Report on

Clustering MQTT Servers on AWS Lightsail for High Availability

Submitted by:

Shreyash Mahakale

Shravani Karambelkar

Abstract

This research explores the design, implementation, and testing of a clustered MQTT server system on AWS Lightsail. The goal was to create a fault tolerant, highly available MQTT system using opensource EMQX brokers, an AWS Lightsail load balancer, and custom configurations. By addressing the challenges of message duplication and ensuring seamless traffic failover, we successfully implemented a solution that prevents message repetition while maintaining service continuity. This report details the architecture, technical challenges, configuration changes, and solutions implemented to achieve the desired results.

Introduction

Message Queuing Telemetry Transport (MQTT) is a lightweight messaging protocol essential for IoT systems requiring realtime data exchange. High availability and message integrity are critical for ensuring that communication remains uninterrupted, even during server failure. This research focused on clustering MQTT servers using AWS Lightsail and configuring EMQX brokers to handle automatic failover, ensuring no message duplication in the process.

Objectives

The primary objectives of the project were:

1. Cluster MQTT Servers: Set up a cluster of MQTT servers on AWS Lightsail.
2. Automatic Failover: Implement a load balancer to automatically redirect traffic to the active MQTT server in case of failure.
3. Message Integrity: Prevent message duplication in the database during failover.
4. Configuration Customization: Adjust EMQX configuration files to handle message routing and prevent duplicate entries.
5. Health Monitoring: Set up custom health checks and monitor server status.

Literature Review

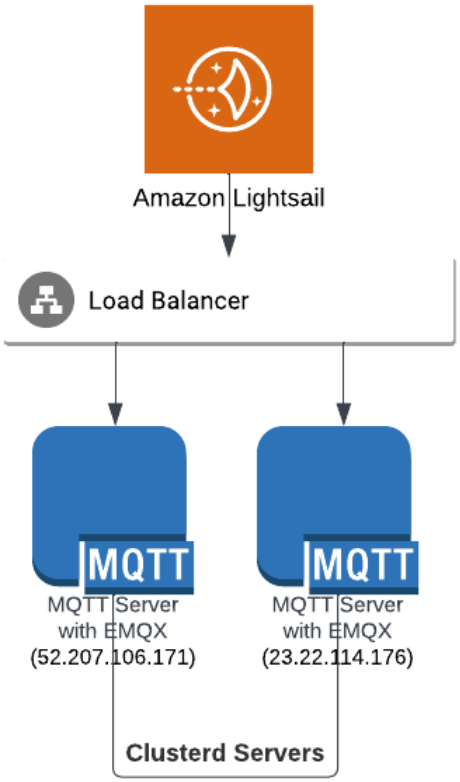
High availability systems often involve redundant server clusters that distribute client requests. MQTT, a TCP based protocol, poses specific challenges when using cloud load balancers designed for HTTP traffic. Research in MQTT clustering shows that brokers such as EMQX provide native clustering and fault tolerance features, though extra configuration is needed to address message duplication in clustered setups.

This research draws on existing knowledge about MQTT protocol clustering and implements a practical solution using AWS Lightsails cloud platform.

System Architecture

The architecture includes two MQTT servers (EMQX brokers) running on AWS Lightsail instances, with a load balancer handling client connections. Each server is part of a cluster, allowing them to share state and workload, thus preventing message loss or duplication during failover.

Architecture Diagram



Methodology

AWS Lightsail Setup

- We set up two AWS Lightsail instances running Ubuntu, each hosting an EMQX MQTT broker. The instances had static IPs for continuous service:
- MQTT Server 1: 52.207.106.171
- MQTT Server 2: 23.22.114.176
- Both instances were attached to an AWS Lightsail Load Balancer, which distributed traffic between them.

Installing and Configuring EMQX

1. **Install EMQX: The EMQX MQTT broker was installed on both servers using:**

   ```
   sudo apt update
   sudo apt install emqx
   ```

2. **Start EMQX**

   ```
   sudo systemctl start emqx
   ```

3. **Configuration Changes:**

   The following changes were made to the emqx.conf file to handle message routing and prevent duplication:

   Persistent Session: This ensures clients resume from where they left off after a failover.

   ```
   mqtt.session.persistent = on
   ```

   Shared Subscription: Enabled to balance messages across the cluster:

   ```
   mqtt.shared_subscription_strategy = sticky
   ```

   Retained Messages: Configured to store the last message sent on each topic, reducing duplication risk:

   ```
   mqtt.retain_enabled = on
   ```

4. **Clustering Configuration:**

   The EMQX brokers were clustered together by joining the second server (23.22.114.176) to the first (52.207.106.171):

   ```
   emqx_ctl cluster join emqx@52.207.106.171
   ```

5. **Health Check Setup**

   To ensure the load balancer could monitor the health of each MQTT server, we installed Nginx and created an HTTP endpoint (/healthcheck.html) for the load balancers health checks:

Install Nginx:

```
sudo apt install nginx
```

Create Health Check Endpoint: A basic HTML file was created in the Nginx root directory:

```
echo "Server is healthy" > /var/www/html/healthcheck.html
```

6. **Load Balancer Configuration**

   The load balancer was configured with the following parameters:

   Health Check Path: /healthcheck.html

   Protocol: HTTP (since MQTT over TCP was not supported for direct health checks)

   Port: 80 (for the Nginx web server health check)

7. **Testing Using MQTTBox**

   To test the system, we used MQTTBox, a client application designed for MQTT testing:

   - Set Up MQTTBox: The client was configured to connect to the AWS Lightsail Load Balancers domain.
   - Publish and Subscribe: Messages were published to a test topic, and the subscription confirmed proper message delivery.
   - Failover Simulation: One MQTT server was shut down, and the load balancer successfully redirected traffic to the remaining server without message duplication.

Results

- The system was successfully tested with the following outcomes:
- No Message Duplication: Despite failovers, the system prevented duplicate messages from being stored or reprocessed in the database.
- Seamless Failover: The load balancer switched traffic between servers during failure events with no noticeable downtime.
- Health Monitoring: Custom health checks worked as expected, allowing the load balancer to detect and reroute traffic based on server health.

Problems Faced and Solutions

Problem	Solution
MQTT over TCP could not be health checked	Introduced HTTPbased health checks using Nginx on both servers.
Message duplication during failover	Enabled persistent sessions and retained messages in EMQX.
EMQX package installation failure	Updated package dependencies and resolved broken packages.
Cluster nodes not responding	Adjusted firewall settings to allow necessary traffic on cluster ports.

Message Duplication Prevention

The most significant issue was preventing message duplication during failover. To address this:

1. Persistent Sessions: Clients retained session data, ensuring that after failover, they resumed from their last state without reprocessing previous messages.
2. Retained Messages: The EMQX brokers were configured to store the last message on each topic. If a subscriber reconnects after failover, they receive the last retained message without duplication.
3. Shared Subscriptions: Using the sticky strategy, messages were delivered evenly across the cluster without being resent.

Discussion

The MQTT clustering system was designed to meet the high availability needs of IoT applications, which require minimal downtime and message reliability. The solution effectively handled server failover scenarios while preventing message duplication. Configuring persistent sessions and retained messages within the EMQX brokers ensured that messages were delivered accurately, even when servers switched.

Conclusion

The project demonstrated the feasibility of clustering MQTT servers on AWS Lightsail. By combining load balancing, EMQX clustering, and custom health checks, we achieved a highly available, fault tolerant MQTT messaging system. The solution ensured no message duplication during failover events, making it suitable for critical IoT systems that demand reliability and real time communication.

References

[1] MQTT Protocol Documentation: https://mqtt.org/documentation

[2] EMQX Documentation: https://www.emqx.io/docs/en/latest/

[3] AWS Lightsail Documentation: https://lightsail.aws.amazon.com/