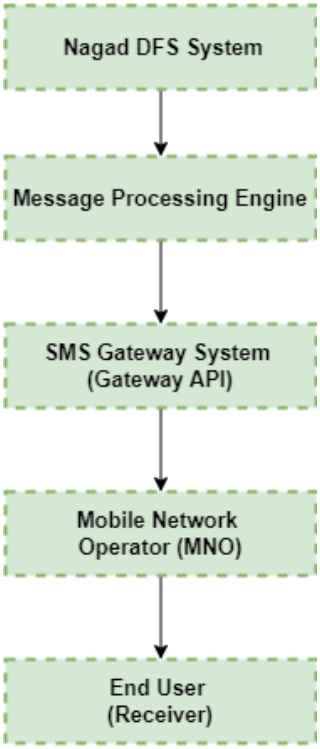

Fig: Push SMS Process

Explanation:

1. **Nagad DFS System:** The DFS System generates a push SMS (e.g., promotional message, alert).
2. **Message Processing Engine:** The engine processes the SMS, applies business logic, and forwards it to the SMS Gateway System.

3. **SMS Gateway System (API Gateway):** The API Gateway receives the SMS, validates it, and routes it to the Mobile Network Operator (MNO).
4. **Mobile Network Operator (MNO):** The MNO receives the SMS and forwards it to the end user.
5. **End User (Receiver):** The user receives the push SMS on their mobile device.

Pull SMS Process

In the Pull SMS process, a user sends a message to a specific number provided by Nagad, such as a query about their account balance or a request for a service. The Mobile Network Operator (MNO) receives this SMS and forwards it to the SMS Gateway System. The API Gateway within the SMS Gateway System validates and routes the message to the Message Processing Engine. The engine processes the SMS, applies the necessary business logic (e.g., identifying the type of request), and forwards it to the Nagad DFS System. The DFS System then processes the request and takes the appropriate action, such as sending a response back to the user or updating the user's account information. This process ensures that user queries and requests are handled efficiently and accurately.

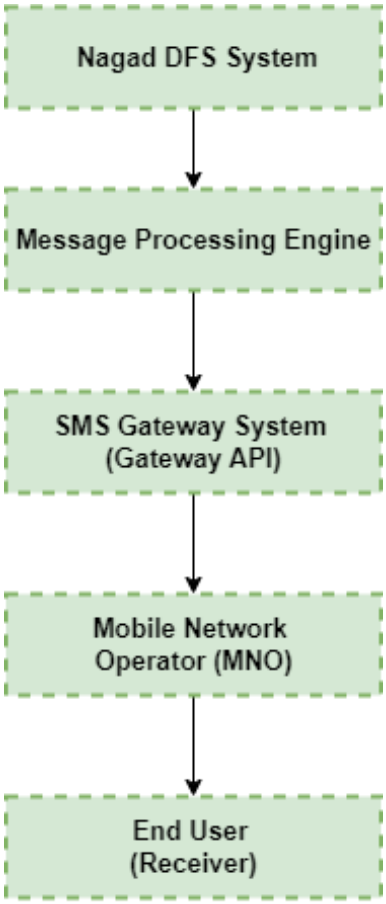

Fig: Pull SMS Process

Explanation:

1. **End User (Sender):** The user sends an SMS to a designated number (e.g., query, request).
2. **Mobile Network Operator (MNO):** The MNO receives the SMS and forwards it to the SMS Gateway System.
3. **SMS Gateway System (API Gateway):** The API Gateway receives the SMS, validates it, and routes it to the Message Processing Engine.
4. **Message Processing Engine:** The engine processes the SMS, applies business logic, and forwards it to the Nagad DFS System.
5. **Nagad DFS System:** The DFS System receives the SMS and processes it according to the business rules (e.g., responding to a query).

These diagrams and explanations illustrate the flow of Inbound, Outbound, Push, and Pull SMS processes within the SMS Gateway System for Nagad Ltd.

2. Server and Storage Specification for Kubernetes Cloud based Cluster:

Server Specifications:

- **App & Consumer Nodes:**
  - **Quantity:** 3 nodes
  - **CPU:** More than 128 vCPUs (distributed across the nodes)
  - **Memory:** 256 GB (distributed across the nodes)
  - **Storage:** 1.5 TB (distributed across the nodes)
  - **Network:** 1 Gbps
- **Database Servers:**
  - **Quantity:** 2 nodes
  - **CPU:** 32 vCPUs per node
  - **Memory:** 64 GB per node
  - **Storage:** 2 TB per node
  - **Network:** 500 Mbps
- **RabbitMQ Servers:**
  - **Quantity:** 2 nodes
  - **CPU:** 32 vCPUs per node
  - **Memory:** 64 GB per node

- **Storage:** 1 TB SSD per node
- **Network:** 1 Gbps

- **Load Balancers:**
  - **Quantity:** 2 nodes
  - **CPU:** 16 vCPUs per node
  - **Memory:** 32 GB per node
  - **Storage:** 200 GB per node
  - **Network:** 1 Gbps
- **Hybrid SAN:**
  - **Quantity:** 1 node
  - **Storage:** 2 TB
- **System Monitoring Server:**
  - **Quantity:** 1 node
  - **CPU:** 16 vCPUs
  - **Memory:** 24 GB
  - **Storage:** 500 GB
  - **Network:** 30 Mbps

Storage Specifications:

- **Total Storage Requirement:**
  - **App & Consumer Nodes:** 1.5 TB
  - **Database Servers:** 4 TB (2 TB per node)
  - **RabbitMQ Servers:** 2 TB SSD (1 TB per node)
  - **Load Balancers:** 400 GB (200 GB per node)
  - **Hybrid SAN:** 2 TB
  - **System Monitoring Server:** 500 GB
- **Storage Type:**
  - **App & Consumer Nodes:** High-performance storage (SSD preferred)
  - **Database Servers:** High-performance storage (SSD preferred)
  - **RabbitMQ Servers:** High-performance SSD storage

- **Load Balancers:** Standard storage
- **Hybrid SAN:** High-capacity storage
- **System Monitoring Server:** Standard storage

3. Infrastructure Dimensions/Specifications for different nodes:

App Server Nodes:

- Quantity: 2 nodes
- CPU: 32 vCPUs per node
- Memory: 64 GB per node
- Storage: 1 TB per node
- Network: 1 Gbps per node

Consumer Nodes:

- Quantity: More than 10 nodes
- CPU: 32 vCPUs per node
- Memory: 64 GB per node
- Storage: 150 GB per node
- Network: 1 Gbps per node

Database Servers:

- Quantity: 2 nodes
- CPU: 32 vCPUs per node
- Memory: 64 GB per node
- Storage: 2 TB per node
- Network: 500 Mbps per node

RabbitMQ Servers:

- Quantity: 2 nodes
- CPU: 32 vCPUs per node
- Memory: 64 GB per node
- Storage: 1 TB SSD per node
- Network: 1 Gbps per node

Redis Server:

- Quantity: 1 node
- CPU: 16 vCPUs
- Memory: 32 GB
- Storage: 500 GB SSD
- Network: 500 Mbps

Load Balancers:

- Quantity: 3 nodes
- CPU: 16 vCPUs per node
- Memory: 32 GB per node
- Storage: 200 GB per node
- Network: 1 Gbps per node

Hybrid SAN:

- Quantity: 1 node
- Storage: 2 TB

System Monitoring Server:

- Quantity: 1 node
- CPU: 16 vCPUs
- Memory: 24 GB
- Storage: 500 GB
- Network: 30 Mbps

4. Detailed zone-based Network architecture highlighting Database, Nodes, Infra, DMZ, Core.

Zone-Based Network Architecture

The network architecture is divided into several zones to ensure security, scalability, and efficient management. The zones include:

1. **DMZ (Demilitarized Zone):** This zone acts as a buffer between the external untrusted network and the internal trusted network. It hosts services that need to be accessible from the outside, such as the API Gateway.

2. **Core Zone:** This zone contains the core infrastructure components, including the application servers, message processing engines, and other critical services.
3. **Database Zone:** This zone hosts the database servers, ensuring data security and integrity.
4. **Infrastructure Zone:** This zone includes the underlying infrastructure components, such as load balancers, monitoring tools, and storage systems

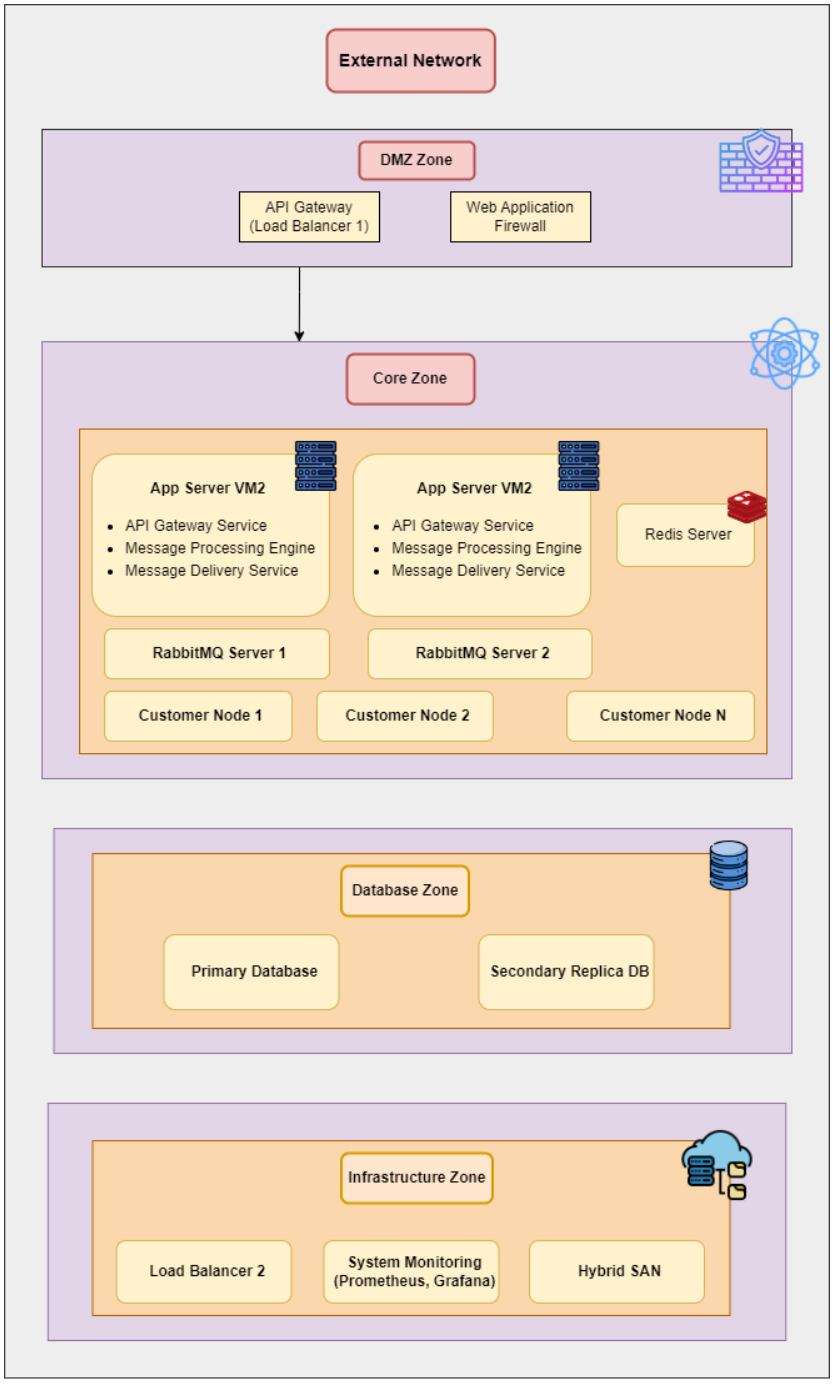

Fig: Overall Network Architecture

Detailed Component Interactions:

1. **DMZ Zone:**
    - API Gateway (Load Balancer 1): Handles incoming API requests, validates, and routes them to the appropriate services in the Core Zone.
    - Web Application Firewall (WAF): Protects the API Gateway from external threats and unauthorized access.
2. **Core Zone:**
    - App Server VM1 and VM2: Host multiple services, including the API Gateway Service, Message Processing Engine, Message Delivery Service, and Redis Server for caching.
    - RabbitMQ Server 1 and 2: Manage message queuing with mirrored queues for high availability.
    - Consumer Nodes: Process messages from RabbitMQ and interact with the Message Delivery Service.
3. **Database Zone:**
    - Primary Database: Handles write operations and stores critical data.
    - Secondary Replica DB: Provides a replica for read operations to reduce load on the primary database.
4. **Infrastructure Zone:**
    - Load Balancer 2: Distributes traffic to RabbitMQ Servers, Databases, and Redis Server.
    - System Monitoring (Prometheus, Grafana): Monitors system performance and health.
    - Hybrid SAN: Provides storage for the system, including All-Flash and Hybrid configurations.

Explanation of Component Interactions:

1. **External Network to DMZ Zone:**
    - External systems (e.g., partner APIs, user devices) send requests to the API Gateway in the DMZ Zone.
    - The WAF filters and protects these requests before they reach the API Gateway.
2. **DMZ Zone to Core Zone:**
    - The API Gateway validates and routes the requests to the appropriate services in the Core Zone (e.g., Message Processing Engine).

- The Message Processing Engine processes the requests, applies business logic, and interacts with other services (e.g., Message Delivery Service, RabbitMQ).

3. **Core Zone to Database Zone:**
   - The Message Processing Engine and other services interact with the Primary Database for write operations and the Secondary Replica DB for read operations.
   - The Redis Server provides caching for fast data access.
4. **Infrastructure Zone:**
   - Load Balancer 2 distributes traffic to ensure high availability and efficient resource utilization.
   - System Monitoring tools (Prometheus, Grafana) monitor the health and performance of the system.
   - The Hybrid SAN provides storage for the system, ensuring data integrity and availability.

This zone-based network architecture ensures security, scalability, and efficient management of the SMS Gateway Solution for Nagad Ltd.

5. Data flow Diagram and ERD for Database

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

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

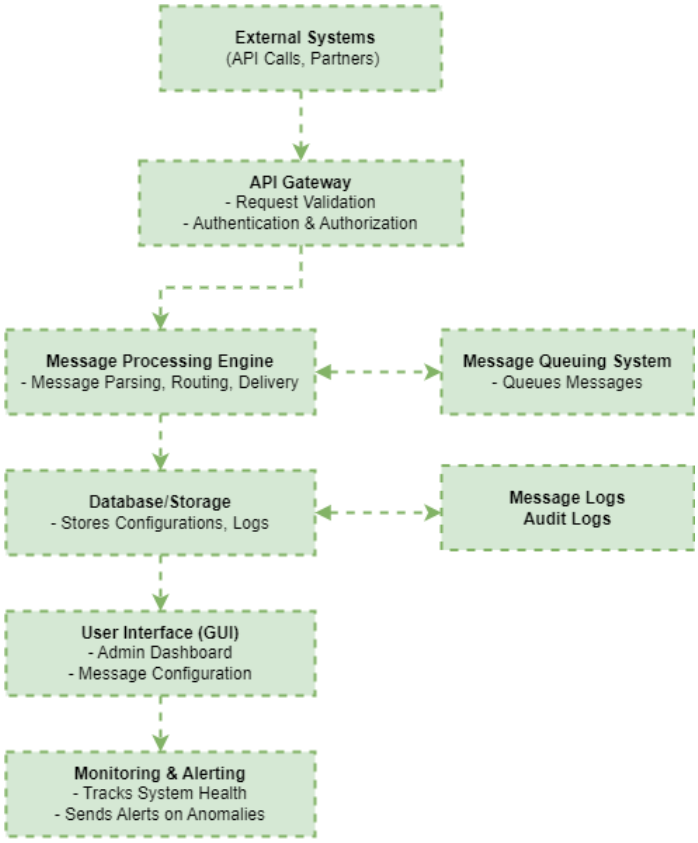

Fig: DFD - System Components Interaction

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

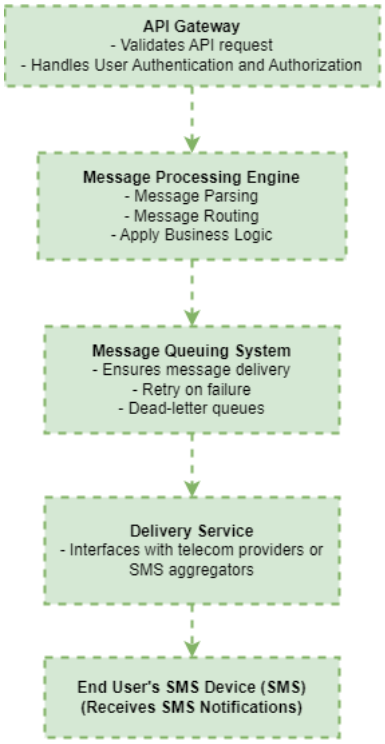

Fig: DFD - Detailed Process Flow of the Message

- **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.

➢ Entity-Relationship Diagram (ERD):

Entities and Relationships

1. **User**
   - UserID (PK)
   - Username
   - Password
   - Email
   - PhoneNumber
   - CreatedAt
   - UpdatedAt
2. **Role**
   - RoleID (PK)
   - RoleName
   - Description
3. **Permissions**
   - PermissionID (PK)
   - PermissionName
   - Description
4. **Role_Permissions**
   - RoleID (FK)
   - PermissionID (FK)
5. **User_Role**
   - UserID (FK)
   - RoleID (FK)
6. **Sender**
   - SenderID (PK)
   - Name
   - Type
   - CreatedAt
   - UpdatedAt
7. **Message**
   - MessageID (PK)
   - Content
   - Type
   - Priority
   - Status
   - SenderID (FK)
   - CampaignID (FK)
   - ScheduleTime

- RetryCount
- MaskingCode
- MNOID (FK)
- ChannelID (FK)
- Archived (Boolean)
- ArchiveTimestamp

8. **Backup**
    - BackupID (PK)
    - Timestamp
    - Type
    - Status
9. **Campaign**
    - CampaignID (PK)
    - Name
    - Description
    - StartDate
    - EndDate
    - Status
10. **Queue**
    - QueueID (PK)
    - Name
    - Priority
11. **Campaigns_Queue**
    - CampaignID (FK)
    - QueueID (FK)
12. **MNO**
    - MNOID (PK)
    - Name
    - Country
13. **Channel**
    - ChannelID (PK)
    - Name
    - Type
    - Priority
14. **MNO_Channels**
    - MNOID (FK)
    - ChannelID (FK)
15. **MessageArchive**
    - ArchiveID (PK)
    - MessageID (FK)
    - ArchiveTimestamp
    - ArchiveStatus

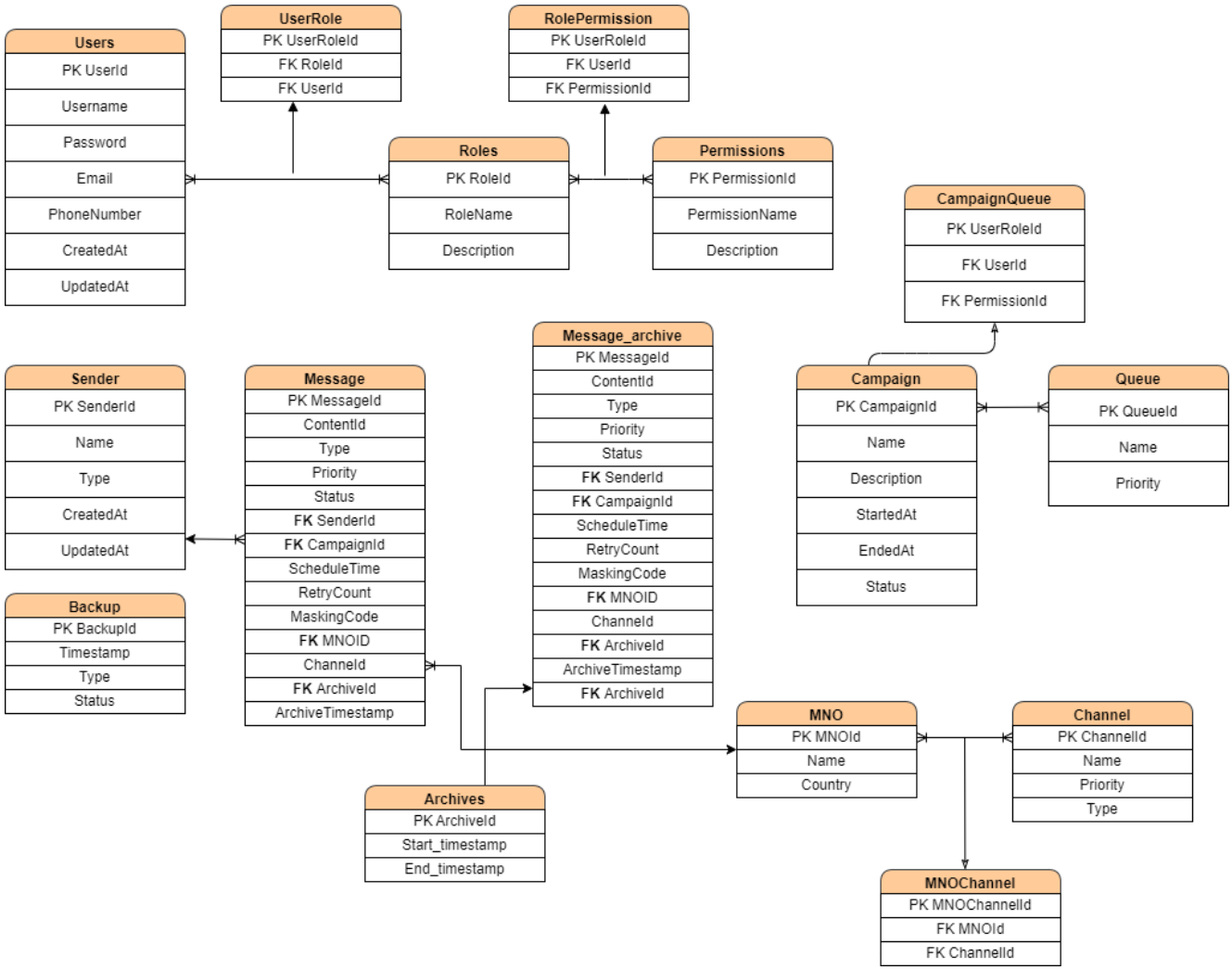

Fig: E-R Diagram

Detailed Explanation of Components

1. **User:**
    - UserID (PK): Unique identifier for each user.
    - Username: Username of the user.
    - Password: Password of the user.
    - Email: Email address of the user.
    - PhoneNumber: Contact number of the user.
    - CreatedAt: Timestamp of when the user was created.
    - UpdatedAt: Timestamp of when the user was last updated.
2. **Role:**
    - RoleID (PK): Unique identifier for each role.
    - RoleName: Name of the role.
    - Description: Description of the role.
3. **Permissions:**
    - PermissionID (PK): Unique identifier for each permission.
    - PermissionName: Name of the permission.
    - Description: Description of the permission.
4. **Role_Permissions:**
    - RoleID (FK): Foreign key referencing the role.
    - PermissionID (FK): Foreign key referencing the permission.
5. **User_Role:**
    - UserID (FK): Foreign key referencing the user.
    - RoleID (FK): Foreign key referencing the role.
6. **Sender:**
    - SenderID (PK): Unique identifier for each sender.
    - Name: Name of the sender.
    - Type: Type of the sender (e.g., individual, business).
    - CreatedAt: Timestamp of when the sender was created.
    - UpdatedAt: Timestamp of when the sender was last updated.

7. **Message:**
    - MessageID (PK): Unique identifier for each message.
    - Content: Content of the message.
    - Type: Type of the message (e.g., transactional, promotional).
    - Priority: Priority of the message.
    - Status: Status of the message (e.g., sent, delivered, failed).
    - SenderID (FK): Foreign key referencing the sender.
    - CampaignID (FK): Foreign key referencing the campaign.
    - ScheduleTime: Timestamp of when the message is scheduled to be sent.
    - RetryCount: Number of retry attempts for the message.
    - MaskingCode: Masking code for the message.
    - MNOID (FK): Foreign key referencing the MNO.
    - ChannelID (FK): Foreign key referencing the channel.
    - Archived: Boolean indicating whether the message is archived.
    - ArchiveTimestamp: Timestamp of when the message was archived.
8. **Backup:**
    - BackupID (PK): Unique identifier for each backup.
    - Timestamp: Timestamp of when the backup was created.
    - Type: Type of the backup (e.g., full, incremental).
    - Status: Status of the backup (e.g., completed, failed).
9. **Campaign:**
    - CampaignID (PK): Unique identifier for each campaign.
    - Name: Name of the campaign.
    - Description: Description of the campaign.
    - StartDate: Start date of the campaign.
    - EndDate: End date of the campaign.
    - Status: Status of the campaign (e.g., active, completed).
10. **Queue:**
    - QueueID (PK): Unique identifier for each queue.

- Name: Name of the queue.
- Priority: Priority of the queue.

11. Campaigns_Queue:

- CampaignID (FK): Foreign key referencing the campaign.
- QueueID (FK): Foreign key referencing the queue.

12. MNO:

- MNOID (PK): Unique identifier for each MNO.
- Name: Name of the MNO.
- Country: Country of the MNO.

13. Channel:

- ChannelID (PK): Unique identifier for each channel.
- Name: Name of the channel.
- Type: Type of the channel (e.g., HTTP, SMPP).
- Priority: Priority of the channel.

14. MNO_Channels:

- MNOID (FK): Foreign key referencing the MNO.
- ChannelID (FK): Foreign key referencing the channel.

15. MessageArchive:

- ArchiveID (PK): Unique identifier for each archive entry.
- MessageID (FK): Foreign key referencing the message.
- ArchiveTimestamp: Timestamp of when the message was archived.
- ArchiveStatus: Status of the archive (e.g., active, deleted).

➢ **Partitioning Strategy**

To ensure scalability and performance, the database can be partitioned based on the following strategies:

1. Range Partitioning:
    - **Messages Table:** Partition the Messages table based on the ScheduleTime or ArchiveTimestamp. This allows for distributing the message data across multiple database nodes based on the time range.

- **Backup Table:** Partition the Backup table based on the Timestamp. This allows for distributing the backup data across multiple database nodes based on the time range.

➢ **Indexing:**
- Create indexes on frequently queried columns (e.g., UserID, MessageID, CampaignID, SenderID, MNOID, ChannelID) to improve query performance.

➢ **Caching:**
- Implement caching for frequently accessed data (e.g., user profiles, message templates) using a caching solution like Redis or Memcached.

➢ **Replication:**
- Set up database replication to ensure high availability and data redundancy. A master-slave replication setup will be used where the master database handles write operations and the slave databases handle read operations.

➢ **Security:**
- Encryption for sensitive data will be implemented (e.g., user passwords, message content) both at rest and in transit.
- Use role-based access control (RBAC) to manage user permissions and access to different parts of the system.
- Implement audit logging to track changes to the data and system configurations.

➢ **Monitoring and Alerting:**
- Set up monitoring and alerting for the database and application layers using tools like Prometheus and Grafana.
- Monitor key metrics such as database performance, query latency, and system health.
- Set up alerts for critical events such as database failures, high query latency, and security breaches.

6. Details on High Availability feature, Archiving solution:

➢ **<u>High Availability:</u>**

High availability (HA) is a critical aspect of the SMS Gateway Solution, ensuring that the system remains operational and accessible even in the event of hardware or software failures. Here are the details on the high availability features and strategies implemented in the SMS Gateway Solution:

1. Redundancy and Failover

Redundancy:

- **Multiple Application Servers:** The system employs multiple application servers (e.g., App Server VM1, App Server VM2) to handle incoming API requests and process messages. This ensures that if one server fails, another can take over without interrupting the service.

- **Mirrored RabbitMQ Queues:** RabbitMQ servers are configured with mirrored queues, ensuring that messages are replicated across multiple RabbitMQ servers. If one RabbitMQ server fails, the other can seamlessly take over without data loss.

- **Primary and Replica Databases:** The database layer uses a master-slave configuration, where the primary database handles write operations and the replica database handles read operations. If the primary database fails, the replica can take over to ensure data availability.

Failover:

- **Automatic Failover:** The system supports automatic failover for critical components. For example, if the primary database becomes unavailable, the system automatically switches to the replica database.
- **Manual Failover:** In cases where automatic failover is not possible, the system supports manual failover options, allowing administrators to switch to backup components manually.

2. Load Balancing

Load Balancers:

- **Load Balancer 1:** Distributes incoming API requests across multiple application servers, ensuring that no single server becomes a bottleneck. This helps in balancing the load and ensuring high availability.
- **Load Balancer 2:** Distributes traffic to RabbitMQ servers, databases, and Redis servers, ensuring efficient resource utilization and high availability.

3. Health Checks and Self-Healing

Health Checks:

- **Liveness and Readiness Probes:** Kubernetes performs liveness and readiness probes to detect unhealthy pods. If a pod fails the health check, Kubernetes automatically restarts it on a healthy node.
- **Monitoring and Alerting:** The system uses monitoring tools like Prometheus and Grafana to track the health and performance of the system. Alerts are configured to notify administrators of any anomalies or failures.

Self-Healing:

- **Pod Restarts:** Kubernetes automatically restarts failed pods, ensuring that services remain available.
- **Node Failure:** If a node fails, Kubernetes reschedules the pods to healthy nodes, ensuring continuous service availability.

4. Autoscaling

Horizontal Pod Autoscaling (HPA):

- Kubernetes uses Horizontal Pod Autoscaling (HPA) to automatically adjust the number of pods based on resource usage or custom metrics. This ensures that the system can handle increased load without manual intervention.

- **Scaling Policies:** Autoscaling policies are configured to scale up or down based on CPU, memory, or custom metrics, ensuring optimal resource utilization.

5. Data Replication and Synchronization

Database Replication:

- The primary database is continuously replicated to the secondary replica database, ensuring data redundancy and availability.
- Synchronous Replication: Ensures that data is written to both the primary and replica databases simultaneously, providing strong consistency.

Message Queue Replication:

- RabbitMQ servers use mirrored queues to replicate messages across multiple servers, ensuring that messages are not lost in the event of a server failure.

6. Disaster Recovery

Disaster Recovery Plan:

- The system includes a comprehensive disaster recovery plan to ensure system continuity in case of catastrophic failure.
- **Backup and Restore:** Regular backups are taken, and a restore process is in place to recover the system to a known good state.
- **Recovery Point Objective (RPO) and Recovery Time Objective (RTO):** The system aims for an RPO of 1 minute and an RTO of 30 minutes, ensuring minimal data loss and quick recovery.

Proposed Archiving Solution

An effective archiving solution is essential for managing the lifecycle of SMS messages, ensuring compliance with data retention policies, and optimizing storage usage. Here is a proposed archiving solution for the SMS Gateway Solution:

1. Archiving Strategy

Archiving Criteria:

- **Age-Based Archiving:** Messages older than a specified period (e.g., 6 months) are automatically archived.
- **Status-Based Archiving:** Messages that have been successfully delivered and acknowledged are archived.
- **Priority-Based Archiving:** Low-priority messages are archived earlier than high-priority messages.

Archiving Process:

- **Automatic Archiving:** The system automatically archives messages based on the defined criteria.

- **Manual Archiving:** Administrators can manually trigger the archiving process for specific messages or batches of messages.

2. Archiving Workflow

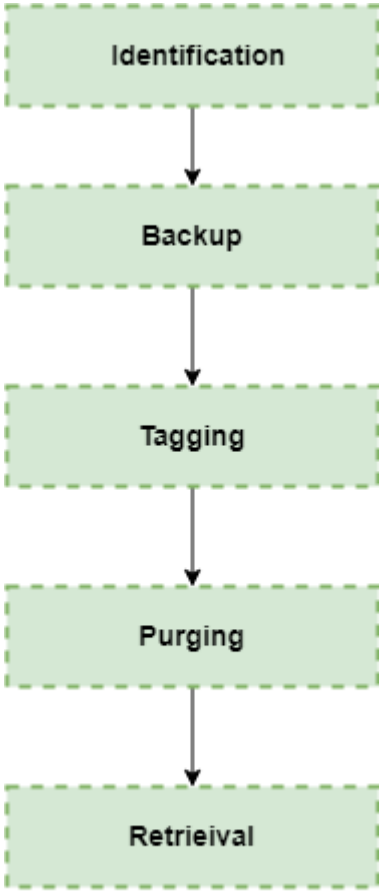

Fig: SMSGW - Workflow

1. **Identification:** The system identifies messages that meet the archiving criteria.
2. **Backup:** The identified messages are backed up to a separate archive storage.
3. **Tagging:** The archived messages are tagged with an archive status and timestamp.
4. **Purging:** The original messages are purged from the primary storage to free up space.
5. **Retrieval:** Archived messages can be retrieved from the archive storage if needed.

3. Archive Storage

Storage Solutions:

- **Cold Storage:** Archived messages are stored in cold storage solutions like Amazon S3 Glacier or Google Cloud Storage Nearline, which are cost-effective for long-term storage.
- **Compressed Storage:** Messages are compressed before archiving to reduce storage space.

Partitioning:

- **Time-Based Partitioning:** Archived messages are partitioned based on the archive timestamp, allowing for efficient retrieval and management.
- **Category-Based Partitioning:** Messages are partitioned based on categories (e.g., transactional, promotional) for easier management and retrieval.

7. Who will provide support module for open-source Database?

Support for the open-source database can be provided through these options:

- **Nagad In-House Support:** With the necessary expertise, an in-house support team can be established in Nagad Limited to manage and support the open-source database.
  Or,
- **Support from Divergent Technologies Ltd:** Divergent Technologies Ltd. may provide following supports in terms of the open-source database.
  - Technical Support
  - Security Support
  - Performance Tuning
  - Backup & Recovery
  - Monitoring & Alerting
  - Consulting & Advisory Support

Note: In case of support from Divergent Technologies Ltd. Additional cost will be incurred.

8. Can the system provide Custom reports with further details aside from the reports shown in demonstration (ex: Robi top up report, Grameen cashout report) and in such case, how.

Yes, the system will be designed to provide custom reports with further details aside from the standard reports shown in the demonstration.

<u>Custom Reporting Capabilities</u>

1. Flexible Reporting Framework

- **Dynamic Query Builder:** A dynamic query builder will be implemented that will allow users to create custom queries based on their specific reporting needs. This will be done through a user-friendly interface where users can select fields, apply filters, and define sorting and grouping criteria.
- **Parameterized Reports:** Parameterized reports will be enabled where users can input specific parameters (e.g., date range, MNO, campaign ID) to generate custom reports.

2. Integration with BI Tools

- **BI Tool Integration:** System will support Integration with the system with business intelligence (BI) tools such as Tableau, Power BI, or Looker to provide advanced reporting and visualization capabilities.
- **Data Export:** Users will be allowed to export report data in various formats (e.g., CSV, Excel, PDF) for further analysis and sharing.

3. **Real-Time Reporting**
    - **Real-Time Data Access:** The system will be capable to access and process real-time data to generate up-to-date reports.
    - **Live Dashboards:** Live dashboards will be provided that will display real-time metrics and KPIs related to SMS Gateway operations.

Information from Divergent:

1. Is it going to be a three-tier architecture implementation?

Yes, the SMS Gateway Solution will be implemented using a three-tier architecture. This separation will promote modularity, scalability, and maintainability. Here's how the SMS Gateway Solution can be implemented using a three-tier architecture:

Three-Tier Architecture Implementation:

1. Presentation Tier (Client Tier)

The presentation tier is the user interface layer where users interact with the application. It handles the display of information and user inputs.

Components:

- **Web GUI:** A web-based graphical user interface for administrators to configure, monitor, and manage the SMS Gateway. This includes dashboards, reports, and configuration settings.
- **API Clients:** Client applications or systems that interact with the SMS Gateway via APIs. These can include partner systems, external applications, and user devices.

Examples:

- **Admin Dashboard:** A web interface where administrators can view real-time message status, configure message templates, and manage user roles and permissions.
- **Partner API Integration:** External systems that send SMS messages via API calls to the SMS Gateway.

2. Application Tier (Logic Tier)

The application tier contains the business logic and processing components of the application. It handles the core functionality and processes user requests.

Components:

- **API Gateway:** Handles incoming API requests, validates them, and routes them to the appropriate services. It also manages authentication, authorization, and rate limiting.
- **Message Processing Engine:** Processes incoming SMS messages, applies business logic, and routes them to the appropriate delivery channels.
- **Message Queuing System:** Manages message queues to ensure reliable message delivery. It handles message retries and dead-letter queues.
- **Campaign Management Service:** Manages the creation, scheduling, and execution of SMS campaigns.
- **Event Notification Service:** Handles event-based notifications triggered by specific events in the system.
- **Monitoring and Alerting Service:** Monitors system performance, health, and logs. It sends alerts for critical events and anomalies.

  Examples:

  - **Message Routing:** The Message Processing Engine determines the best delivery route for each message based on factors such as sender, recipient, and message type.
  - **Campaign Scheduling:** The Campaign Management Service schedules and executes SMS campaigns based on predefined criteria.
  - **Real-Time Monitoring:** The Monitoring and Alerting Service tracks system performance and sends alerts for any anomalies or failures.

3. Data Tier (Database Tier)

The data tier is responsible for storing, retrieving, and managing the data used by the application. It includes databases and storage systems.

Components:

- **Primary Database:** Stores critical data such as user information, message metadata, and configuration settings.
- **Secondary Replica Database:** Provides a replica for read operations to reduce load on the primary database.
- **Message Archive:** Stores archived messages for compliance and audit purposes.
- **Backup and Restore:** Manages backup and restore operations to ensure data integrity and availability.
- **Redis Cache:** Provides caching for fast data access and session management.

2. How will the segregation take place in servers/services in DMZ & MZ zone?

Here's how the segregation can be implemented for the SMS Gateway Solution:

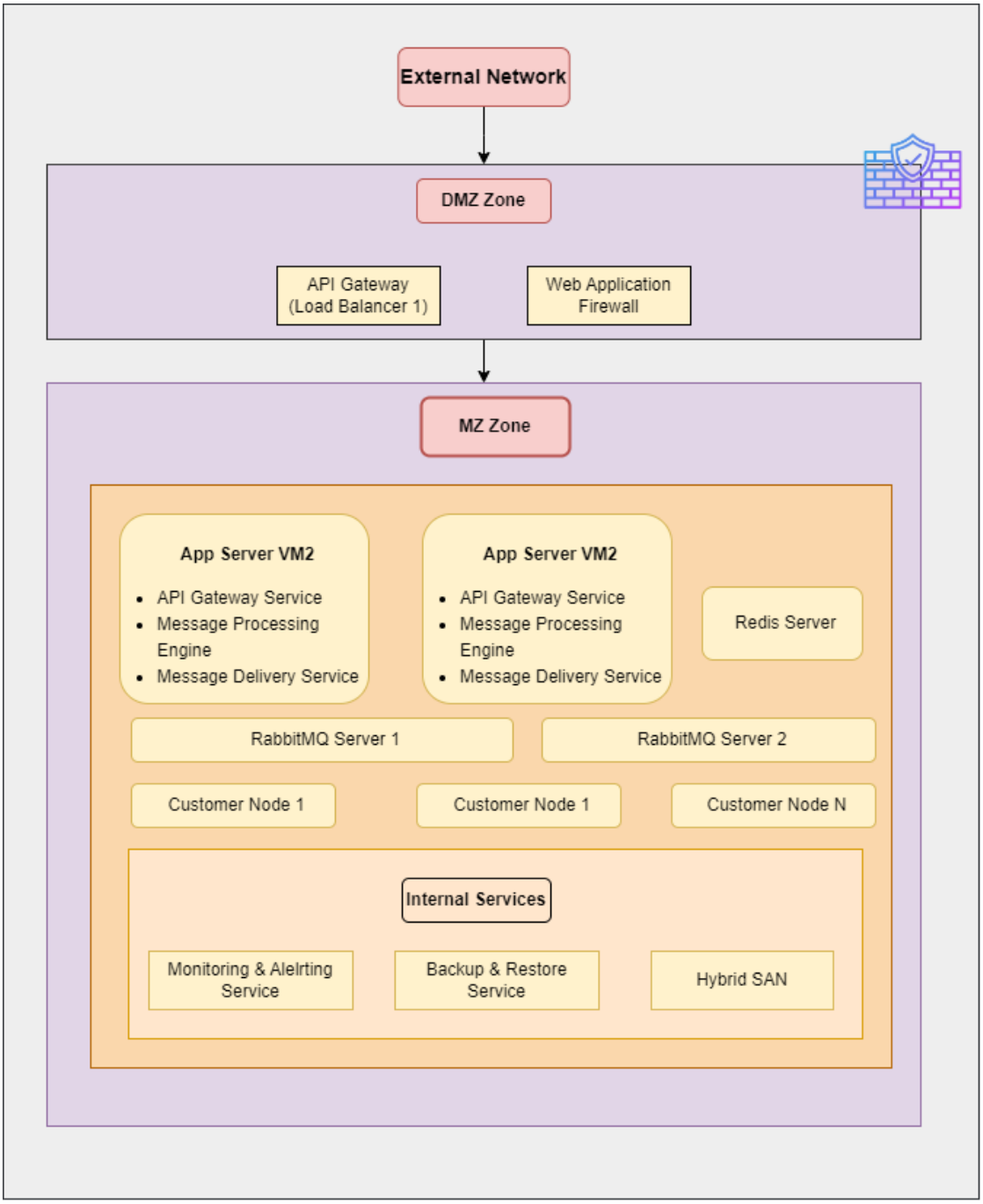

Fig: Segregation of components

Detailed Explanation of the Diagram

1. **External Network:**
    - External systems (e.g., partner APIs, user devices) send requests to the API Gateway in the DMZ.
2. **DMZ Zone:**
    - **API Gateway (Load Balancer 1):** Handles incoming API requests, validates them, and routes them to the appropriate services in the MZ.
    - **Web Application Firewall (WAF):** Protects the API Gateway from external threats and unauthorized access.
3. **MZ Zone:**
    - **App Server VM1 and VM2:** Host multiple services, including the API Gateway Service, Message Processing Engine, Message Delivery Service, and Redis Server for caching.
    - **RabbitMQ Server 1 and 2:** Manage message queuing with mirrored queues for high availability.
    - **Consumer Nodes:** Process messages from RabbitMQ and interact with the Message Delivery Service.
    - **Internal Services:** Include Monitoring and Alerting Service, Backup and Restore Service, and Hybrid SAN for storage.

Segregation and Communication Flow

1. **External to DMZ:**
    - **External systems** (e.g., partner APIs, user devices) will send requests to the API Gateway in the DMZ.
    - The **WAF** will filters and protects these requests before they reach the API Gateway.
    - The **reverse proxy** can forward requests to the appropriate internal services if needed.
2. **DMZ to MZ:**
    - The **API Gateway** will validate and route the requests to the appropriate services in the MZ.
    - **Communication between the DMZ and MZ** will be controlled and monitored to ensure security.
    - **Internal firewalls** and access **control lists** (ACLs) will be used to restrict and monitor traffic between the DMZ and MZ.
3. **Internal Communication:**

- Services within the MZ communicate with each other securely, using internal firewalls and access control mechanisms.
- Data is **encrypted in transit** between services to ensure security.
- **Monitoring and alerting** tools track the health and performance of internal services and databases.

3. Divergent is expected to confirm if Internet service can only be provided in DMZ Services and If internet service would be required through proxy or direct internet.

- Yes, **confirmed**. The internet connectivity will be required in the DMZ to send SMS through the operator's SMSC API.

4. Is Divergent's native load balancer included in the solution & also F5 compatibility if required?

➢ **Load Balancer:**

Yes, Divergent may provide software-based load balancer given the required resources.

Based on the preference from Nagad Limited, Divergent will implement on of the following:

- **HAProxy:** A reliable, high-performance TCP/HTTP load balancer.
- **NGINX:** Can be configured as a reverse proxy and load balancer, providing additional features like caching and SSL termination.

➢ **F5 Compatibility:**

Analysis of Your Proposed Solution for F5 Compatibility:

1. **Native Load Balancer:**
   - **Current Setup:** Solution will include a native load balancer (e.g., HAProxy, NGINX) to distribute incoming traffic.
   - **F5 Compatibility:** The native load balancer can work in conjunction with F5 load balancers. F5 can act as an advanced load balancer, providing additional features and security. Native load balancer can forward traffic to F5 for advanced processing.
2. **Security:**
   - **Current Setup:** Solution will include a Web Application Firewall (WAF) and reverse proxy for security.
   - **F5 Compatibility:** F5's WAF and security features can enhance existing security setup. Our solution will be capable to integrate with F5's security features for comprehensive protection.
3. **Scalability:**

- **Current Setup:** Solution is designed to scale horizontally and vertically using Kubernetes and other scalability features.
- **F5 Compatibility:** F5 load balancers support dynamic scaling, which aligns with the scalability requirements. Our solution will scale seamlessly with F5.

4. **Monitoring and Analytics:**
    - **Current Setup:** Solution includes monitoring and alerting services using tools like Prometheus and Grafana.
    - **F5 Compatibility:** F5's monitoring and analytics tools can complement your existing monitoring setup. Our solution will integrate with F5's monitoring tools for real-time insights.
5. **Integration and Management:**
    - **Current Setup:** Solution is designed to be managed and configured using APIs and management tools.
    - **F5 Compatibility:** F5 provides APIs and management interfaces for dynamic configuration. Our solution can be managed and configured using F5's APIs and management tools.

5. How will web service vulnerability be mitigated?

Mitigating web service vulnerabilities is crucial for ensuring the security and integrity of your SMS Gateway solution. Here are several strategies and best practices to mitigate common web service vulnerabilities:

1. Input Validation and Sanitization

Purpose: To prevent injection attacks (e.g., SQL injection, command injection) by ensuring that all user inputs are validated and sanitized.

Implementation:

- **Validation:** Implement strict validation rules for all user inputs to ensure they conform to expected formats and values.
- **Sanitization:** Sanitize user inputs to remove or escape potentially dangerous characters and sequences.
- **Libraries and Frameworks:** Use libraries and frameworks that provide built-in validation and sanitization features (e.g., OWASP ESAPI, parameterized queries).

2. Authentication and Authorization

Purpose: To ensure that only authorized users and services can access specific resources and perform certain actions.

Implementation:

- **Authentication:** Implement robust authentication mechanisms (e.g., OAuth 2.0, API keys) to verify the identity of users and services.
- **Authorization:** Use role-based access control (RBAC) to enforce access permissions based on user roles and permissions.
- **Multi-Factor Authentication (MFA):** Implement MFA to add an additional layer of security for critical operations.

3. Secure Communication

Purpose: Protect data in transit from eavesdropping and man-in-the-middle attacks.

Implementation:

- **SSL/TLS:** Enforce the use of SSL/TLS for all communications between clients and servers, as well as between internal services.
- **Certificate Management:** Use trusted certificate authorities (CAs) for issuing and managing SSL/TLS certificates.
- **HTTP Strict Transport Security (HSTS):** Implement HSTS to ensure that browsers only communicate with your services over HTTPS.

4. Web Application Firewall (WAF)

Purpose: Protect against common web application attacks by filtering and monitoring HTTP/HTTPS traffic.

Implementation:

- **WAF Deployment:** Deploy a WAF in the DMZ to inspect and filter incoming traffic for malicious patterns and behaviors.
- **Configuration:** Configure the WAF to detect and block common attacks such as SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF).
- **Monitoring:** Continuously monitor WAF logs and alerts to identify and respond to potential threats.

5. Rate Limiting and Throttling

Purpose: Prevent denial-of-service (DoS) attacks and abuse by limiting the number of requests a client can make in a given time period.

Implementation:

- **Rate Limiting:** Implement rate limiting to restrict the number of requests a client can make to your APIs within a specific time frame.
- **Throttling:** Use throttling to temporarily block or slow down clients that exceed rate limits.
- **API Gateway:** Configure rate limiting and throttling rules in your API gateway to protect against DoS attacks.

6. Security Headers

Purpose: Enhance the security of web applications by configuring security-related HTTP headers.

Implementation:

- **Content Security Policy (CSP):** Implement CSP to prevent XSS attacks by specifying which sources of content are trusted.
- **X-Content-Type-Options:** Use this header to prevent MIME-sniffing vulnerabilities.
- **X-Frame-Options:** Use this header to prevent clickjacking attacks by controlling whether a browser should be allowed to render a page in a frame.
- **X-XSS-Protection:** Enable the XSS filter built into most web browsers to protect against reflected XSS attacks.

7. Regular Security Audits and Penetration Testing

Purpose: Identify and mitigate vulnerabilities in your web services through regular security assessments.

Implementation:

- **Security Audits:** Conduct regular security audits to identify and address vulnerabilities in your web services.
- **Penetration Testing:** Perform penetration testing to simulate real-world attacks and identify potential security weaknesses.
- **Vulnerability Management:** Use vulnerability management tools to continuously scan for and address vulnerabilities in your web services.

8. Patch Management

Purpose: Ensure that all software components are up-to-date and protected against known vulnerabilities.

Implementation:

- **Regular Updates:** Implement a patch management process to ensure that all software components, including web servers, application servers, and libraries, are regularly updated with the latest security patches.
- **Automated Patching:** Use automated patching tools to streamline the patch management process and ensure timely updates.

9. Logging and Monitoring

Purpose: Detect and respond to security incidents by monitoring and logging web service activities.

Implementation:

- **Comprehensive Logging:** Implement comprehensive logging to capture all relevant security events and activities.
- **Centralized Logging:** Use a centralized logging solution to aggregate and analyze logs from all web services.

- **Real-Time Monitoring:** Implement real-time monitoring to detect and respond to security incidents promptly.
- **Security Information and Event Management (SIEM):** Use a SIEM system to correlate and analyze security events from multiple sources.

10. Secure Coding Practices

Purpose: Prevent vulnerabilities by following secure coding practices during development.

Implementation:

- **Code Reviews:** Conduct regular code reviews to identify and address potential security vulnerabilities.
- **Static Application Security Testing (SAST):** Use SAST tools to analyze source code for security vulnerabilities.
- **Dynamic Application Security Testing (DAST):** Use DAST tools to test running applications for security vulnerabilities.
- **Security Training:** Provide regular security training for developers to ensure they are aware of common vulnerabilities and best practices for secure coding.

6. Does the solution support SSL/TLS communication between client & servers?

Yes, the SMS Gateway solution should support SSL/TLS communication between clients and servers to ensure secure data transmission. This encryption is essential for protecting sensitive information from eavesdropping, tampering, and other security threats.

7. All third-party connectivity needs to connect from/to DMZ services only. No core service can be directly exposed to external parties. – Does Divergent comply with this requirement?

Yes, Divergent Technologies Ltd, will comply with the requirement that all third-party connectivity should connect from/to DMZ services only, and no core service should be directly exposed to external parties.