Build a Multi-Language Notification System Using Amazon Pinpoint and Amazon Translate

**SPL-TF-300-SVMLTP-1 - Version 1.0.8**

© 2024 Amazon Web Services, Inc. or its affiliates. All rights reserved. This work may not be reproduced or redistributed, in whole or in part, without prior written permission from Amazon Web Services, Inc. Commercial copying, lending, or selling is prohibited. All trademarks are the property of their owners.

Note: Do not include any personal, identifying, or confidential information into the lab environment. Information entered may be visible to others.

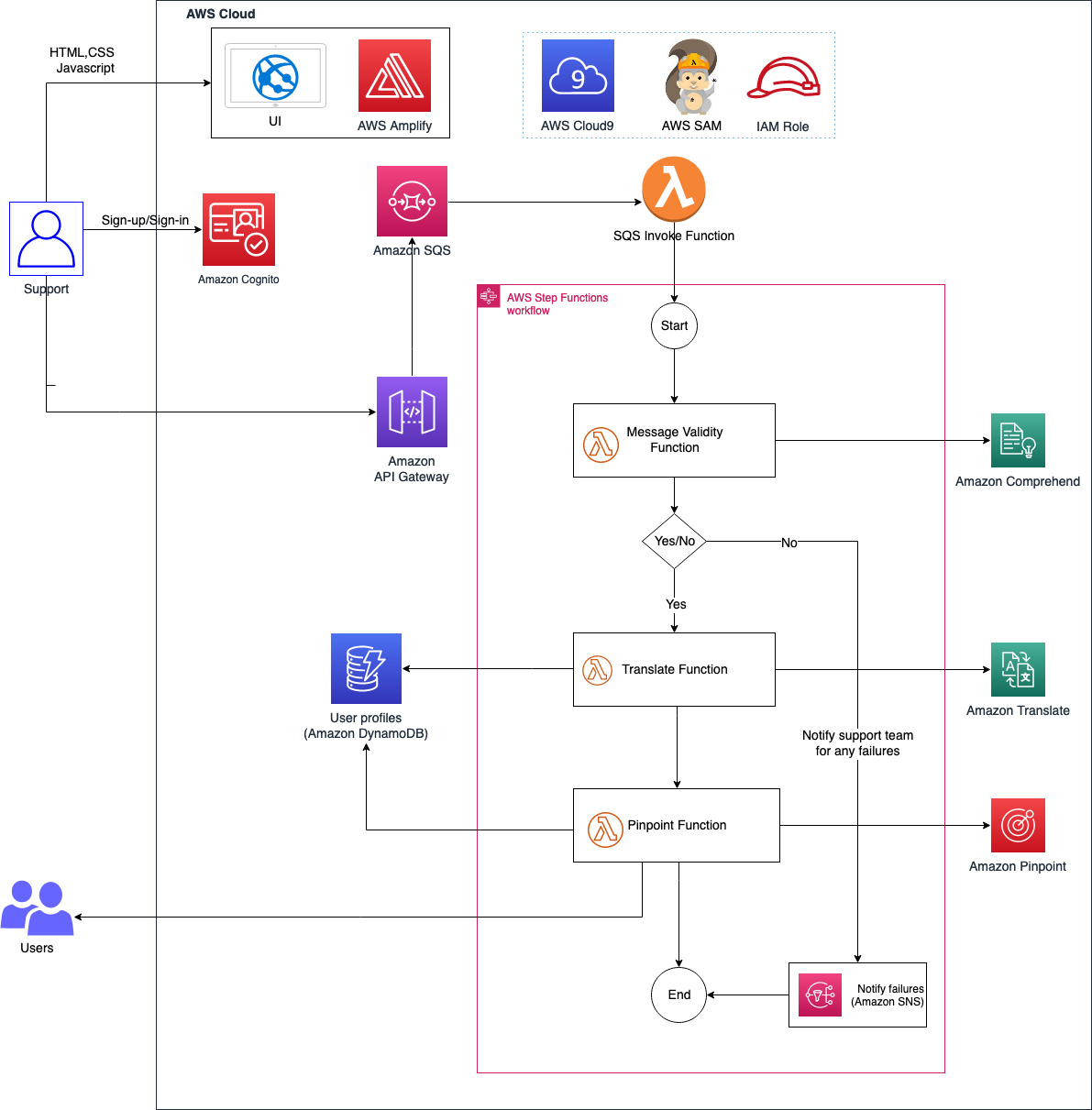
Corrections, feedback, or other questions? Contact us at [*AWS Training and Certification*](https://support.aws.amazon.com/#/contacts/aws-training).

**Lab overview**

Organizations with global operations can struggle to notify their customers of any business-related announcements or notifications in different languages. Their customers want to receive notifications in their local language and communication preference. Organizations often rely on complicated third-party services or individuals to manually translate the notifications. This can lead to a loss of revenue because of delayed communication and additional operational expenses.

This lab demonstrates how a customer support department communicates with its customers when a significant event occurs in their preferred language. The communication workflow is orchestrated using AWS Step Functions, which invokes multiple AWS Lambda functions. The workflow takes the input notification message and translates it into different languages using Amazon Translate. The translated notification is delivered in different formats, based on the customer preference, using Amazon Pinpoint. The languages and format preferences are defined in an Amazon DynamoDB table.

The following architecture diagram shows the components that have been, or will be, deployed in this lab.



This lab uses the following AWS services:

* Amazon Pinpoint
* Amazon Translate
* AWS Lambda
* Amazon API Gateway
* Amazon Simple Storage Service (Amazon S3)
* AWS Step Functions
* Amazon Simple Queue Service (Amazon SQS)
* Amazon Simple Notification Service (Amazon SNS)
* AWS Cloud9
* AWS Identity and Access Management (IAM)
* AWS Amplify
* Amazon Cognito
* Amazon Comprehend

OBJECTIVES

By the end of this lab, you will be able to do the following:

* Deploy a serverless backend application using the AWS Server Application Model (AWS SAM).
* Set up an Amazon Pinpoint project using the AWS Management Console.
* Configure data in a DynamoDB table.
* Deploy the frontend application using Amplify.
* Post a notification using the UI, and receive the notification in different languages and different formats.

DURATION

The lab will take approximately *75* minutes to complete.

ICON KEY

Various icons are used throughout this lab to call attention to certain aspects of the guide. The following list explains the purpose for each one:

* Specifies the command you must run.
* Verify the output of a command or edited file.
* Specifies important hints, tips, guidance, or advice.
* Specifies where more information can be found.

TECHNICAL KNOWLEDGE PREREQUISITES

To successfully complete this lab, you should be familiar with basic navigation of the AWS Management Console and be comfortable editing scripts using a text editor.

**Start lab**

1. To launch the lab, at the top of the page, choose **Start lab**.

**Caution:** You must wait for the provisioned AWS services to be ready before you can continue.

1. To open the lab, choose **Open Console**.

You are automatically signed in to the AWS Management Console in a new web browser tab.

**Warning:** Do not change the **Region** unless instructed.

COMMON SIGN-IN ERRORS

**Error: Choosing Start Lab has no effect**

In some cases, certain pop-up or script blocker web browser extensions might prevent the **Start Lab** button from working as intended. If you experience an issue starting the lab:

* Add the lab domain name to your pop-up or script blocker’s allow list or turn it off.
* Refresh the page and try again.

**Task 1: Understand key services and set up application code**

In this task, you review the key services that you will use and set up the application code for the lab.

* **Amazon Pinpoint** is a flexible and scalable outbound and inbound marketing communications service. You can connect with customers over channels such as email, SMS, push, and voice or in-app messaging. Amazon Pinpoint is designed to be straightforward to set up and use and is flexible for all marketing communication scenarios. Segment your campaign audience for the right customer and personalize your messages with the right content. Delivery and campaign metrics in Amazon Pinpoint measure the success of your communications. Amazon Pinpoint can grow with you and scales globally to billions of messages per day across channels.
* **Amazon Translate** is a neural machine translation service that is designed to deliver fast, high-quality, and affordable language translation. Neural machine translation is a form of language translation automation that uses deep learning models to deliver more accurate and more natural sounding translation than traditional statistical and rule-based translation algorithms. With Amazon Translate, you can localize content such as websites and applications for your diverse users, translate large volumes of text for analysis, and efficiently facilitate cross-lingual communication between users.
* **AWS Lambda** is a serverless compute service that you can use to run code without provisioning or managing servers, creating workload-aware cluster scaling logic, maintaining event integrations, or managing runtimes. With Lambda, you can run code for virtually any type of application or backend service—all with zero administration. You can write Lambda functions in various languages, such as Node.js, Python, Go, and Java. And you can use both serverless and container tools, such as the AWS SAM or Docker CLI, to build, test, and deploy your functions.
* **Amazon Simple Queue Service (Amazon SQS)** is a fully managed message queuing service that you can use to decouple and scale microservices, distributed systems, and serverless applications. It is designed to reduce the complexity and overhead associated with managing and operating message-oriented middleware, and it empowers developers to focus on differentiating work. Using Amazon SQS, you can send, store, and receive messages between software components at any volume, without losing messages or requiring other services to be available.
* **Amazon API Gateway** is a fully managed service that developers can use to create, publish, maintain, monitor, and secure APIs at any scale. APIs act as the “front door” for applications to access data, business logic, or functionality from your backend services. Using API Gateway, you can create RESTful APIs and WebSocket APIs that facilitate real-time two-way communication applications. API Gateway supports containerized and serverless workloads in addition to web applications.
* **Amazon Simple Notification Service (Amazon SNS)** is a fully managed messaging service for both application-to-application (A2A) and application-to-person (A2P) communication. The A2A pub/sub functionality provides topics for high-throughput, push-based, many-to-many messaging between distributed systems, microservices, and event-driven serverless applications. Using SNS topics, your publisher systems can fan out messages to a large number of subscriber systems, such as SQS queues, Lambda functions and HTTPS endpoints, for parallel processing, and Amazon Kinesis Data Firehose. You can use the A2P functionality to send messages to users at scale through SMS, mobile push, and email.
* **Amazon Simple Email Service (Amazon SES)** is a cost-effective, flexible, and scalable email service that developers can use to send mail from within any application. You can configure Amazon SES quickly to support several email use cases, including transactional, marketing, or mass email communications. The Amazon SES flexible IP deployment and email authentication options help drive higher deliverability and protect sender reputation. And sending analytics measure the impact of each email message. With Amazon SES, you can send email securely, globally, and at scale.
* **AWS Step Functions** is a serverless function orchestrator that you can use to sequence Lambda functions and multiple Amazon Web Services (AWS) offerings into business-critical applications. Through its visual interface, you can create and run a series of checkpointed and event-driven workflows that maintain the application state. The output of one step acts as input to the next. Each step in your application runs in order, as defined by your business logic.
* **AWS Cloud9** is a cloud-based integrated development environment (IDE) that you can use to write, run, and debug your code with just a browser. It includes a code editor, debugger, and terminal. You can use it to write, run, and debug serverless applications. It pre-configures the development environment with all the SDKs, libraries, and plugins needed for serverless development.
* **Amazon DynamoDB** is a key-value and document database that delivers single-digit millisecond performance at any scale. It’s a fully managed, multi-Region, durable database with built-in security, backup and restore, and in-memory caching for internet-scale applications. DynamoDB can handle more than 10 trillion requests per day and can support peaks of more than 20 million requests per second.
* **AWS Amplify** is a set of purpose-built tools and features that frontend web developers and mobile developers can use to quickly build full-stack applications on AWS. With Amplify, you can configure a web or mobile app backend, and connect your app in minutes. You can visually build a web frontend UI and manage app content outside the AWS Management Console. You can use Amplify to ship faster and scale with little effort and cloud expertise.
* **Amazon Cognito** is an AWS service with which you can quickly add user sign-up, sign-in, and access control to your web and mobile apps. It scales to millions of users and supports sign-in with social identity providers, such as Apple, Facebook, Google, and Amazon, and enterprise identity providers through SAML 2.0 and OpenID Connect.
* **Amazon Comprehend** is a natural language processing (NLP) service that uses machine learning (ML) to discover insights from text. It provides Custom Entity Recognition, Custom Classification, Keyphrase Extraction, Sentiment Analysis, Entity Recognition, and more APIs. You can use these to integrate NLP into your applications. You only need to call the Amazon Comprehend APIs in your application and provide the location of the source document or text. The APIs will output entities, key phrases, sentiment, and language in a JSON format, which you can use in your application.

To set up the application code for this lab, use the following steps:

1. In the AWS Management Console, use the **AWS search bar** to search for

Cloud9

, and then choose the service from the list.

1. In the **Environments** section, you should see an environment with the name **MultiNotificationApp** created for this lab purpose.
2. Under **Cloud9 IDE** , choose **Open**.

Within a few seconds, the AWS Cloud9 environment launches. Notice the Linux-style terminal window in the bottom pane.

1. In the **AWS Cloud9** terminal, to download the source code,  run the following commands:

* Replace the *<AWS\_REGION>* placeholder value with the value of **Region** provided to the left of these instructions.

cd ~/environment

wget https://<AWS\_REGION>-tcprod.s3.<AWS\_REGION>.amazonaws.com/courses/SPL-TF-300-SVMLTP/v1.0.8.prod-9375df94/scripts/notification-app.zip

unzip notification-app.zip

cd notification-app

**Note** Keep the browser tab that’s running the AWS Cloud9 IDE open because you will use it throughout this lab.

**Task 2: Deploy the backend application using the AWS SAM**

In this task, you run AWS Command Line Interface (AWS CLI) commands to deploy the backend code using the AWS SAM.

The following are some key AWS SAM CLI commands to know:

* **sam build:** Builds a serverless application and prepares it for subsequent steps in your workflow, such as locally testing the application or deploying it to the AWS Cloud
* **sam deploy:** Deploys an AWS SAM application
* **sam init:** Initializes a serverless application with an AWS SAM template

**Learn more** For more information about AWS SAM CLI commands, see [AWS SAM CLI Command Reference](https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/serverless-sam-cli-command-reference.html).

1. Choose the AWS Cloud9 browser tab.
2. In the navigation pane of the AWS Cloud9 IDE, choose the dropdown arrow next to the **notification-app** folder and then the **backend** folder.
3. Choose and open the **template.yaml** file.

This file contains the AWS SAM template that defines your application’s AWS resources. Take a moment to scroll through and review the anatomy of the architecture.

1. On the left side of the AWS Cloud9 IDE, choose the dropdown arrow next to the **src** folder, and then choose the dropdown arrow next to the **translate\_function** folder.
2. Choose and open the **app.py** file to view the code within this file.

This file contains your actual Lambda handler logic for the **translate\_function** Lambda function.

Review the other Lambda functions in the **comprehend\_function**, **pinpoint\_function**, **sqs\_invoke** folders. The **comprehend\_function** checks the validity of the message to make sure that the message posted has a score greater than 0.5.

Review the state machine definition in the **statemachine** folder. The statemachine invokes all the Lambda functions except the **sqs\_invoke** function.

1. Choose and open the **samconfig.toml** file.

This is the configuration file for the backend application. To replace *BUCKET\_NAME*, *AWS\_REGION* and the role Amazon Resource Names (ARNs) with actual values in the **samconfig.toml** file, you will run a set of CLI commands to update these values before deploying the application.

1. To update the values in the **samconfig.toml** file, in the **AWS Cloud9** terminal,  run the following AWS CLI and bash commands:

sudo yum -y install jq

cd ~/environment/notification-app/backend

export BUCKET\_NAME=$(aws s3api list-buckets --query "Buckets[?contains(Name, 'multi-notification-app')].Name" --output text)

sed -Ei "s|<BUCKET\_NAME>|${BUCKET\_NAME}|g" samconfig.toml

export AWS\_REGION=$(curl -s 169.254.169.254/latest/dynamic/instance-identity/document | jq -r '.region')

sed -Ei "s|<AWS\_REGION>|${AWS\_REGION}|g" samconfig.toml

export LAMBDA\_ROLE\_ARN=$(aws iam list-roles --query "Roles[?contains(RoleName, 'LambdaRole')].Arn" --output text)

sed -Ei "s|<LAMBDA\_ROLE\_ARN>|${LAMBDA\_ROLE\_ARN}|g" samconfig.toml

export SQS\_ROLE\_ARN=$(aws iam list-roles --query "Roles[?contains(RoleName, 'SQSRole')].Arn" --output text)

sed -Ei "s|<SQS\_ROLE\_ARN>|${SQS\_ROLE\_ARN}|g" samconfig.toml

export STEP\_FUNCTIONS\_ROLE\_ARN=$(aws iam list-roles --query "Roles[?contains(RoleName, 'StepFunctionsRole')].Arn" --output text)

sed -Ei "s|<STEP\_FUNCTIONS\_ROLE\_ARN>|${STEP\_FUNCTIONS\_ROLE\_ARN}|g" samconfig.toml

1. To save the updated **samconfig.toml** file, on the **File** menu, choose **Save**.

**Note** This file is usually generated during the **sam deploy --guided** command, but for this lab, the prebuilt file is provided. This file contains the parameters of the S3 bucket from where the code is deployed, the role needed for the AWS SAM to deploy, and the AWS Region. The file contains the parameters for the AWS CloudFormation stack to be deployed.

1. In the AWS Cloud9 terminal, , run the following command:

sam deploy

This command deploys your application to the AWS Cloud. The command takes the deployment artifacts that you build with the **sam build** command, packages and uploads them to an S3 bucket created by the AWS SAM CLI, and deploys the application using CloudFormation.

**Note** The command takes a minute or two to run.

**Learn more** The AWS SAM CLI comes preinstalled on AWS Cloud9. For more information about installing the AWS SAM CLI, see [Installing the AWS SAM CLI on Linux](https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/serverless-sam-cli-install-linux.html).

You can see the output for the changeset. Notice the operations, in this case **+ Add**, and look under **ResourceType** to see the AWS resources about to be deployed.

The changeset is now being deployed. You can watch the sequence of events from your AWS Cloud9 instance and from CloudFormation.

1. In the console, use the **AWS search bar** to search for

CloudFormation

, and then choose the service from the list.

1. Choose the **multi-notification-app** stack.

This is the stack that was deployed from your backend code in AWS Cloud9.

1. To view the AWS resources that were created within this stack, choose the **Resources** tab.
2. To open the DynamoDB resource, choose the **Physical ID** of **user\_profiles** with **Logical ID** named **EventTable**.

This is the table created in DynamoDB using the AWS SAM template.

1. In the left navigation pane, choose **Tables**.
2. Choose **user\_profiles**.
3. In the upper-right corner of the page, choose **Explore table items**.

Nothing is populated in the table yet because no user profiles have been added.

1. In the console, use the **AWS search bar** to search for

API Gateway

, and then choose the service from the list.

This is the **MlnApi** API that was launched through the AWS SAM template.

1. Choose **MLNApp**.
2. On the left side of the page, choose **Stages**.
3. In the **Stages** pane, choose the  **dev** stage.

You can view the endpoint URL in the right pane.

1. In the console, use the **AWS search bar** to search for

Step Functions

, and then choose the service from the list.

1. Choose the state machine with the name **EventStateMachine** and choose **Edit**.

You can view the state machine definition deployed through the AWS SAM template.

**Task 3: Register an email address and a phone number using the Amazon Pinpoint console**

Amazon Pinpoint email, voice, push notification, and SMS channels offers deliverability and scale to reach hundreds of millions of customers around the globe. With Amazon Pinpoint, you can start sending emails in minutes, and you only pay for what you use.

Transactional messages are on-demand messages that you send to specific recipients. You can use the Amazon Pinpoint API and the AWS SDKs to send transactional messages through email, push, SMS, or voice. You can also programmatically send a transactional message from a customer activity in web or mobile applications. For example, you can automatically send a receipt when a customer makes a purchase or deliver a one-time password immediately after a customer requests it.

In this task, you create an Amazon Pinpoint project using the console. The serverless application invokes the Amazon Pinpoint service to send the notifications to the customers.

1. In the console, use the **AWS search bar** to search for

Pinpoint

, and then choose the service from the list.

1. Choose the **AWS\_REGION** that you have deployed the application in.
2. In the **Get Started** section, choose **Pinpoint campaign orchestration** and select **Manage Projects**.
3. For **Project name**, enter

MLNproject

.

1. Choose **Create a project**.
2. In the **Configure features** page, under **Project features**, in the **Email** pane, choose **Configure**.
3. In the **Set up email** page, under **Verify an email address**, for **Email address**, enter a valid email address.
4. Choose **Verify**.

This will send a verification link to the email address that you specified. You will need to verify the link in your email before you proceed to the next step.

1. In the **Amazon Pinpoint** browser tab, choose **Save**.

**Note** If you want to use different email addresses for sender and recipient, you will need to register and verify both the recipient and the sender email addresses in Amazon Pinpoint. Otherwise, you can use the same email address for sender and recipient.

1. In the navigation pane, under **Settings**, choose **SMS and voice** and scroll down to the **Phone numbers** section.
2. Choose **Request phone number**.
3. In the **Define your phone numbers** page, under **Phone number 1**, choose the dropdown list under **Country**.
4. In the dropdown list, choose **United States**.

**Note** You see that more attributes are populated in the **Phone number 1** page.

1. Configure the following details:

* For **Number type**, choose **Long code**.
* For **Capabilities**, choose **Voice**.
* For **Default message type**, choose **Transactional**.
* Keep the defaults in the **Summary** section.

1. Choose **Next**
2. In the **Review and request** page, choose **Request**

A phone number is created and listed in the **Phone numbers** section. Copy the phone number to a text editor because you will use it in the later steps.

**Note** Notice that only the **Voice** option is enabled for the phone number that you created. You need to create a toll-free number for SMS to be enabled. However, according to new rules in the United States, you need to register the toll-free number as part of your company. You skip the toll-free registration for this lab because it needs approvals and takes more time.

1. Choose **MLNproject**.
2. In the navigation pane, choose **Settings**.
3. Choose **SMS and voice** and then, in the **SMS settings** pane, choose **Edit**.
4. In the **Edit SMS** page, under **General settings**, choose **Enable the SMS channel for this project**.
5. Choose **Save changes**.

The Amazon Pinpoint project setup is complete.

**Task 4: Update environment variables for the Amazon Pinpoint Lambda function**

In this task, you configure the Amazon Pinpoint Lambda function to invoke the Amazon Pinpoint project, the phone numbers, and the email IDs that you set up in the previous task.

1. In the console, use the **AWS search bar** to search for

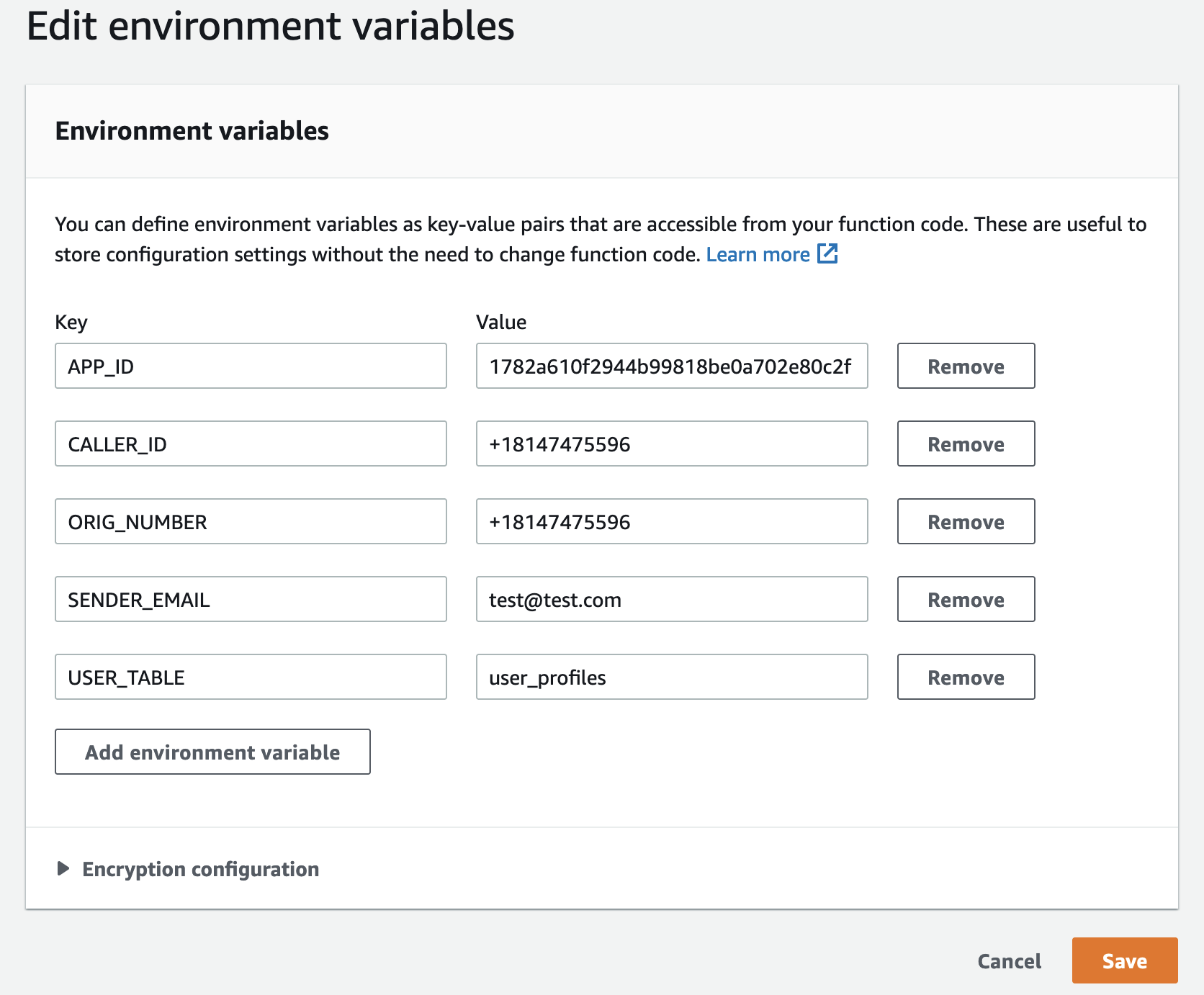
Lambda

, and then choose the service from the list.

1. Choose **multi-notification-app-PinpointFunction**.
2. Choose the **Configuration** tab and then choose the **Environment variables** section.
3. Choose **Edit** and update the values for the following variables:

* For **APP\_ID**, replace the value with the **MLNApp** Amazon Pinpoint project ID. You can find the Amazon Pinpoint project ID by choosing **All projects** in the navigation pane of the Amazon Pinpoint console.
* For **CALLER\_ID**, replace the value with the phone number that you requested in Amazon Pinpoint. Remove any spaces and dashes from the phone number.
* For **ORIG\_NUMBER**, replace the value with the phone number that you requested in Amazon Pinpoint. Remove any spaces and dashes from the phone number.
* For **SENDER\_EMAIL**, replace the email ID that you registered and verified in Amazon Pinpoint.

The environment variables should look similar to the attached screenshot for the Amazon Pinpoint Lambda function after updates:



**Note** Make sure there are no extra spaces at the end of the replaced values.

1. Choose **Save**.

**Task 5: Set up DynamoDB records to test the application**

In this task, you set up some records in the **user\_profiles** DynamoDB table. The application will use these records for the preferred way of communication and the users corresponding email addresses and phone numbers. So, make sure you enter a valid email address and phone number to receive the messages.

1. In the console, use the **AWS search bar** to search for

DynamoDB

, and then choose the service from the list.

1. In the navigation pane, choose **PartiQL editor**.
2. In the **Query 1** editor,  paste the following SQL statement.

INSERT INTO user\_profiles VALUE

{

'event\_id': 'Winter12022',

'user\_id': 'ui101',

'first\_name' : 'John',

'email': 'john@example.com', --A valid email id to receive email that has been verified in Pinpoint

'language' : 'en',

'phone': '+11235550101', --Use your phone number to receive the voice prompt

'preference': 'email',

'phoneme': 'en-US'

}

1. Before you proceed to the next step, in the SQL statement shown, replace the email and phone columns with a valid email address and a valid phone number.
2. Choose **Run**.
3. For the following inserts, repeat the previous three steps. You can only insert one statement at a time into the **PartiQL** editor.

**Note** The below insert statements use different preferences and languages. You can use the same phone number and email address that you used in the first insert statement.

INSERT INTO user\_profiles VALUE

{

'event\_id': 'Winter22022',

'user\_id': 'ui102',

'first\_name' : 'Mary',

'email': 'mary@example.com', --A valid email id to receive email that has been verified in Pinpoint

'language' : 'de',

'phone': '+11235550191', --Use your phone number to receive the voice prompt

'preference': 'sms',

'phoneme': 'de-DE'

}

INSERT INTO user\_profiles VALUE

{

'event\_id': 'Winter32022',

'user\_id': 'ui103',

'first\_name' : 'Arnav',

'email': 'doe@example.com', --A valid email id to receive email that has been verified in Pinpoint

'language' : 'fr',

'phone': '+11235550190', --Use your phone number to receive the voice prompt

'preference': 'voice',

'phoneme': 'fr-FR'

}

1. In the navigation pane, choose **Explore items**.
2. To view all the items inserted into the table, choose **user\_profiles** table.

**Task 6: Review the frontend application code and update the configuration file**

AWS Amplify Hosting is a fully managed continuous integration and continuous delivery (CI/CD) and hosting service for fast, secure, and reliable static and server-side rendered apps that scale with your business. It supports modern web frameworks such as React, Angular, Vue, Next.js, Gatsby, Hugo, Jekyll, and more.

In this task, you build the frontend for the multi-notification application using JavaScript libraries and upload the zip file to the S3 bucket.

1. Return to the AWS Cloud9 IDE browser tab.
2. To switch to your **frontend** directory and install packages and dependencies, run the following commands:

cd ../frontend

npm install

1. On the left side of the AWS Cloud9 IDE, choose the dropdown arrow next to the **frontend** folder, and then choose the dropdown arrow next to the **src** folder.
2. Choose and open the **main.js** file to view the code. The following is an example.

import Vue from 'vue'

import App from './App'

import router from './router'

import '@aws-amplify/ui-vue';

import Amplify from '@aws-amplify/core';

import { Auth } from '@aws-amplify/auth';

import awsmobile from './aws-exports';

Amplify.configure(awsmobile);

Auth.configure(awsmobile)

Vue.config.productionTip = false

/\* eslint-disable no-new \*/

new Vue({

el: '#app',

router,

components: { App },

template: '<App/>'

})

Notice that the Vue framework has been imported in addition to **import { Auth } from ‘@aws-amplify/auth’;**, which is used for Amazon Cognito authentication.

Next, you need to update the **aws-exports.js** file to add the Amazon Cognito user pool that was created earlier.

1. Choose and open the **aws-exports.js** file.
2. In the AWS Cloud9 terminal,  run the following AWS CLI and bash commands to replace the parameters with the actual values in the **aws-exports.js** file.

cd src

export API\_GATEWAY\_ID=$(aws apigateway get-rest-apis --query 'items[?name==`MLNApp`].id' --output text)

export AWS\_REGION=$(curl -s 169.254.169.254/latest/dynamic/instance-identity/document | jq -r '.region')

sed -Ei "s|<AWS\_REGION>|${AWS\_REGION}|g" aws-exports.js

export API\_GATEWAY\_URL=https://${API\_GATEWAY\_ID}.execute-api.${AWS\_REGION}.amazonaws.com/dev

sed -Ei "s|<API\_GATEWAY\_URL>|${API\_GATEWAY\_URL}|g" aws-exports.js

export COGNITO\_USER\_POOL\_ID=$(aws cognito-idp list-user-pools --query "UserPools[?contains(Name, 'MLNUserPool')].Id" --max-results 1 --output text)

sed -Ei "s|<COGNITO\_USER\_POOL\_ID>|${COGNITO\_USER\_POOL\_ID}|g" aws-exports.js

export APP\_CLIENT\_ID=$(aws cognito-idp list-user-pool-clients --user-pool-id ${COGNITO\_USER\_POOL\_ID} --query "UserPoolClients[?contains(ClientName, 'AppClientForMLNUserPool')].ClientId" --output text)

sed -Ei "s|<APP\_CLIENT\_ID>|${APP\_CLIENT\_ID}|g" aws-exports.js

cd ..

1. Save the **aws-exports.js** file.

 The changed file should resemble the following screenshot:



**Note** The frontend of the application can connect to the backend using the parameters provided in the **aws-exports.js** file. The URL provided is the backend URL for the application that was deployed through AWS SAM.

1. To build the application for production,  run the following command:

npm run build

Notice the **Build complete** message.

Next, you need to zip your build files so that you can then launch the application with Amplify.

1. From the AWS Cloud9 terminal,  run the following command:

cd ~/environment/notification-app/frontend/dist

ls

This command switches you into the **dist** directory, which has the build assets from the previously run **build** command.

1. To zip the contents of the build folder,  run the following command:

zip -r app.zip \*

1. To upload the **app.zip** file to the pre-provisioned S3 bucket,  run the following commands:

export BUCKET\_NAME=$(aws s3api list-buckets --query "Buckets[?contains(Name, 'notification')].Name" --output text)

aws s3 cp app.zip s3://${BUCKET\_NAME}

**Task 7: Deploy the multi-notification application using Amplify**

AWS offers purpose-built tools and services for frontend web and mobile developers through Amplify, which they can use to build apps with cloud functionality on AWS, so you can get to market faster.

In this task, you deploy the multi-notification application using the Amplify console.

1. In the console, use the **AWS search bar** to search for

Amplify

, and then choose the service from the list.

1. Choose **GET STARTED**.
2. In the **Get Started** section, under **Amplify Hosting**, choose **Get started**.
3. Choose **Deploy without Git provider**, and then choose **Continue**
4. For **Manual deploy**, configure the following information:

* For **App name**, enter

Multi-Language-NotificationApp

* For **Environment name**, enter

dev

* For **Method**, choose **Amazon S3**.
* For **Bucket**, choose the bucket with the name **multi-notification-app** in it.
* For **Zip file**, choose **app.zip** (when the **Bucket** is chosen, this dropdown menu auto-populates.)

1. Choose **Save and deploy**

That’s it! You just deployed your frontend code to AWS through Amplify. You should see a green **Deployment successfully completed** bar in the middle of your page.

**Task 8: Test the multi-notification application**

Amazon Cognito user pools provide a secure identity store that scales to millions of users. As a fully managed service, user pools can be set up without provisioning any infrastructure. They store user profiles and support authentication for users who sign up directly and for federated users who sign in with social and enterprise identity providers.

Amazon Cognito provides a built-in and customizable UI for user sign-up and sign-in. You can use Android, iOS, and JavaScript SDKs for Amazon Cognito to add user sign-up and sign-in pages to your apps.

In this task, you test the multi-notification application that was deployed in Amplify in the previous task. You start by creating an account, and then post a notification in the application. To verify, you can check either your email or your phone for a voice message, depending on the event ID that you used for testing.

1. In the Amplify console, choose the URL under **Domain**.

This opens the notification application.

1. On the multi-notification application page, choose **Create account**.
2. Complete the fields with your information, and choose **CREATE ACCOUNT**
3. To post a new notification, enter the following information:

* **EventId:** Enter one of the event\_id values that you inserted into the **user\_profiles** table in DynamoDB earlier, during task 5. For example, **Winter12022**.
* **Message:** Enter the following:

The company is adding a new item to the menu, which will go live by December 10th. Please ensure that you are prepared for this change and plan out accordingly.

1. Choose **SEND MESSAGE**
2. In the console, use the **AWS search bar** to search for

Step Functions

, and then choose the service from the list.

1. Choose the latest **Execution Id**. It should have a status of **Succeeded**.
2. Check your email if you have entered an eventId to receive an email. Otherwise check your phone for a voice message, depending on the eventId.

**Challenge task**

In this lab, Amazon Comprehend is used to validate the message authenticity. According to your company rules, you can change the score on the message authenticity. If it falls below the expected score, the whole flow fails. For example, if you use inappropriate or unacceptable words, the workflow fails.

This is very important for organizations. They need to ensure that inappropriate messages are not sent to their users and determine how they can automate this validation and prevent it automatically.

For this challenge task, you update the score value and try out different messages. Also, check the Step Functions workflow or the Amazon CloudWatch logs of the **comprehend\_function** to see what the score is, depending on the messages you post.

1. Return to the AWS Cloud9 IDE browser tab and choose the **comprehend\_function**.
2. Analyze the code in the **app.py** file to determine where you can change the score value. Then, update the code.
3. After you have updated the code and saved it, deploy the Lambda function using **sam deploy**.

**Note** The **sam deploy** process should be deployed from the backend folder and it will only deploy the updated **comprehend\_function**.

1. Check the updated function in the Lambda console after the deployment is successful.
2. Return to the multi-notification application and post a message to fail the validation.
3. Return to the Step Functions browser tab and check if the workflow has succeeded or not.
4. Check the score of the message validity in the workflow or the CloudWatch logs of the Lambda function.

Review the [Solution](https://labs.skillbuilder.aws/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A470679935125%3Ablueprintversion%2FSPL-TF-300-SVMLTP-1%3A1.0.8-76f4280d/en-US#solution-1) if you have any problems.

**Conclusion**

**Congratulations!** You now have successfully:

* Deployed a serverless backend application using the AWS SAM
* Set up an Amazon Pinpoint project using the console
* Configured data in a DynamoDB table
* Deployed the frontend application using Amplify
* Posted a message using the UI and receive the message in different languages and different formats

**End lab**

Follow these steps to close the console and end your lab.

1. Return to the **AWS Management Console**.
2. At the upper-right corner of the page, choose **AWSLabsUser**, and then choose **Sign out**.
3. Choose **End lab** and then confirm that you want to end your lab. For more information about AWS Training and Certification, see [*https://aws.amazon.com/training/*](https://aws.amazon.com/training/).

*Your feedback is welcome and appreciated.*  
If you would like to share any feedback, suggestions, or corrections, please provide the details in our [*AWS Training and Certification Contact Form*](https://support.aws.amazon.com/#/contacts/aws-training).

**Additional resources**

* For more information about the Vue JavaScript framework, see <https://vuejs.org/>.
* For more information about Step Functions, see <https://docs.aws.amazon.com/step-functions/latest/dg/welcome.html>.
* For more information about Amazon Pinpoint, see <https://aws.amazon.com/pinpoint/features>.
* For more information about Amazon Translate, see <https://aws.amazon.com/translate/details>.
* For more information about Amazon Comprehend, see <https://aws.amazon.com/comprehend/features>.

**Solution**

1. Update the **neutral\_score** value in the if statement from 0.5 to a higher value or lower value to test the validation.

#If the score is less than 50%, message will go to review for support manager approval.

fail\_msg = ''

if neutral\_score < 0.5 :

pass\_flag = False

fail\_msg = '\*\*Original message:\*\* \n"' + text + '"\n\n This announcement message is not meeting the company standards, please revisit the message and resend it.'