Creation Of Pub-Sub Model Based Web Application Using SNS

Abstract:

Here in this project, we are creating a web application which runs on top of pub-sub model using AWS service. AWS offers a wealth of options for implementing messaging patterns such as pub-sub with Lambda. Publish – Subscribe often shortened to pub-sub is a messaging pattern where publishers and subscribers are decoupled through intermediary broker (ZeroMQ, RabbitMQ, SNS, etc…). In the AWS ecosystem, the obvious candidate for the broker role is SNS and SNS will make 3 attempts for your function to process a message before sending it to a Dead Letter Queue (DLQ) if a DLQ is specified for the function. However, we can increase it to 6 number of retries.

Another thing to consider is the degree of parallelism this architecture offers. For each message SNS will create a new innovation of the function. So, if we publish 100 messages to SNS then we can have 100 concurrent executions of the subscribed Lambda function. This is just great if we are optimizing for throughput.

However, we are often constrained by the max throughput our downstream dependencies can handle the databases, S3, internal/external services etc. If the burst in throughput is short then there is a good chance the retries would be sufficient and we won't miss any messages.

Our web application running on AWS EC2 with AWS SNS Pub-Sub Model, it can deliver the content, and messages to all the subscribers.

Existing System:

In the existing system, many companies are using HTTP protocol for replicating the same usage as us and in such scenarios the HTTP is heavy weight and mis-fire of requests will lost the messages and human intervention will be needed to recover the messages in those cases.

That is why, we are proposing a web app to be built on top of pub-sub model.

Proposed System:

In modern cloud architecture, applications are decoupled into smaller, independent building blocks that are easier to develop, deploy and maintain. Publish/Subscribe message provides instant event notifications for these distributed applications.

The publish-subscribe model enables event-driven architectures and asynchronous parallel processing, while improving performance, reliability and scalability.

Our Web App is divided into small micro-services and these micro-services communicate each other using pub-sub models.

Software Tools:

1. AWS Cognito
2. AWS SNS
3. AWS EC2
4. AWS S3
5. AWS DynamoDB
6. AWS Lambda
7. AWS CLI
8. AWS CloudWatch
9. AWS IAM
10. VS Code
11. Python3

Hardware Tools:

1. Laptop
2. Operating System: Windows 11
3. RAM: 16GB
4. ROM: 4GB
5. Fast Internet Connectivity