

AWS Simple Reminder Service

Gokul Gopakumar

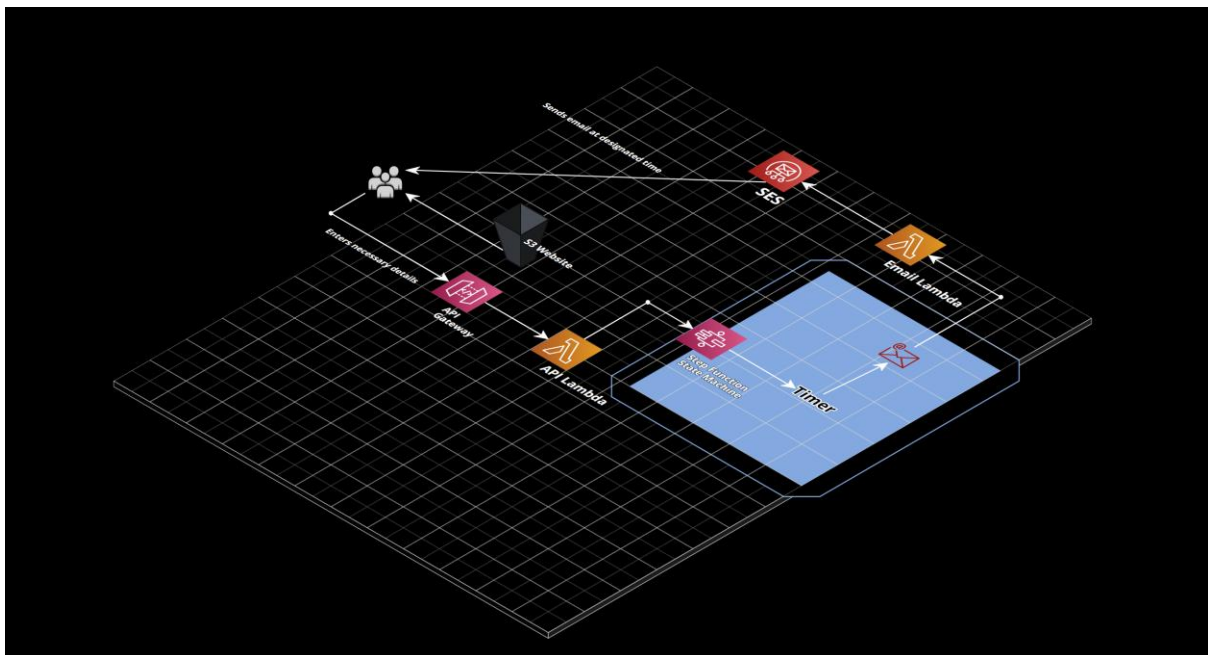
Abstract:

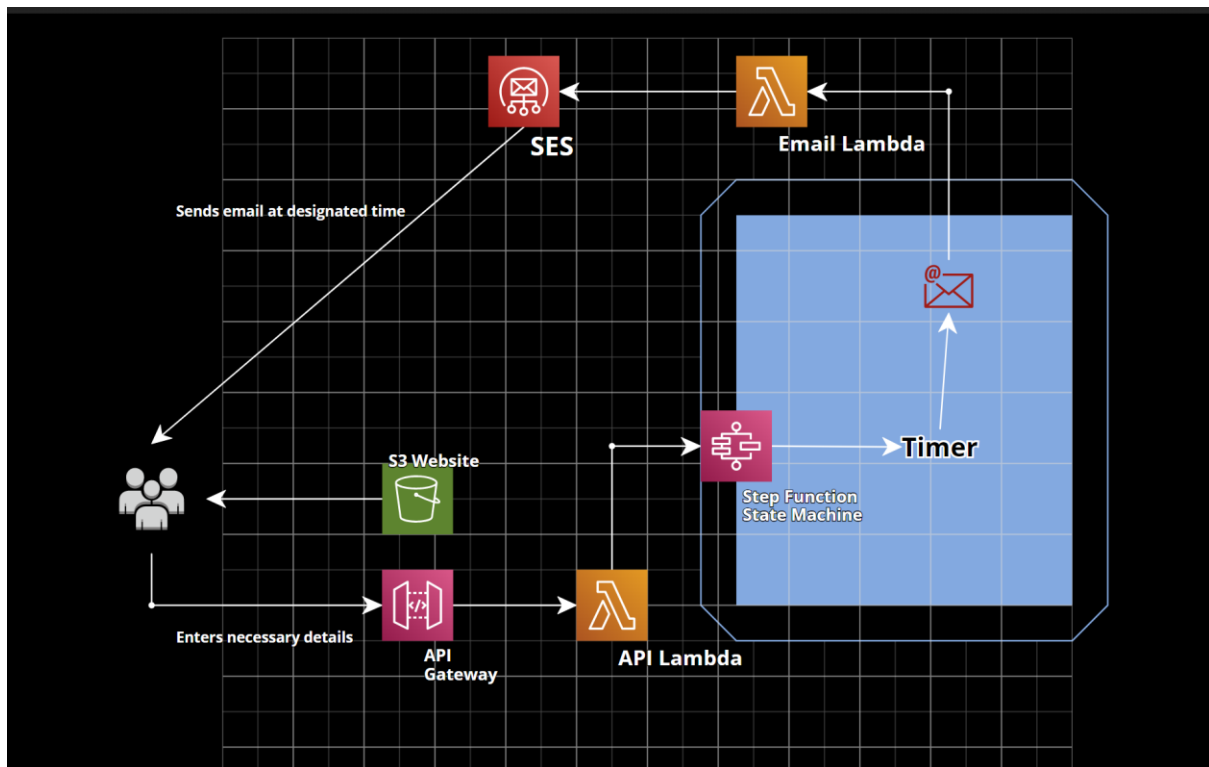
This project report describes the development of a simple AWS serverless reminder service, a serverless application designed to create fast reminders sent to the email id. This application uses various services of AWS such as SES, SNS, API Gateways, Lambda Functions, S3 for storage and Static hosting and CloudWatch for Logging.

Introduction:

The application was designed to provide a seamless user experience, ensure scalability, and minimize costs. The project's goal was to develop a serverless application that created a reminder to the email address in case the idea or a message is forgotten. This report outlines the architecture and design of the application, the development and testing process, and the application's results and limitations.

Materials and Methods:





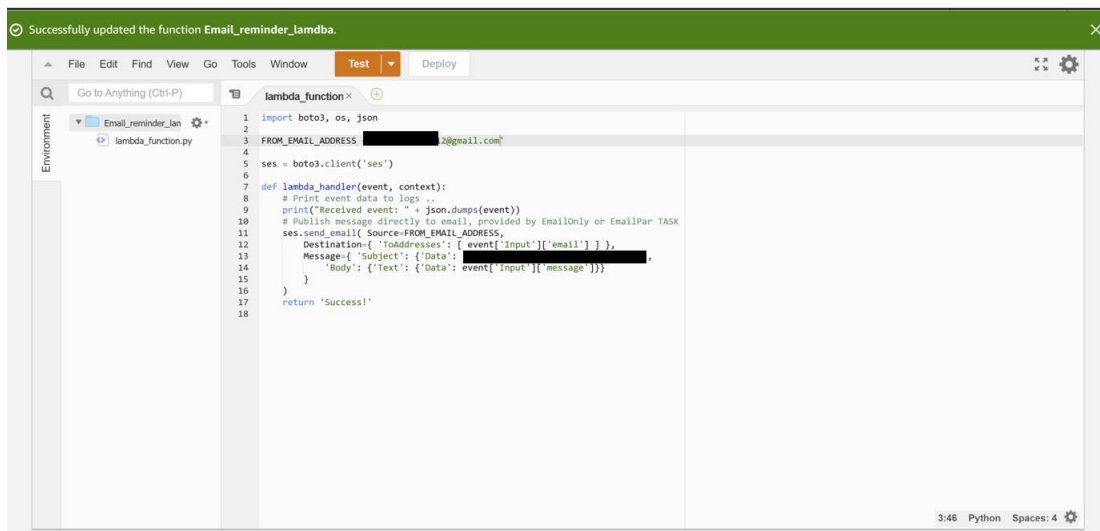
This is the whole architecture of the application where the user will be provided the webpage to interact by the S3 hosted website and sends the details to the API Gateway which then invokes the API Lambda function and sends it to the Step Function of the State Machine which is configured to use SES via the email Lambda function after a set amount of time decided by the user initially.

The Email reminder application is going to send reminder messages using Email service in AWS called SES (Simple Email Service). We run this service in sandbox mode which makes the application only send messages to the application to the verified addresses. We first confirm the identities of the addresses in SES which will be both sender and receiver.

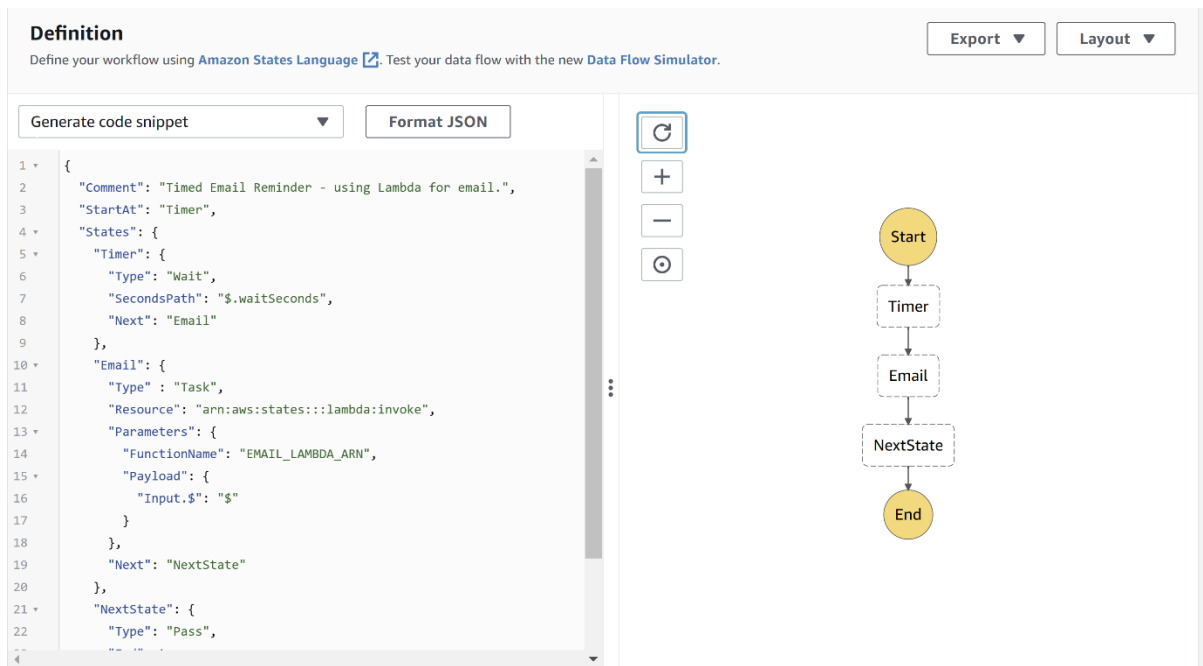
We then create an IAM execution role for the Lambda function for access of the services of SES, SNS and Logging Permissions, which will be used by the serverless application to create an email and then send it using SES.

The function is configured to send the email and the message to the receiver. The sender is inserted inside the lambda function and the receiver is inserted by the application.

Note : Lambda functions are created using Python 3.9 version

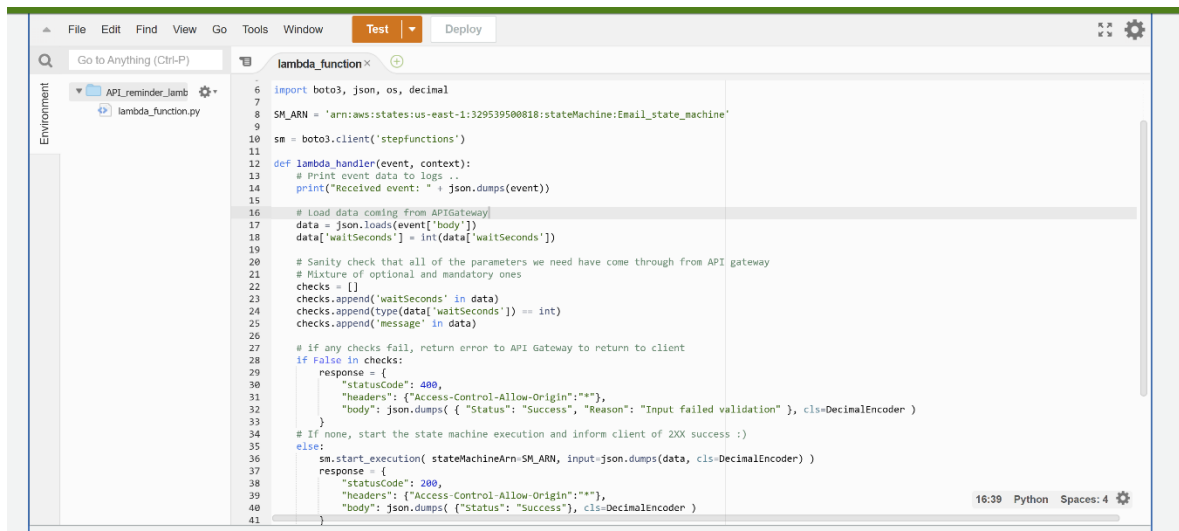


We then create an IAM role for the State machine which provide permissions for logging, invoking the lambda function when it needs to send emails and SNS to send text messages. We then create the State machine in which we write the workflow in code in the Amazon State Language (ASL).



We then configure the State machine with the function name of the lambda ARN.

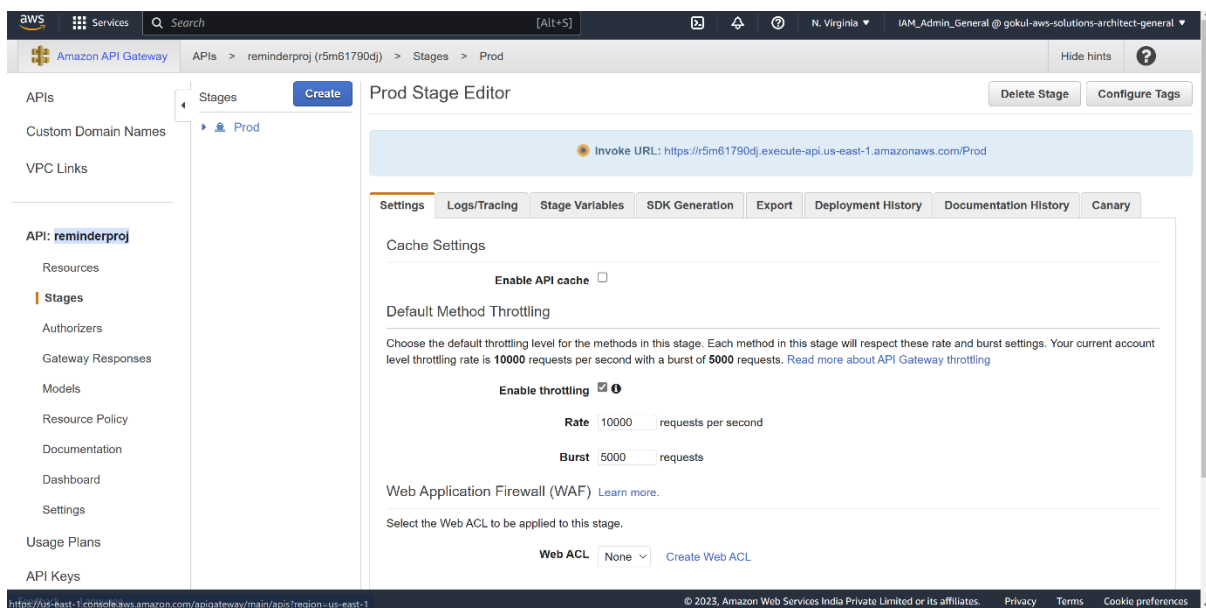
After that, we create the front end using a Rest API for the serverless application using API Gateways. First we create the API lambda which will support the API Gateway.



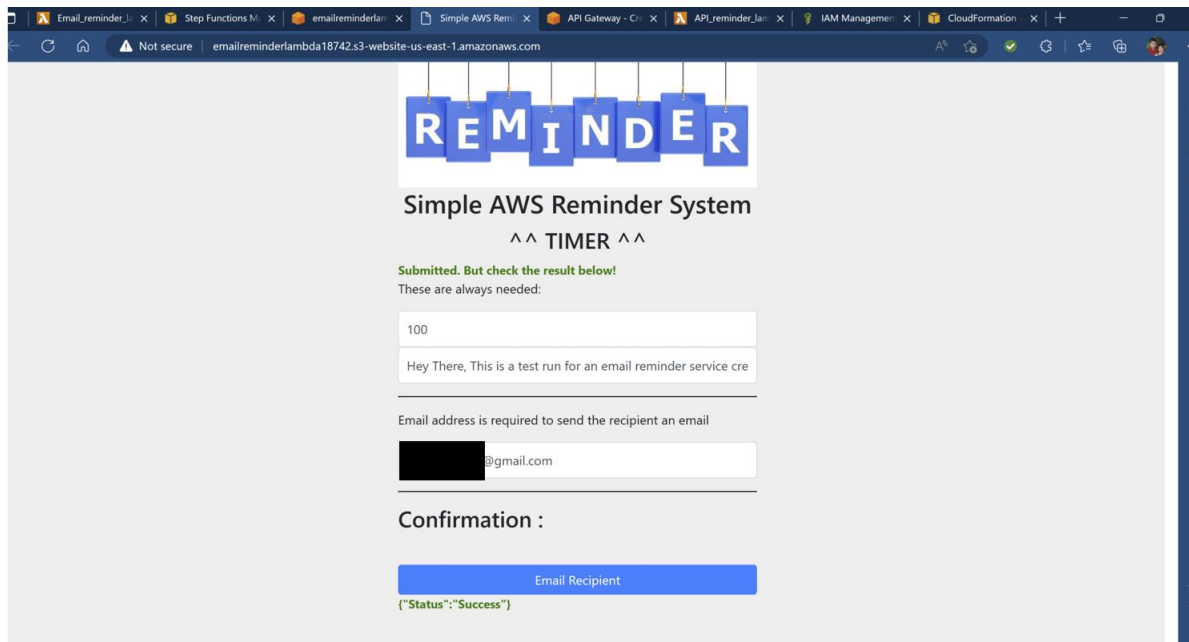
```
6 import boto3, json, os, decimal
7
8 SM_ARN = 'arn:aws:states:us-east-1:329539500818:stateMachine:Email_state_machine'
9
10 sm = boto3.client('stepfunctions')
11
12 def lambda_handler(event, context):
13     # Print event data to logs ..
14     print("Received event: " + json.dumps(event))
15
16     # Load data coming from API Gateway
17     data = json.loads(event['body'])
18     data['waitSeconds'] = int(data['waitSeconds'])
19
20     # Sanity check that all of the parameters we need have come through from API gateway
21     # Mixture of optional and mandatory ones
22     checks = []
23     checks.append('waitSeconds' in data)
24     checks.append(type(data['waitSeconds']) == int)
25     checks.append('message' in data)
26
27     # If any checks fail, return error to API Gateway to return to client
28     if False in checks:
29         response = {
30             "statusCode": 400,
31             "headers": {"Access-Control-Allow-Origin": ""},
32             "body": json.dumps( { "Status": "Success", "Reason": "Input failed validation" }, cls=DecimalEncoder )
33         }
34     # If none, start the state machine execution and inform client of 200 success :)
35     else:
36         sm.start_execution( stateMachineArn=SM_ARN, input=json.dumps(data, cls=DecimalEncoder) )
37         response = {
38             "statusCode": 200,
39             "headers": {"Access-Control-Allow-Origin": ""},
40             "body": json.dumps( { "Status": "Success" }, cls=DecimalEncoder )
41         }
```

This is the function which will provide compute to the API Gateway. It's job is to be called by API Gateway when its used by the serverless front end part of the application (loaded by S3) It accepts some information from you, via API Gateway and then it starts a state machine execution - which is the logic of the application.

Now we create the Rest API in the API Gateway. We create a resource followed by a method which makes sure that all of the information provided to this API is sent on to lambda for processing in the event data structure and deploy the API.



Now we create the front end of the application. The front end loads from S3, runs in your browser and communicates with this API. We create the S3 bucket with public access and add the necessary files to the bucket and enable static hosting to be used as the website for the API. After Uploading, we test the website application.



Implementation:

In the implementation, we try to add other emails other than confirmed emails in SES and as expected failed to process.

CloudWatch Logs Insights

CloudWatch Logs Insights provides a query language for analyzing log entries. The table below lists the last 20 log entries for your state machine. [Learn more](#)

					1h	3h	12h	1d	3d	1w	Custom		
Recent Logs													
#	@timestamp	@logStream	: type		: details								
1	2023-02-22T07:38:54.281Z	states/Email_state_machine/2023-02-22-07/71ee45a3	ExecutionFailed		{\"cause\": \"\\\"errorMessage\\\": \\\"An e								
2	2023-02-22T07:38:53.578Z	states/Email_state_machine/2023-02-22-07/7a61b537	TaskStarted		{\"resource\": \"invoke\", \"resourceType								
3	2023-02-22T07:38:53.517Z	states/Email_state_machine/2023-02-22-07/7a61b537	WaitStateExited		{\"name\": \"Timer\", \"output\": {\\\"waitSi								
4	2023-02-22T07:38:53.517Z	states/Email_state_machine/2023-02-22-07/7a61b537	TaskStateEntered		{\"input\": {\\\"waitSeconds\\\": 120, \\								
5	2023-02-22T07:38:53.517Z	states/Email_state_machine/2023-02-22-07/7a61b537	TaskScheduled		{\"parameters\": {\\\"FunctionName\\\": \\								
6	2023-02-22T07:36:53.471Z	states/Email_state_machine/2023-02-22-07/7a61b537	WaitStateEntered		{\"input\": {\\\"waitSeconds\\\": 120, \\								
7	2023-02-22T07:36:53.429Z	states/Email_state_machine/2023-02-22-07/7a61b537	ExecutionStarted		{\"input\": {\\\"waitSeconds\\\": 120, \\								
8	2023-02-22T07:36:53.251Z	states/Email_state_machine/2023-02-22-07/71ee45a3	WaitStateEntered		{\"input\": {\\\"waitSeconds\\\": 120, \\								
9	2023-02-22T07:36:53.193Z	states/Email_state_machine/2023-02-22-07/71ee45a3	ExecutionStarted		{\"input\": {\\\"waitSeconds\\\": 120, \\								

Cloudwatch Logs clearly show the failed process

Results:

The Serverless Reminder Application has successfully met all functional and non-functional requirements. User feedback was positive, with users reporting high satisfaction with the user experience. Metrics on system performance and scalability indicated that the application was highly scalable and cost-effective. Limitations of the application included the need for further testing and refinement of some features and making the SES service able to send messages to any email address. Addition of SNS service to send messages through SMS can also be considered for the application.

Conclusion:

In conclusion, the application was designed successfully to be highly scalable, cost-effective, and user-friendly. Future work may involve additional testing and refinement of features, as well as the addition of new features to enhance the user experience. Overall, the AWS Reminder service represents a successful application of serverless architecture to a real-world problem.