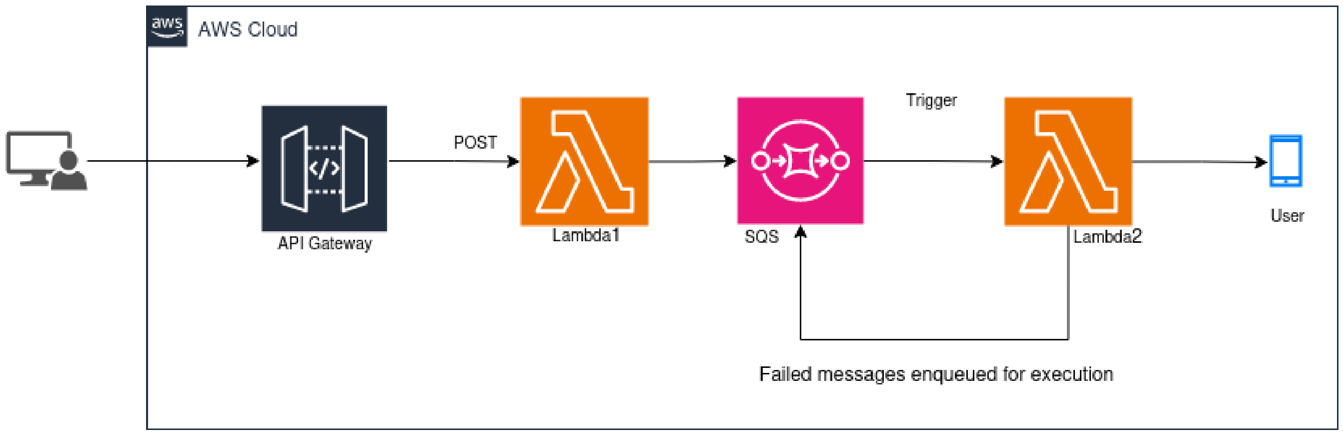
Notification System

Prerequisites:

1. API Gateway
2. IAM Permissions
3. Lambda Functions
4. Simple Queue Service (SQS)

Below is the architecture of the robust notification system:

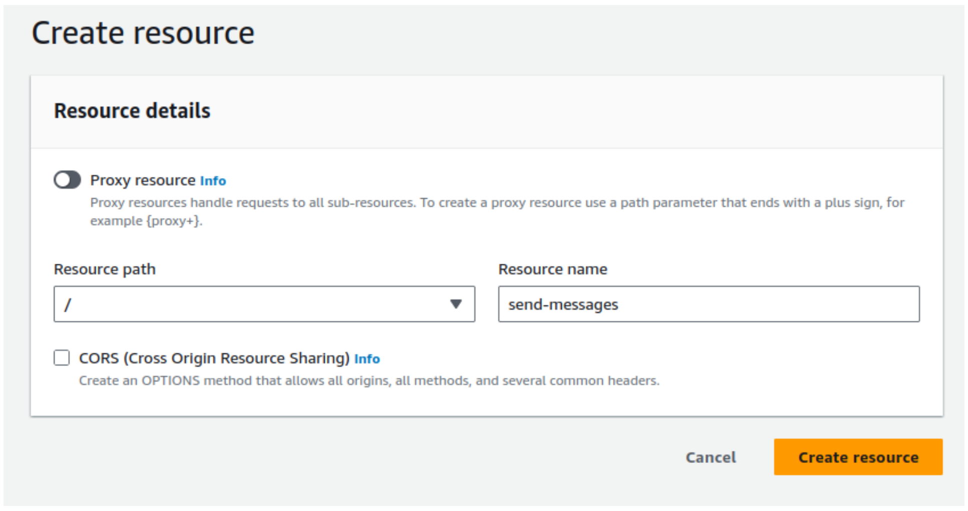


Calculator Link: (10000 messages/month): [Notification System]()

Creation of resources are:

A. **API Gateway**:

1. Create the *REST API* with *API endpoint* type *Regional*
2. *Create resource* and click on *Create method* and choose method type as *POST* and Integrate type as *Lambda1*



Create resource

Resource details

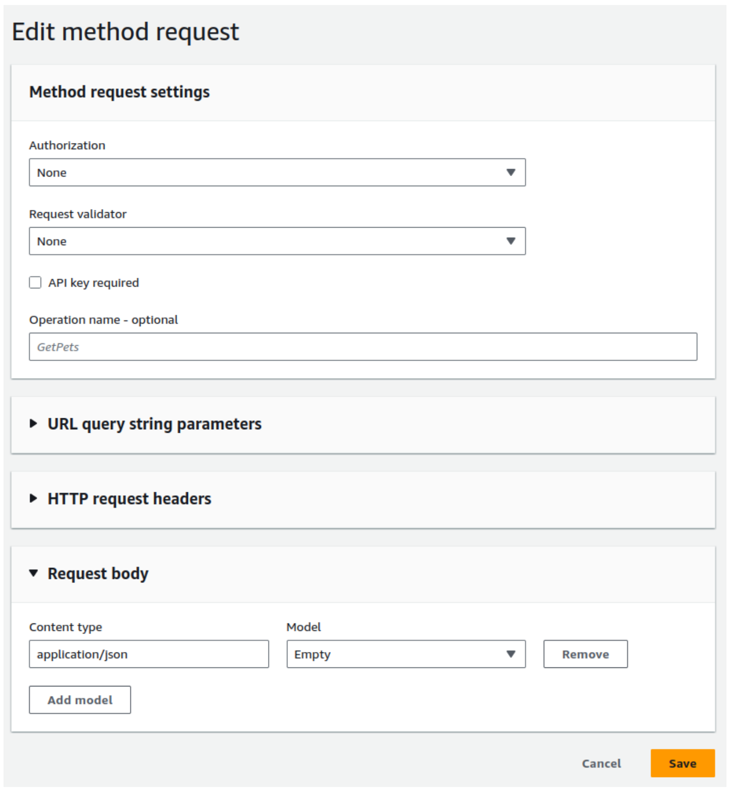
Proxy resource Info
Proxy resources handle requests to all sub-resources. To create a proxy resource use a path parameter that ends with a plus sign, for example {proxy+}.

Resource path: /

Resource name: send-messages

CORS (Cross Origin Resource Sharing) Info
Create an OPTIONS method that allows all origins, all methods, and several common headers.

a. Add the *Request body* in Method request as *application/json*:



Edit method request

Method request settings

Authorization: None

Request validator: None

☐ API key required

Operation name - optional: GetPets

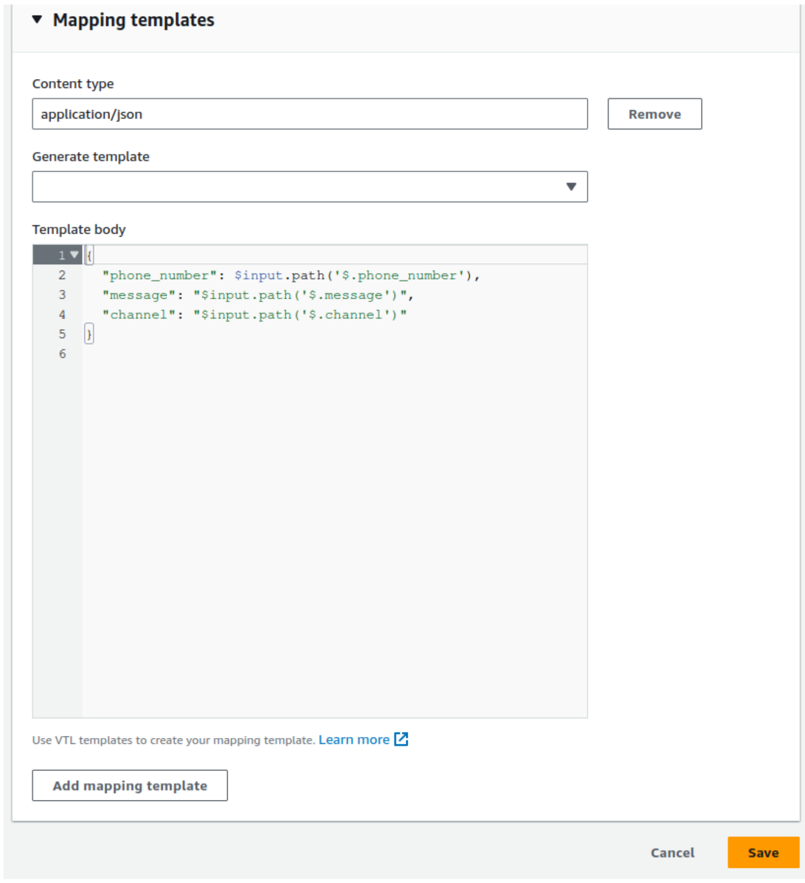
▸ **URL query string parameters**

▸ **HTTP request headers**

▾ **Request body**

Content type	Model
application/json	Empty

b. Add below *template body* in *Mapping templates* in *Integration request*:



▾ **Mapping templates**

Content type: application/json

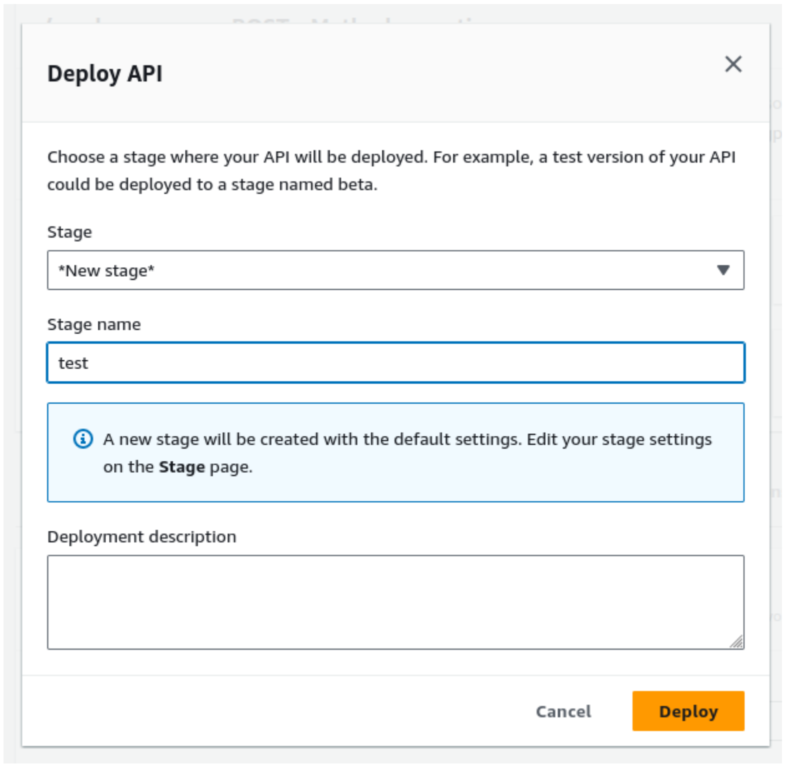
Generate template:

Template body

```
{
  "phone_number": $input.path('$.phone_number'),
  "message": "$input.path('$.message')",
  "channel": "$input.path('$.channel')"
}

```

Use VTL templates to create your mapping template. Learn more

c. *Deploy API* in the API by creating *new* *stage*:



Deploy API

Choose a stage where your API will be deployed. For example, a test version of your API could be deployed to a stage named beta.

Stage: *New stage*

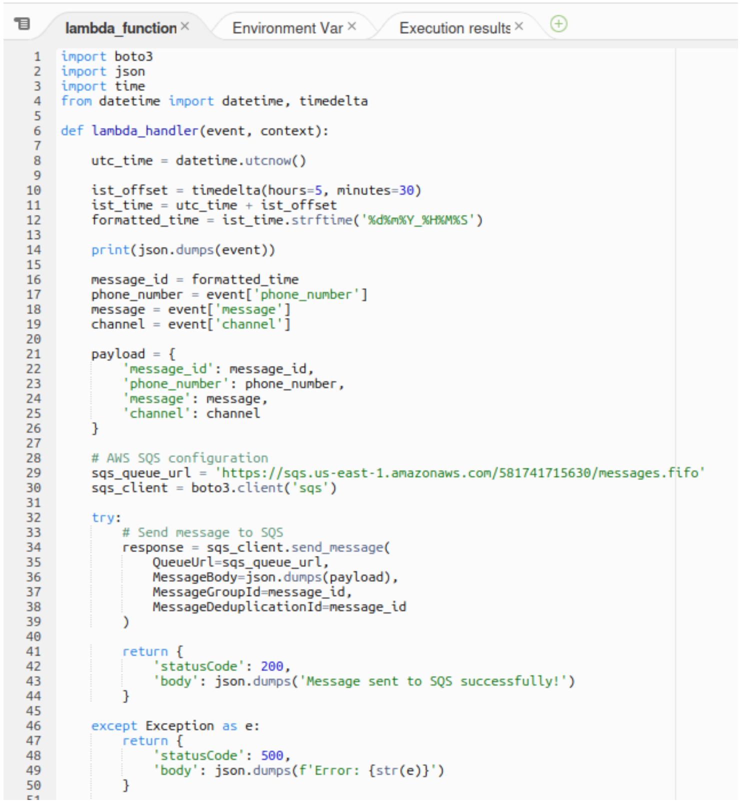
Stage name: test

A new stage will be created with the default settings. Edit your stage settings on the **Stage** page.

Deployment description:

B. **Lambda Function (**Passes json payload to the SQS**)**:

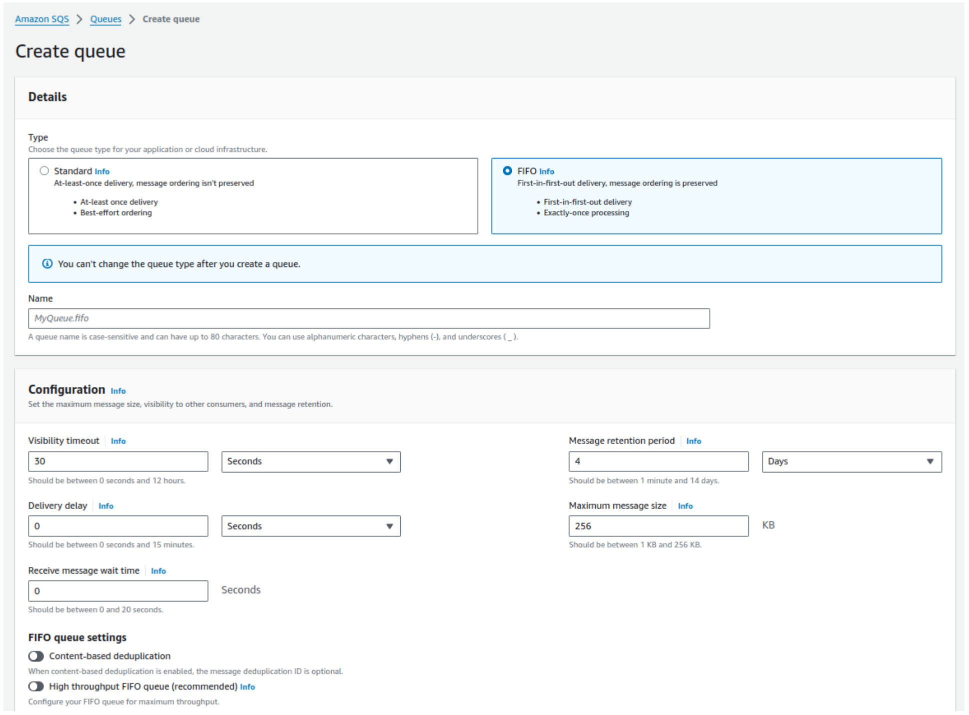
1. Create the Lambda Function
2. Add permissions *AmazonSQSFullAccess* and *AWSLambdaBasicExecutionRole* to the role and attach the role to the lambda function
3. Add the *lambda1_code* in the Lambda function



```python
import boto3
import json
import time
from datetime import datetime, timedelta

def lambda_handler(event, context):

    utc_time = datetime.utcnow()

    ist_offset = timedelta(hours=5, minutes=30)
    ist_time = utc_time + ist_offset
    formatted_time = ist_time.strftime('%d%m%Y_%H%M%S')

    print(json.dumps(event))

    message_id = formatted_time
    phone_number = event['phone_number']
    message = event['message']
    channel = event['channel']

    payload = {
        'message_id': message_id,
        'phone_number': phone_number,
        'message': message,
        'channel': channel
    }

    # AWS SQS configuration
    sqs_queue_url = 'https://sqs.us-east-1.amazonaws.com/581741715630/messages.fifo'
    sqs_client = boto3.client('sqs')

    try:
        # Send message to SQS
        response = sqs_client.send_message(
            QueueUrl=sqs_queue_url,
            MessageBody=json.dumps(payload),
            MessageGroupId=message_id,
            MessageDeduplicationId=message_id
        )

        return {
            'statusCode': 200,
            'body': json.dumps('Message sent to SQS successfully!')
        }

    except Exception as e:
        return {
            'statusCode': 500,
            'body': json.dumps(f'Error: {str(e)}')
        }
```

C. **Simple Queue Service (**SQS**):**

1. Create *SQS Queue* by selecting *FIFO* and other configurations as per your requierments



Amazon SQS > Queues > Create queue

Create queue

Details

Type
Choose the queue type for your application or cloud infrastructure.

○ Standard Info
At-least-once delivery, message ordering isn't preserved
- At-least once delivery
- Best-effort ordering

● FIFO Info
First-in-first-out delivery, message ordering is preserved
- First-in-first-out delivery
- Exactly-once processing

You can't change the queue type after you create a queue.

Name
MyQueue.fifo
A queue name is case-sensitive and can have up to 80 characters. You can use alphanumeric characters, hyphens (-), and underscores (_).

Configuration Info
Set the maximum message size, visibility to other consumers, and message retention.

Visibility timeout Info
30 Seconds
Should be between 0 seconds and 12 hours.

Message retention period Info
4 Days
Should be between 1 minute and 14 days.

Delivery delay Info
0 Seconds
Should be between 0 seconds and 15 minutes.

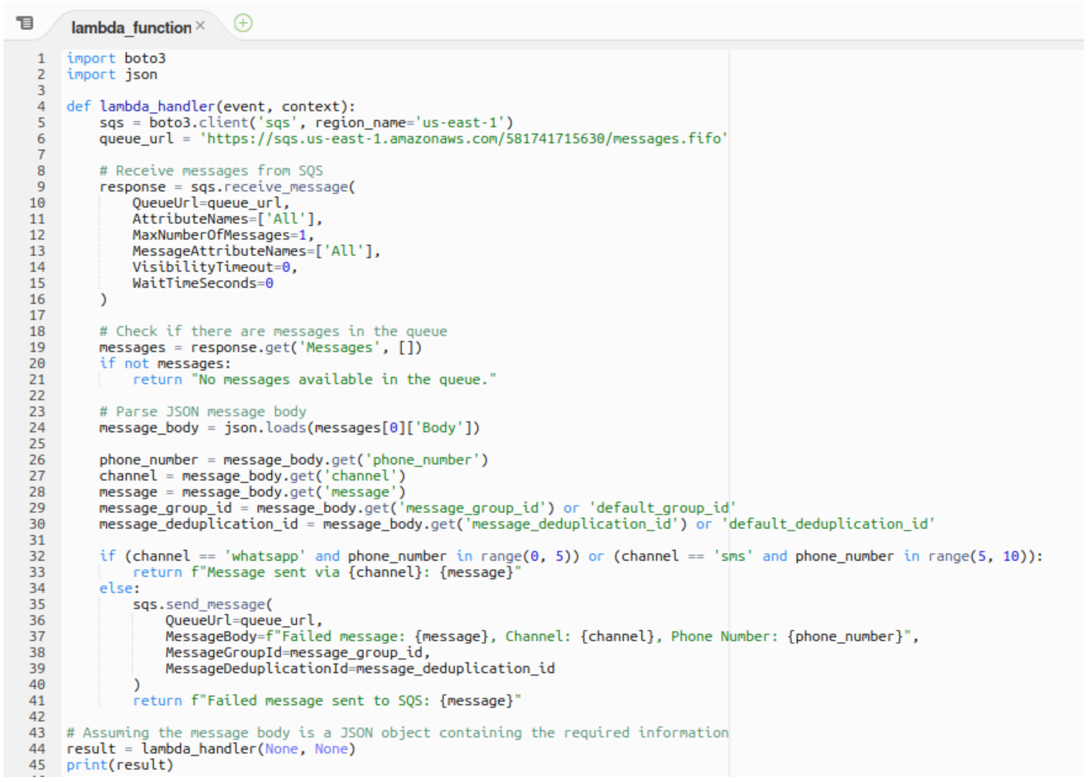
Maximum message size Info
256 KB
Should be between 1 KB and 256 KB.

Receive message wait time Info
0 Seconds
Should be between 0 and 20 seconds.

FIFO queue settings
Content-based deduplication
When content-based deduplication is enabled, the message deduplication ID is optional.
High throughput FIFO queue (recommended) Info
Configure your FIFO queue for maximum throughput.

D. **Lambda Function (**Sends the message to the user**):**

1. Create the Lambda Function which polls messages from SQS.
2. Add permissions *AmazonSQSFullAccess* and *AWSLambdaBasicExecutionRole* to the role and attach the role to the lambda function
3. Add the [lambda2_code] in the Lambda function



lambda_function

```python
import boto3
import json

def lambda_handler(event, context):
    sqs = boto3.client('sqs', region_name='us-east-1')
    queue_url = 'https://sqs.us-east-1.amazonaws.com/581741715630/messages.fifo'

    # Receive messages from SQS
    response = sqs.receive_message(
        QueueUrl=queue_url,
        AttributeNames=['All'],
        MaxNumberOfMessages=1,
        MessageAttributeNames=['All'],
        VisibilityTimeout=0,
        WaitTimeSeconds=0
    )

    # Check if there are messages in the queue
    messages = response.get('Messages', [])
    if not messages:
        return "No messages available in the queue."

    # Parse JSON message body
    message_body = json.loads(messages[0]['Body'])

    phone_number = message_body.get('phone_number')
    channel = message_body.get('channel')
    message = message_body.get('message')
    message_group_id = message_body.get('message_group_id') or 'default_group_id'
    message_deduplication_id = message_body.get('message_deduplication_id') or 'default_deduplication_id'

    if (channel == 'whatsapp' and phone_number in range(0, 5)) or (channel == 'sms' and phone_number in range(5, 10)):
        return f"Message sent via {channel}: {message}"
    else:
        sqs.send_message(
            QueueUrl=queue_url,
            MessageBody=f"Failed message: {message}, Channel: {channel}, Phone Number: {phone_number}",
            MessageGroupId=message_group_id,
            MessageDeduplicationId=message_deduplication_id
        )
        return f"Failed message sent to SQS: {message}"

# Assuming the message body is a JSON object containing the required information
result = lambda_handler(None, None)
print(result)
```

4. Failed messages will be sent back to SQS queue for re-execution.