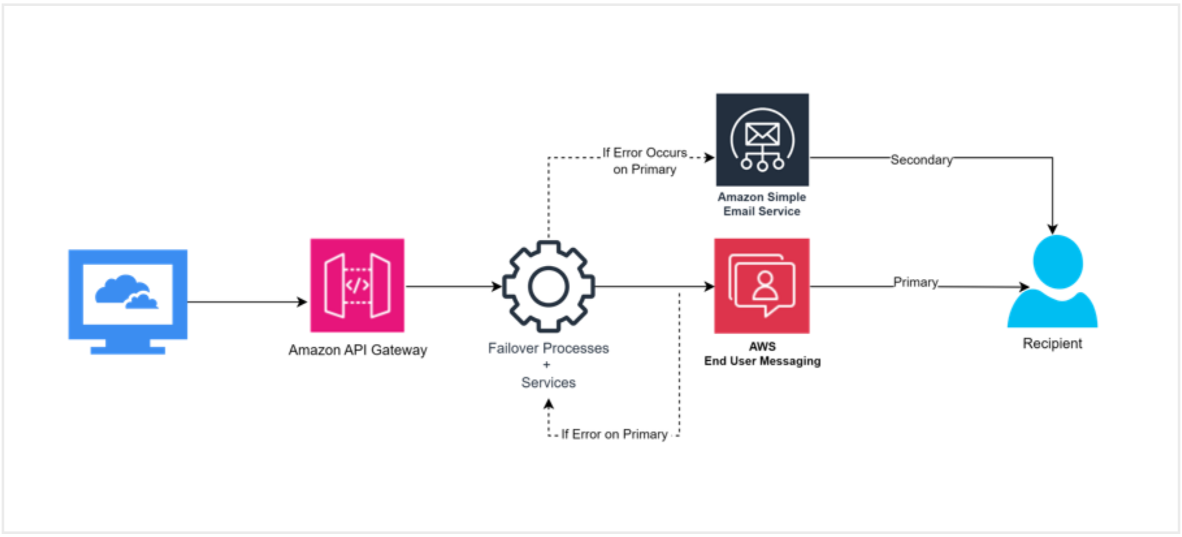
Workshop 1

Implement Resilient Omni-Channel Notifications with AWS

In today's fast-paced business environment, it's critical to have reliable and timely communication channels to reach your customers. This workshop will guide you through the process of building a resilient omni-channel notification system that leverages both primary and fallback communication channels to ensure your time-sensitive messages are delivered. For this workshop you will use SMS and Email channels, but this solution could be extended to other communication channels like the social messaging app WhatsApp.



In this workshop, you will learn how to use AWS services including Amazon API Gateway, AWS Lambda, Amazon SNS, Amazon SQS, Amazon DynamoDB, Amazon Simple Email Service (SES)

, and AWS End User Messaging

to create a solution that monitors message delivery and automatically triggers a secondary channel if the primary channel fails within a specified time. This fallback mechanism helps ensure your important notifications, such as account updates, order confirmations, or security alerts, reach your customers without interruption.
By the end of this workshop, you will have the knowledge and hands-on experience to deploy your own customizable omni-channel communication solution that provides redundancy and improves customer engagement.

In July 2024, AWS announced a new name, AWS End User Messaging, for Amazon Pinpoint's SMS, MMS, push, and text to voice messaging capabilities. The introduction post is here

.

Use Case Context

AnyCompany Co. is a leading online service provider that prioritizes user security and seamless authentication. To ensure the safety of their customers' accounts, they implement a two-factor authentication system that sends one-time passwords (OTPs) to users when they attempt to log in to the company's website. These time-sensitive notifications are crucial for completing the authentication process and must be delivered quickly to maintain an optimal user experience. However, AnyCompany Co. faces a challenge: if these OTPs don't reach users promptly, it can lead to frustration, failed login attempts, and potential loss of business. To address this, AnyCompany Co. is exploring a resilient omni-channel notification system that can ensure timely delivery of these critical messages, even when the primary communication channel encounters issues.
Let's build!

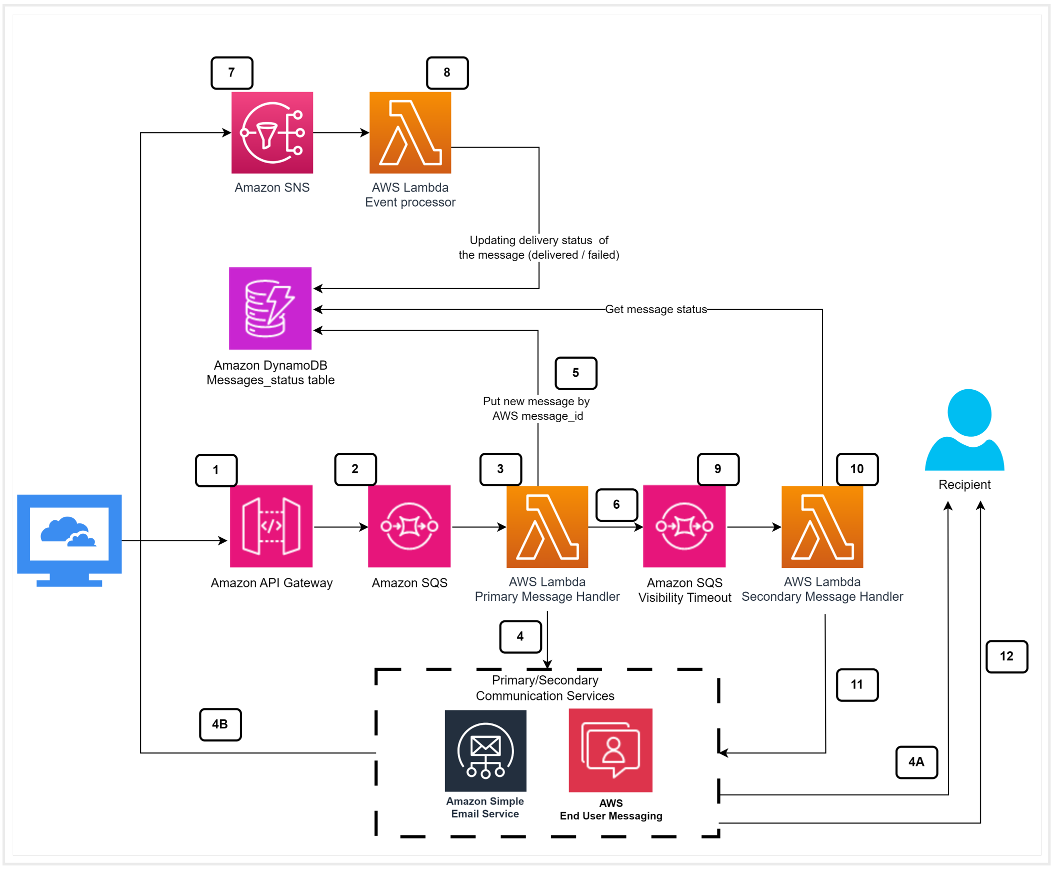
Introduction

This workshop contains an Omni-channel Fallback solution that uses Amazon API Gateway, AWS Lambda, Amazon Simple Email Service (SES), and AWS End User Messaging. The key points about this solution are:

1. Users can send email or SMS messages as the primary and secondary channels using an API Gateway endpoint.
2. The payload specifies the primary and secondary channel, sender, receiver addresses, message content, and the fallback time (in seconds) before the solution sends the message using the secondary channel.
3. The solution supports two use cases:
    - Fallback: Send the message using the primary channel first, and if there is no successful delivery event after the specified fallback time, it will send the message using the fallback channel.
    - Broadcast: Send the message from both the primary and fallback channels at the same time.
4. The solution can create and manage the necessary Amazon SES and AWS End User Messaging SMS configurations to track message delivery events and trigger the fallback mechanism.
5. The configuration options allow customizing the names of the SES and SMS configuration sets.

In summary, this is a flexible omni-channel messaging solution that provides fallback capabilities to ensure message delivery using alternative channels if the primary channel fails to deliver in time.

Detailed Solution Architecture



Target Audience

This workshop is designed for developers, email and IT infrastructure administrators who want to learn how to set up, manage and optimize their communications with Amazon Simple Email Service and AWS End User Messaging.
This workshop assumes you have a basic understanding of AWS services and their related terminologies. Prior experience with Amazon Web Services (AWS) Command Line Interface (CLI) and knowledge of Simple Mail Transfer Protocol (SMTP) and email protocols will be advantageous. While the workshop is designed to be comprehensive and beginner-friendly, having a basic background in these areas will enable you to make the most out of this learning experience.
You will use the Amazon SES API v2 (sesv2) and the AWS End User Messaging API (pinpoint-sms-voice-v2) in this workshop.

Note
You will use the AWS CLI for most of the exercises in this workshop. If you are more comfortable using the AWS console, most (but not all) of the exercises can be completed using the SES and End User Messaging consoles, however instructions for navigating via the console are not provided herein.

Duration

The workshop is designed to be completed in 2 hours.

Cost Considerations

If you are participating in an AWS Workshop event, there are no cost

considerations and you may skip to the next section.
While the workshop itself is free of charge, please note that implementing this solution in your own account may incur costs associated with the AWS services used, including Amazon API Gateway, Amazon SES, Amazon SNS, Amazon SQS, Amazon DynamoDB, AWS Lambda, and AWS End User Messaging. To understand the potential costs, we recommend reviewing the pricing pages for each service: Amazon API Gateway pricing

, Amazon Simple Email Service pricing

, Amazon SNS pricing

, Amazon SQS pricing

, Amazon DynamoDB pricing

, AWS Lambda pricing

, and AWS End User Messaging pricing

.
Additionally, be aware that sending SMS messages may incur charges from third-party providers. We encourage you to estimate your usage and calculate potential costs before deploying this solution in a production environment.

Lab 1:

Lab 1: Email Identities

1. Create Email Identities

In Amazon SES, a verified identity is a domain or email address that you use to send or receive email. Before you can send an email using Amazon SES, you must create and verify each identity that you're going to use as a "From", "Source", "Sender", or "Return-Path" address. Verifying an identity with Amazon SES confirms that you own it and helps prevent unauthorized use.
If your account is still in the Amazon SES sandbox, you also need to verify any email addresses which you plan on sending email to, unless you're sending to test inboxes provided by the Amazon SES mailbox simulator

.
You can create an identity by using the Amazon SES console, the Amazon SES API, or via the AWS CLI. The identity verification process depends on which type of identity you choose to create.

Email Identity and Domain Identity

- Domain Identity
  Email Identity

Recommended - A domain identity represents an entire domain (or sub-domain). When you verify a domain identity, you can use any email address from that domain (or sub-domain) as the sender address for your emails. In addition, you can set up important email authentication methods like DKIM

, SPF

and DMARC

for the domain, improving your deliverability and mailing reputation. We recommend using a verified domain identity in production because this method fully supports industry best practices, especially sender authentication. Refer to the domain identity documentation

for more details.
You'll need to have a verified identity to be able to send emails through Amazon SES. For this next part of the lab, you will verify your own email address, which will act as the sender address for your notification and the destination. You can choose to verify more than one address by repeating the process with different email addresses or using aliases in the form of example+sender@example.com and example+recipient@example.com.

Note
The workshop accounts are in Amazon SES sandbox. You'd need to also verify the recipient address because sandbox environment will only allow sending to verified email addresses.
While your account is in the sandbox, you can use all of the features of Amazon SES. However, when your account is in the sandbox, we apply the following restrictions to your account:

- You can only send mail to verified email addresses and domains, or to the Amazon SES mailbox simulator
.
- You can send a maximum of 200 messages per 24-hour period.
- You can send a maximum of 1 message per second.

Steps

1. Open the AWS console, and open AWS CloudShell terminal (check About the workshop if you need help)
2. Copy and paste the AWS CLI commands shown below into CloudShell:

```
aws sesv2 create-email-identity --email-identity example@example.com
```

Replace example@example.com with the email address you want to verify, which will be used as the "From" (source) and "To" (destination) addresses for your notification.

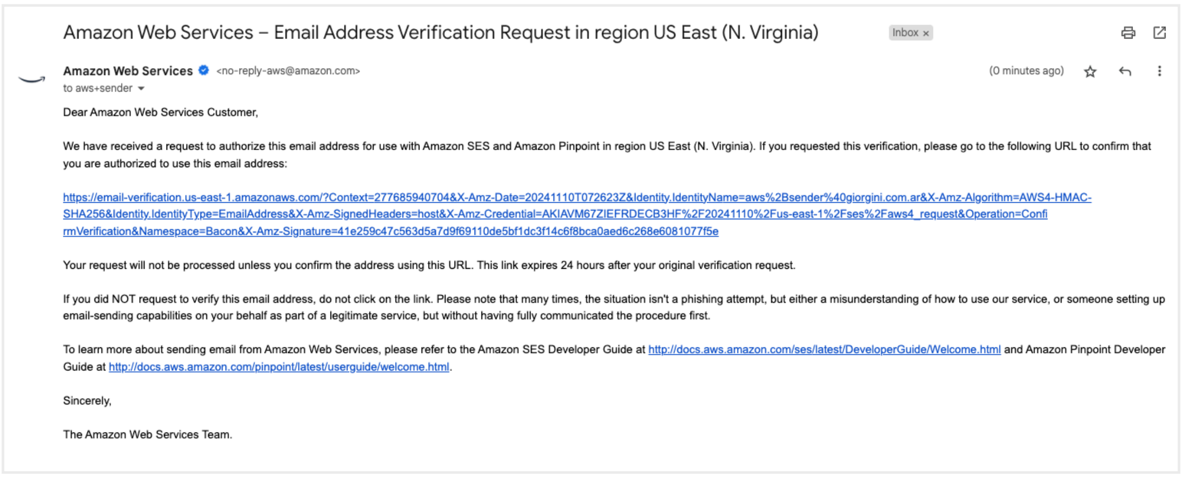
2. Verify Email Identities

Once you create your email address identity in Amazon SES, you will receive an email with a verification link.

Steps

1. Click the link in the received email to complete the verification

process.



Amazon Web Services – Email Address Verification Request in region US East (N. Virginia)

Amazon Web Services <no-reply-aws@amazon.com>
to aws+sender

Dear Amazon Web Services Customer,

We have received a request to authorize this email address for use with Amazon SES and Amazon Pinpoint in region US East (N. Virginia). If you requested this verification, please go to the following URL to confirm that you are authorized to use this email address:

https://email-verification.us-east-1.amazonaws.com/?Context=277685940704&X-Amz-Date=20241110T072623Z&Identity.IdentityName=aws%2Bsender%40giorgini.com.ar&X-Amz-Algorithm=AWS4-HMAC-SHA256&Identity.IdentityType=EmailAddress&X-Amz-SignedHeaders=host&X-Amz-Credential=AKIAVM67ZIEFRDECB3HF%2F20241110%2Fus-east-1%2Fses%2Faws4_request&Operation=ConfirmVerification&Namespace=Bacon&X-Amz-Signature=41e259c47c563d5a7d9f69110de5bf1dc3f14c6f8bca0aed6c268e6081077f5e

Your request will not be processed unless you confirm the address using this URL. This link expires 24 hours after your original verification request.

If you did NOT request to verify this email address, do not click on the link. Please note that many times, the situation isn't a phishing attempt, but either a misunderstanding of how to use our service, or someone setting up email-sending capabilities on your behalf as part of a legitimate service, but without having fully communicated the procedure first.

To learn more about sending email from Amazon Web Services, please refer to the Amazon SES Developer Guide at http://docs.aws.amazon.com/ses/latest/DeveloperGuide/Welcome.html and Amazon Pinpoint Developer Guide at http://docs.aws.amazon.com/pinpoint/latest/userguide/welcome.html.

Sincerely,

The Amazon Web Services Team.

2.

3. To check the verification status of an email address, enter the following command:

```
aws sesv2 get-email-identity --email-identity example@example.com
```

This will return a JSON object with the verification status of the specified email address, similar to below:

```
{
  "IdentityType": "EMAIL_ADDRESS",
  "FeedbackForwardingStatus": true,
  "VerifiedForSendingStatus": true,
  "DkimAttributes": {
    "SigningEnabled": false,
    "Status": "NOT_STARTED",
    "SigningAttributesOrigin": "AWS_SES",
    "NextSigningKeyLength": "RSA_1024_BIT"
  },
  "MailFromAttributes": {
    "BehaviorOnMxFailure": "USE_DEFAULT_VALUE"
  },
  "Policies": {},
  "Tags": [],
  "VerificationStatus": "SUCCESS"
}
```

3. Send Test email

Let's confirm your addresses are verified for sending and receiving with a simple test email.

Steps

1. Replace sender@example.com and recipient@example.com with the email addresses you verified

```
aws sesv2 send-email \
    --content "Simple={Subject={Data=Test Email,Charset=UTF-8},Body={Text={Data=This is a test email sent from Amazon SES using the AWS CLI,Charset=UTF-8}}}" \
    --from-email-address "sender@example.com" \
    --destination "ToAddresses=recipient@example.com"
```

Lab 2:

Lab 2: SMS Origination Identity

1. Create a Phone Pool

A phone pool, is a collection of phone numbers or sender IDs that you can use to send messages using the same settings. When you send messages through a phone pool, AWS End User Messaging chooses an appropriate origination identity from which to send messages. If an origination identity in the phone pool fails, the pool will fail over to another origination identity in the same pool.
When you create a phone pool, you can configure a specified origination identity. This identity includes keywords, message type, opt-out list, two-way configuration, and self-managed opt-out configuration. For example, using pools you can associate a list of opted-out destination phone numbers with your phone number for a particular country. By doing so, you can prevent messages from being sent to users who have already opted out of receiving messages from you.
In this lab, you will create a phone pool for sending SMS. To create a phone pool, you need to assign at least one originating identity to the pool.
In this workshop, you will be using an alphanumeric Sender ID as an example for our SMS messaging. However, it's important to note that this is just for demonstration purposes. In practice, the use of Sender IDs may be subject to specific regulations in different countries.
For instance, as of September 2023, the United Kingdom has implemented a requirement for all Sender IDs to be registered before use. This regulation aims

to reduce spam and protect consumers. Other countries may have similar or different requirements, so it's crucial to always check and comply with the local regulations in the countries where you plan to send SMS messages.
When implementing SMS messaging in a real-world scenario, make sure to research and adhere to the most current regulations in your target regions to ensure compliance and maintain good deliverability rates.Your AWS End User Messaging SMS will operate in the sandbox mode, limited throughput, however this won't negatively affect the workshop.

Steps

In the AWS console, navigate to the AWS End User Messaging SMS service, and open AWS CloudShell terminal:

1. Copy and paste the CLI commands shown below into CloudShell to create a phone pool
   for AnyCompany's transactional messages:

```
POOL_ID=$(aws pinpoint-sms-voice-v2 create-pool \
--origination-identity AnyCompany \
--iso-country-code GB \
--message-type TRANSACTIONAL \
--output text \
--query PoolId) && echo $POOL_ID
```

You might get a warning in CloudShell for pasting multiline text. Remember to disable the option to Ask before pasting multiline code to avoid being prompted each time.

2. Verify Phone Pool creation

The CLI commands you executed above created a phone pool and a system variable named POOL_ID used to store the pool_id in the CLI. Copy the PoolId value and paste it into a local text file (disable text formatting) so you'll have your PoolId to use later in this workshop.

Steps

1. To see a full description of the phone pool you just created, execute the AWS CLI command below:

```
aws pinpoint-sms-voice-v2 describe-pools \
--pool-ids $POOL_ID \
--query 'Pools[0]'
```

The response should look like this:

```
1
2
3
4
5
6
7
```

```
{
   "PoolArn": "arn:aws:sms-voice:<AWSRegion>:<AWSaccountID>:pool/<pool-id>",
   "PoolId": <value>,
   "Status": "ACTIVE",
   "MessageType": "TRANSACTIONAL",
   "TwoWayEnabled": false,
   "SelfManagedOptOutsEnabled": false,
   "OptOutListName": "Default",
   "SharedRoutesEnabled": false,
   "DeletionProtectionEnabled": false,
   "CreatedTimestamp": <value>
}
```

3. Send SMS test

Now, let's test the SMS sending capability using the AWS End User Messaging CLI. You'll use a simulator phone number that simulates a successfully delivered message, avoiding to sending actual messages.

Steps

1. In your terminal and enter the following command:
2. aws pinpoint-sms-voice-v2 send-text-message \
3.   --destination-phone-number "+447860019066" \
4.   --message-body "This is a test SMS message sent using AWS End User Messaging CLI" \
5.   --origination-identity "ANYCOMPANY"

1. This command sends a test message to the UK simulator number (+447860019066) that simulates a successfully delivered message. Remember, you're using this Sender ID as an example, but in practice, you must comply with local regulations. After running the command, check the CLI output to confirm the message was sent successfully to the simulator.

To move out of the sandbox and send messages to real phone numbers, you would need to request production access from AWS. This typically involves demonstrating legitimate use cases and compliance with AWS policies and local regulations.

In the next lab, you'll learn how to Deploy the solution using AWS CDK through AWS CloudShell.

Lab 3:

Lab 3: Deploy Solution

Deploy the solution using AWS CDK through AWS CloudShell

In this lab, you will deploy the solution using AWS CDK project from a GitHub repository using AWS CloudShell. This guide walks you through every step required, from cloning the repository to configuring the parameters for deployment.

Steps

1. Clone the GitHub Repository

To clone the sample repository containing the CDK project, in the CloudShell terminal, run the following command to clone the repository:

```
git clone https://github.com/aws-samples/omnichannel-fallback-messaging
```

This command downloads the project files from GitHub into your CloudShell environment.

Change into the project directory:

```
cd omnichannel-fallback-messaging
```

2. Review and Update Configuration File

The config.params.json file defines parameters required for the CDK project to configure services like SQS, SES, and SMS for the fallback messaging system. Let’s walk through its contents:

```
{
  "CdkBackendStack": "FallbackMessagingService",
  "SqsVisibilityTimeout": 300,
  "sesConfigSetName": "ses-config-set",
  "createSESConfigSet": "true",
  "smsConfigSetName": "sms-config-set",
  "createSMSConfigSet": "true",
  "tags": {
    "Application": "MyApp",
    "Environment": "Dev",
    "Owner": "YourName",
    "Project": "FallbackMessagingService"
  }
}
```

Explanation of Fields

Update the timeout to 30 seconds

Execute the following command to replace the SqsVisibilityTimeout from the original value of 300 to 30 seconds to test this faster in our workshop.

```
sed -i 's/"SqsVisibilityTimeout": 300/"SqsVisibilityTimeout": 30/'
config.params.json
```

3. Update AWS CDK and Install Project Dependencies

AWS CloudShell's CDK version is not always up to date with the latest version, you need to update the CDK version before running other commands

Info

The following steps may take a moment and the terminal may seem unresponsive while they process.

Install the latest version of AWS CDK locally:

```
sudo npm install -g aws-cdk@latest
```

Now that the configuration file is ready, install the project's dependencies and bootstrap CDK.

Run the following command to install all necessary packages:

```
npm install && cdk bootstrap --output build/cdk.out
```

This installs the libraries required by the CDK project and bootstraps the environment. This step will take a couple of minutes, but the output should look like this once complete:

✅ Environment aws://123456789012/us-east-1 bootstrapped.

More details and screenshot

4. Deploy the CDK Stack

Once the environment is bootstrapped, deploy the CDK project. You will store the results of the deployment in a file named .deploy-log to retrieve the API values needed later.

Deploy the stack:
cdk deploy 2>&1 | tee -a .deploy-log
API_KEY=$(grep ApiKeyValueOutput .deploy-log | awk '{print $3}' | tr -d '"')
API_URL=$(grep ApiUrl .deploy-log | awk '{print $3}' | tr -d '"')

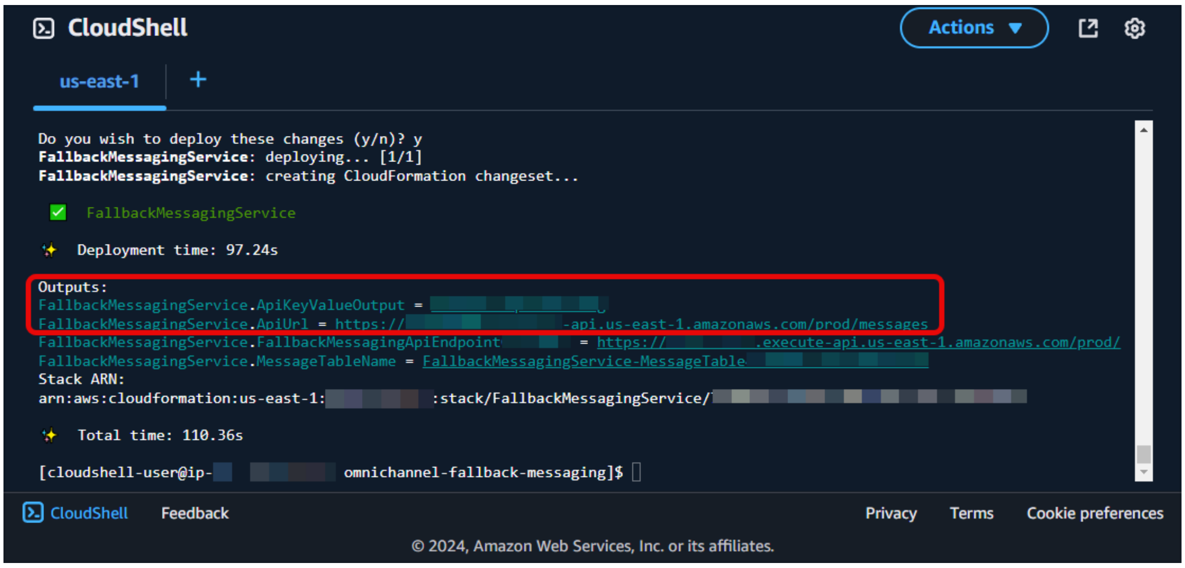
2. The project is synthesized, which means CDK is preparing the resources that will be deployed. Once the synthesis is complete, you are prompted to confirm the deployment. Type y and press Enter to proceed.

5. CDK Stack outputs

After the deployment completes, you will see the CDK stack outputs. You stored these values in the file .deploy-log:

- API Key Value: <API-key-value>
  This is the value of the API key that you will use to authenticate requests to the API. Make sure to save this key securely, as you will need it to make API calls.
- API URL: <API-url>/prod/messages
  This is the endpoint URL for the API Gateway. Save this URL as it will be used for making requests to the /messages endpoint.
- Base API Endpoint: <API-base-url>/prod/
  This is the base URL of the API. You can use it to navigate to different endpoints within the service.
- DynamoDB Table Name: <MessageTableName>
  This is the name of the DynamoDB table created by the CDK stack, where the messages' status is being stored.

Screenshot



```
Do you wish to deploy these changes (y/n)? y
FallbackMessagingService: deploying... [1/1]
FallbackMessagingService: creating CloudFormation changeset...

 ✅  FallbackMessagingService

✨  Deployment time: 97.24s

Outputs:
FallbackMessagingService.ApiKeyValueOutput = [redacted]
FallbackMessagingService.ApiUrl = https://[redacted]-api.us-east-1.amazonaws.com/prod/messages
FallbackMessagingService.FallbackMessagingApiEndpoint[redacted] = https://[redacted].execute-api.us-east-1.amazonaws.com/prod/
FallbackMessagingService.MessageTableName = FallbackMessagingService-MessageTable[redacted]
Stack ARN:
arn:aws:cloudformation:us-east-1:[redacted]:stack/FallbackMessagingService/[redacted]

✨  Total time: 110.36s

[cloudshell-user@ip-[redacted] omnichannel-fallback-messaging]$
```

Lab 4:

Lab 4: Omni-channel Fallback in Action

Test the Omni-channel Fallback in Action

In this lab, you will intentionally cause the primary SMS delivery to fail and observe how the solution automatically implements the fallback mechanism by sending the notification via email.

Obtain the API Gateway endpoint URL from the CDK deployment output

1. Explanation of the API Gateway:
You will be interacting with the solution deployed through Amazon API Gateway. API Gateway is like a front door for applications to access data, business logic, or functionality from your backend services. Here, it's the entry point for sending messages through AnyCompany's notification system.
2. Explanation of the API Key Value:
This is the value used to authenticate requests to the API.
3. Explanation of the endpoint URL:
In order to send messages, identify the URL of the API Gateway endpoint.
4. How to find the API Key Value and URL:
The API key value and URL were each provided as outputs when you deployed the CFK solution at the end of the previous Lab. You saved them in a local file .deploy-log and store the values in 2 variables API_KEY and API_URL. Scroll up in your CloudShell terminal, to find the section called "Outputs:". You'll find the outputs FallbackMessagingService.ApiKeyValueOutput and FallbackMessagingService.ApiUrl.

```
echo "API_KEY =" $API_KEY
echo "API_URL =" $API_URL
```

Fill in the placeholders in the payload with their own values

1. Explanation of the payload:
You need to provide information to the notification system about the message you want to send. This information is included in a "payload." The payload is formatted as a JSON object, which is a way to structure data using key-value pairs. Think of it as an envelope containing all the necessary details for your message delivery.

```
{
  "use_case": "fallback",
  "fallback_seconds": "30",
  "pc": {
     "channel": "sms",
     "sender": "ANYCOMPANY",
     "recipient": "%2B447860019067",
     "sms": {
        "message": "Your account has been updated!",
        "message_type": "TRANSACTIONAL",
        "configuration_set": "sms-config-set"
           }
   },
   "fc": {
     "channel": "email",
      "sender": "sender@example.com",
      "recipient": "recipient@example.com",
      "email": {
        "subject": "Important Account Notification",
        "text": "Your account has been updated!",
        "html": "<p>Your account has been updated!</p>",
        "configuration_set": "ses-config-set"
      }
    }
  }
```

3. How to fill the placeholders:
Now you will personalize this JSON payload with your specific information. Here's how you can do that within CloudShell:

- Open a Text Editor: CloudShell provides a built-in text editor called nano. Let's use it to edit your payload. Type the following command in your terminal:

```
nano fallback_payload.json
```

Summary:

Summary

In this workshop, you have been introduced to the Omni-channel Fallback Messaging solution and learned how to implement a reliable messaging system using multiple channels. Through a series of hands-on exercises, you set up AWS components, configured message delivery options, and explored fallback mechanisms. Key accomplishments include:

- Setting up an API Gateway endpoint for sending messages via Email and SMS.
- Configuring primary and fallback channels for message delivery.
- Implementing asynchronous message processing using Amazon SQS.
- Utilizing AWS Lambda functions for primary and secondary message handling.
- Leveraging Amazon SNS for tracking message events across different channels.
- Using Amazon DynamoDB to store message status and ID mappings.

To get started with the Omnichannel Fallback Messaging solution, consider the following steps:

- Ensure you have the necessary prerequisites, such as verified SES sending identities.
- Follow the deployment guide to set up the solution components.
- Familiarize yourself with the API request body structure and available options.
- Configure SES and SMS configuration sets as needed for event monitoring.
- Test the solution using various scenarios, including primary channel

failures and fallbacks.
- Monitor message delivery status and events using the provided DynamoDB tables and SNS topics.

For further exploration, consider customizing the solution to fit your specific use cases, integrating it with other AWS services, or expanding its capabilities to support additional messaging channels.