Functional Requirements
sendMessage(messageBody)
receiveMessage()

Non-Functional Requirements
Scalable (Handle load with increasing queues and messages)
Highly Available (Survives hardware & network failure)
Highly Performant (single digit latency for operations)
Durable (data persisted when submitted to queue until polled)
SLA (minimum throughput)

Design

Design Patterns

- Bulkhead Pattern
- Circuit Breaker Pattern

Components

- **FrontEnd WebService**
    - **Lightweight web service**
    - **Stateless service deployed across data centers**
    - **Actions**
        - Request validation
            - Required params are present (e.g. Queue name)
            - Message Size not exceeding threshold value
        - Authentication / Authorization
        - TLS/SSL termination
            - SSL termination on load balancer is expensive
            - Usually handled by a separate TLS HTTP proxy runs as a process on same host
        - Caching (Server Side)
            - Cache stores copies of source data
            - Stores info about most frequently used queue to save cost on authentication and authorization service
            - It helps to reduce load to Backend services, increases overall system throughput & availability, decreases latency
            - Stores previously seen requestIds
        - Rate Limiting (Throttling)
        - Request Dispatching
            - Responsible for all activities associated with sending request to Backend service like client management, response handling, resource isolation etc.
        - User data collection
        - Server Side encryption
            - Messages are encrypted as soon as received

- Messages are stored in encrypted format and FrontEnd decrypt message only when they are sent back to client

- Request Deduplication
  - May occur when a response from a successful sendMessage(messageBody) failed to reach client
- Delivery Semantics
  - Lesser an issue for ‘at least once’, a bigger issue for ‘exactly once’ & ‘at most once’
  - Caching is usually done to store previously seen requestIds

- **Metadata Service**
  - Stores information about queues
  - Many reads, little writes
  - Strong consistency storage is preferred but not required
  - Sharded to handle load
- **BackEnd Service**
  - Data stored in RAM and disk of backend host
  - FrontEnd Service retrieves BackEnd Service host information from Metadata Service
  - Replicated with in groups of host
    a) Leader-Follower relationship
      - Incluster manager
        (QueueId, Leader Host, Followers)
        - Manages queue assignment within cluster
        - Maintain a list of host in the cluster
        - Monitors heartbeat from hosts
        - Deals with leader & follower failures
        - Splits queue between cluster nodes

    b) Small clusters of independent hosts
      - Outcluster Manager
        (QueueId, ClusterId)
        - Manages queue assignment among clusters
        - Maintain a list of clusters
        - Monitors each cluster health
        - Deals with overrated cluster
        - Splits queue between clusters

Queue creation & deletion
Message deletion and mark invisible like SQS
Message Replication Synchronous & Asynchronous
Message delivery semantics atleast once is hard
Push vs Pull vs Long Polling
FIFO
Security (SSL over HTTP, encrypt in backend service)
Monitoring - FE, BE, Metadata Service, Log Data, Alerts/Alarms, Customer Dashboard