Distributed Pub/Sub Architecture Proposal

Current Limitations

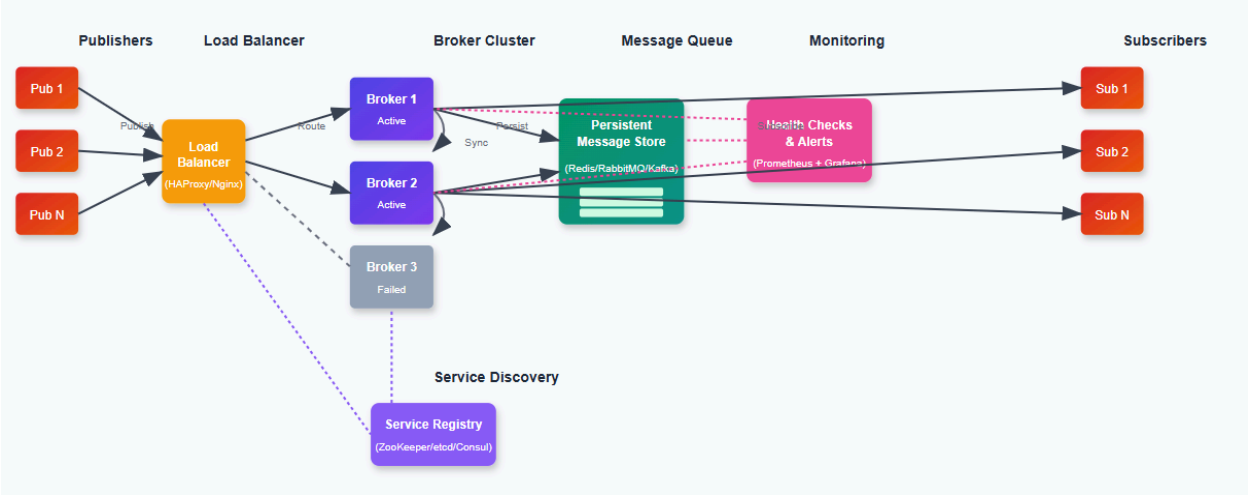
The existing implementation has several critical limitations:

1. Single point of failure: The entire system fails if the server crashes
2. Limited scalability: All traffic flows through one server
3. No message persistence: Messages are lost during server failures
4. No fault tolerance: No recovery mechanism exists

Proposed Architecture

Components:

1. Broker Cluster: Multiple broker nodes that handle client connections
2. Load Balancer: Distributes client connections evenly
3. Message Queue: Persistent storage for messages
4. Service Discovery: Tracks available brokers
5. Monitoring: Health checks and alerts



How It Works

1. Publishers connect via the load balancer
2. The load balancer routes them to an available broker
3. Brokers persist messages to the shared queue
4. Subscribers receive messages from their connected broker
5. Brokers replicate messages among themselves

Failure Handling

- If a broker fails:
  - Load balancer stops routing to it
  - Service discovery updates available brokers
  - Subscribers reconnect to available brokers
  - Messages are not lost (persisted in queue)

Implementation Options

- Brokers: Custom implementation or Kafka brokers
- Load Balancer: Nginx, HAProxy
- Message Queue: Redis, RabbitMQ, Kafka
- Service Discovery: ZooKeeper, etcd, Consul
- Monitoring: Prometheus + Grafana

Tradeoffs

- Increased complexity vs improved reliability
- Eventual consistency vs strong consistency
- Resource requirements vs performance