AWS SERVERLESS MESSAGING USING SQS

Ashutosh Tripathi
Engineering Manager,
Clara Analytics, USA
ashu.goldi@gmail.com

Publication History
Research Article | Open Access
Peer-review: Double-blind Peer-reviewed
Article ID: IJIRAE/RS/Vol.07/Issue11/NVAE10082
Received: 04, November 2020
Accepted: 17, November 2020
Published Online: 22, November 2020
Volume 2020 | Article ID NVAE10082 | https://doi.org/10.26562/ijirae.2020.v0711.003
Ashutosh (2020). "AWS Serverless Messaging using SQS". IJIRAE:: International Journal of Innovative Research in Advanced Engineering, Volume VII, 391-393.
doi: https://doi.org/10.26562/ijirae.2020.v0711.003
Editor-Chief: Dr.A.Arul Lawrence Selvakumar, Chief Editor, IJIRAE, AM Publications, India

INTRODUCTION

In the dynamic and rapidly evolving realm of cloud computing, messaging stands as a pivotal element in constructing resilient and high-performing serverless applications. AWS offers a range of services facilitating seamless communication among various application components, including AWS Lambda, Simple Queue Service (SQS), and event-driven architectures. Within this detailed guide, we will delve into the diverse facets of messaging within AWS serverless environments, showcasing its capacity to empower developers in crafting scalable and robust applications.

UNDERSTANDING AWS LAMBDA AND ITS INTEGRATION WITH SQS

AWS Lambda stands as a serverless computing solution that eliminates the need for server provisioning or management, offering an optimal platform for constructing event-driven architectures. These architectures rely on functions triggered by events, responding with specified actions. A critical element within such architectures is message queues, with AWS providing SQS a fully managed message queuing service. SQS facilitates component decoupling by enabling one component to dispatch a message to another for asynchronous processing. Integrating AWS Lambda with SQS involves allocating a distinct queue for each application stack. These queues trigger handler methods to process incoming messages. Messages can be delivered to the application's SQS queue by any AWS service. Upon successful processing, messages are removed from the queue, ensuring dependable and scalable message processing in a decoupled manner.

Advantages:

Scalability and Elasticity: Notification traffic tends to vary significantly, experiencing surges during events or specific times of the day. To effectively manage these fluctuations, a serverless architecture offers the scalability and elasticity needed for seamless handling of peaks. Explore how AWS Lambda and similar serverless platforms can be leveraged for notification logic.

Cost-Efficiency: Traditional server-based architectures necessitate provisioning servers based on peak anticipated loads, often leading to underutilization of resources and increased operational expenses. Conversely, serverless architectures embrace a pay-as-you-go model, where charges are incurred solely for actual usage. This cost-effective paradigm optimizes resource utilization, mitigating superfluous expenses and fostering operational efficiency. By dynamically scaling resources in response to demand, serverless architectures ensure optimal performance while economizing on infrastructure costs, thus empowering organizations to allocate resources judiciously and achieve greater financial prudence.

Operational Overhead Reduction: The tasks associated with managing servers, implementing patches, and handling infrastructure maintenance can be time-intensive. In a serverless architecture, these responsibilities are offloaded to the cloud provider, leading to a substantial reduction in operational overhead.

Defining the Queue and Mapping to Lambda

To set up the integration between SQS and AWS Lambda, you need to define the queue and map it to the Lambda function. This can be done using a YAML configuration file. Here is an example of how the queue and mapping can be defined:

```
TestApplicationQueue:
Type:AWS::SQS::Queue
Properties:
QueueName:!Sub'TestApplicationQueue'
DelaySeconds:0
VisibilityTimeout:900
TestQueueToLambdaEventMapping:
Type:AWS::Lambda::EventSourceMapping
Properties:
BatchSize:1
Enabled:true
EventSourceArn:!GetAttTestApplicationQueue.Arn
FunctionName:!GetAttSQSEventHandler.Arn
```

This configuration sets up a queue named Test Application Queue and maps it to the Lambda function SQS Event Handler. The Batch Size property determines the number of messages to be processed in a single batch, and the Visibility Time out property specifies the amount of time a message remains invisible after being read by a consumer. Message is supposed to be consumed and deleted during the visibility timeout.

Data Structure of the Message

The messages placed in the SQS queue have a specific data structure. The data structure is defined using a class or DTO (Data Transfer Object). Below is an example of a data structure for the message. Desired fields can be added to the DTO class.

```
publicclassMessageBaseDTO{
publicStringname;
publicbooleanisAsync;
publicStringuserName;
publicStringuserEmail;
publicStringtaskPerformed;
// extra fields per requirement
}
```

The Message BaseD TO class represents the structure of the message. It contains fields such as name, is Async, user Name, user Email, task Performed and task Performed. The task Performed is used to determine which type of service should process the message. For example, if the task Performed is set to EMAIL_CUSTOMERS, the message body will be processed by the Email Customer Action class. We can have multiple Action files defined for various actions. i.e. DELETE_USER value of task Performed can trigger Delete Customer Action; BUY_PRODUCT value of task Performed can trigger Buy Product Action.

Registering Services for SQS

To register a service for processing messages from the SQS queue, you need to define the task Performed key and associate it with the corresponding service class instance. This can be done in the application's module file using the Notification Util. Here is an example of how to register a service for SQS:

```
static{
NotificationUtil.getNotificationRegistrationInstance()
registerService("EMAIL_CUSTOMERS",newEmailCustomerAction());
NotificationUtil.getNotificationRegistrationInstance()
registerService("DELETE_USER",newDeleteCustomerAction());
}
```

In the above example, the EMAIL_CUSTOMERS task Performed key is registered with the Email Customer Action class. This ensures that messages with the EMAIL_CUSTOMERS task Performed key are processed by the Email Customer Action class.

Implementing the SQS Service Class

To process the request body sent to SQS, you need to create a service class that extends the Notification Base Action class and implements the process method. This method is responsible for handling the incoming message and performing the required actions. Here is an example of how to write an SQS service class (AWS libraries included):

```
publicabstractclassNotificationBaseAction{
publicabstractvoidprocess(Object input,Context context);
}
```

```
publicclassEmailCustomerActionimplementsNotificationBaseAction{
Gsongson=newGsonBuilder().serializeNulls().create();
@Override
publicvoidprocess(Object input,Context context){
SQSResponsesqsResponse=(SQSResponse)ServerlessContext.getContextMap().get(ServerlessConstants.SQS_RES
PONSE);
EmailNotificationDTOemailDto=gson.fromJson(sqsResponse.getBody(),EmailNotificationDTO.class);
System.out.println(emailDto.toString());
}
}
```

In the above example, the Email Notification Action class processes the message body by deserializing it into an Email Notification DTO object using Gson. The process method can then perform any required actions based on the content of the message.

Deleting Messages after Consumption

After a message undergoes consumption and processing by an SQS service class, it becomes imperative to promptly remove it from the queue. This action is essential to prevent the message from being processed anew, thereby averting redundancy, or being rerouted to the Dead Letter Queue due to repeated processing failures. To streamline this process and ensure seamless message management, developers can devise an interceptor mechanism. This interceptor, strategically positioned at the exit of Lambda calls, automatically triggers the deletion of messages from the queue post-task completion. By implementing this interceptor, developers can streamline message handling, enhance efficiency, and fortify the reliability of their serverless applications.

CONCLUSION

Messaging represents a crucial element in constructing scalable and robust serverless applications within AWS. By harnessing tools such as AWS Lambda and SQS, developers can establish decoupled architectures and handle message processing asynchronously. This guide delves into the intricacies of integrating AWS Lambda with SQS, covering aspects such as queue definition, message structure, and the implementation of service classes for message handling. Emphasized is the necessity of removing consumed messages to avert duplicate processing. Armed with this understanding, you can adeptly devise and deploy messaging solutions within your AWS serverless applications. However, the integration of AWS Lambda with SQS merely scratches the surface of messaging options in AWS serverless environments. Additional services like Amazon SNS (Simple Notification Service) and Amazon EventBridge offer equally robust messaging capabilities. Selecting the most appropriate messaging service hinges on aligning with your application's specific requirements and use cases.

REFERENCE

1. Aws-lambda-sqs-lambda https://docs.aws.amazon.com/solutions/latest/constructs/aws-lambda-sqs-lambda.html
2. Pogiatzis, A.; Samakovitis, G. An Event-Driven Serverless ETL Pipeline on AWS. Appl. Sci. 2021, 11, 191. https://doi.org/10.3390/app11010191
3. Patterson, Scott. Learn AWS Serverless Computing: A Beginner's Guide to Using AWS Lambda, Amazon API Gateway, and Services from Amazon Web Services. Packt Publishing Ltd, 2019.
4. Sbarski, P., Kroonenburg, S. (2017). Serverless Architectures on AWS: With Examples Using AWS Lambda. (n.p.): Manning.