Building Serverless Applications with an Event-Driven Architecture

**spl-254 - Version 1.0.18**

© 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**

Play Video

Serverless is the native architecture of the cloud. You can build serverless architectures for nearly any type of application or backend service—without thinking about servers. This type of architecture eliminates infrastructure management tasks such as server or cluster provisioning, patching, operating system maintenance, and capacity provisioning. Everything required to run and scale your application with high availability is handled for you.

Serverless enables you to shift more of your operational responsibilities to Amazon Web Services (AWS), increasing your agility and innovation and lowering your total cost of ownership. Your developers can focus on their core product instead of worrying about managing and operating servers or runtimes, either in the cloud or on premises. This reduced overhead lets developers reclaim time and energy that can be spent on developing great products that scale and are reliable.

In this lab, you build a web-based book printing application using a set of serverless technologies including Amazon API Gateway, AWS Step Functions, AWS Lambda, Amazon Simple Storage Service (Amazon S3), Amazon DynamoDB, Amazon Simple Notification Service (Amazon SNS), Amazon Simple Queue Service (Amazon SQS), and Amazon Rekognition.

The application allows customers to upload a collection of images that they want printed in a physical book. Each image is processed to ensure that it is displayed properly in the book. The application uses Step Functions to manage the workflow. The workflow uses Lambda functions to make sure that each image is the proper file type and uses Amazon Rekognition to ensure that the content is appropriate. The workflow also resizes and watermarks the images and then generates a PDF proof for the customer to approve. Next, Amazon SNS sends an email to the customer for approval before sending the job to Amazon SQS. Amazon SQS sends the book to the third-party printing service to be printed and shipped to the customer.

OBJECTIVES

After completing this lab, you should be able to:

* Understand an event-driven architecture.
* Understand how Step Functions is configured to orchestrate serverless applications.
* Take advantage of Amazon SQS and Amazon SNS.
* Create and configure Lambda functions and API Gateway resources.
* Made configuration updates to restore API functionality.

ICON KEY

Various icons are used throughout this lab to call attention to different types of instructions and notes. The following list explains the purpose for each icon:

* **Note:** A hint, tip, or important guidance.
* **Command:** A command that you must run.
* **Expected output:** A sample output that you can use to verify the output of a command or edited file.
* **Task complete:** A conclusion or summary point in the lab.
* **Learn more:** Where to find more information.

**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: You must first sign out**



If you see the message, **You must first log out before logging into a different AWS account:**

* Choose the **click here** link.
* Close your **Amazon Web Services Sign In** web browser tab and return to your initial lab page.
* Choose **Open Console** again.

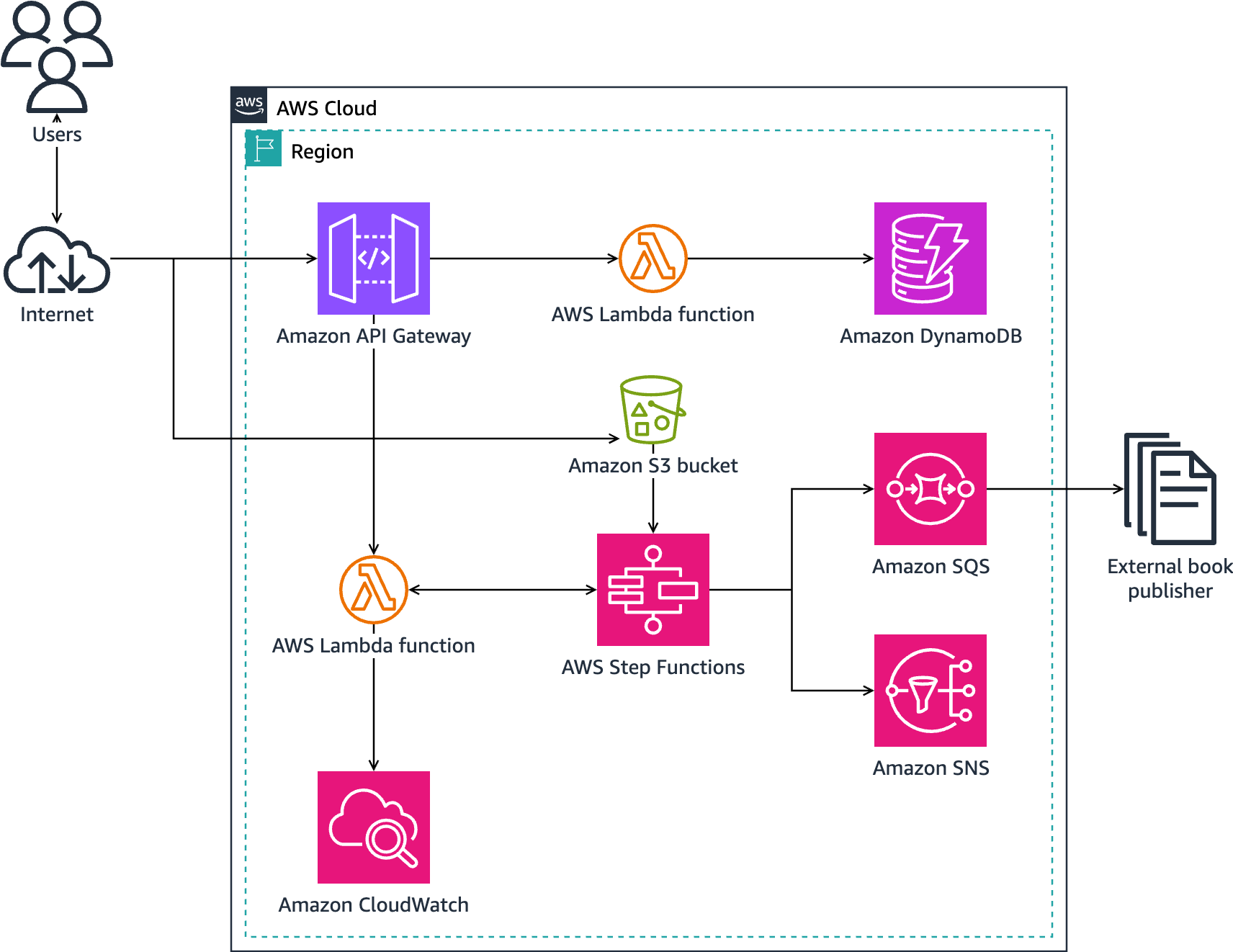
**Error: Choosing Start Lab has no effect**

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.

LAB ENVIRONMENT

The following diagram shows the basic architecture of the photo book printing application used in the lab:



*Image description: The preceding image depicts the AWS services used in this lab, as well as the overall workflow from the end user through to a third party printing service. The lab environment uses API Gateway, Lambda functions, Step Functions, Amazon S3 buckets, DynamoDB tables, Amazon SQS queues, Amazon SNS topics and Amazon Rekognition.*

API Gateway hosts several APIs in front of several Lambda functions. DynamoDB is used to store metadata about the images a user uploads. A Step Functions workflow is started to validate the images and render them for printing, before sending a link (via Amazon SNS) to the user for their verification. A user can preview a PDF of the book before sending it to a third party for printing and shipping.

The application uses the following APIs:

1. **/presigned:** The user sends a request for presigned URLs to upload their images to an Amazon S3 bucket. The user uses the presigned URLs to upload their images directly to Amazon S3. You create this API later in the lab.
2. **/bookprint:** The user starts the book creation process by indicating they have completed uploading images. This invokes the image-processing state machine. This API is already created in the lab build.
3. **/invocation:** The user approves the PDF by acknowledging an email from Amazon SNS. This API is already created in the lab build.

The application uses the following state machines:

* **ImageProcessStateMachine:** The job of this state machine is to pick up the user-uploaded images from the Amazon S3 */Incoming* folder, process the images, and create a PDF album for book printing. In this flow, a few key steps are image validation, resizing, and watermarking. This state machine is already created in the lab build.
* **BookprintStateMachine:** This state machine reads the messages from the print vendor Amazon SQS queue and sends the photo book to a third-party vendor for printing. You create this state machine later in the lab.

SERVICES USED IN THIS LAB

**Amazon API Gateway**

Amazon API Gateway is a fully managed service that makes it easy for developers 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.

**AWS Lambda**

AWS Lambda lets you run code without provisioning or managing servers. You pay only for the compute time you consume. With Lambda, you can run code for virtually any type of application or backend service—all with zero administration. Just upload your code, and Lambda takes care of everything required to run and scale your code with high availability. You can set up your code to automatically be invoked from other AWS services or call it directly from any web or mobile app.

**AWS Step Functions**

AWS Step Functions lets you coordinate multiple AWS services into serverless workflows so you can build and update apps quickly. Using Step Functions, you can design and run workflows that stitch together services, such as AWS Lambda, AWS Fargate, and Amazon SageMaker, into feature-rich applications.

**Amazon Simple Storage Service (Amazon S3)**

Amazon S3 is an object storage service that offers industry-leading scalability, data availability, security, and performance. This means customers of all sizes and industries can use it to store and protect any amount of data for a range of use cases, such as websites, mobile applications, backup and restore, archive, enterprise applications, Internet of Things (IoT) devices, and big data analytics. Amazon S3 provides easy-to-use management features so you can organize your data and configure finely tuned access controls to meet your specific business, organizational, and compliance requirements. Amazon S3 is designed for 99.999999999% (11 9s) of durability and stores data for millions of applications for companies all around the world.

**Amazon Simple Queue Service (Amazon SQS)**

Amazon SQS is a fully managed message queuing service that enables you to decouple and scale microservices, distributed systems, and serverless applications. Amazon SQS eliminates the complexity and overhead associated with managing and operating message-oriented middleware and 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 Simple Notification Service (Amazon SNS)**

Amazon SNS is a highly available, durable, secure, fully managed pub/sub messaging service that enables you to decouple microservices, distributed systems, and serverless applications. Amazon SNS provides topics for high-throughput, push-based, many-to-many messaging.

**Amazon Rekognition**

Amazon Rekognition makes it easy to add image and video analysis to your applications using proven, highly scalable, deep learning technology that requires no machine learning expertise to use. With Amazon Rekognition, you can identify objects, people, text, scenes, and activities in images and videos, as well as detect any inappropriate content.

AWS SERVICES NOT USED IN THIS LAB

The lab environment adheres to the [principle of least-privilege permissions](https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html#grant-least-privilege). As such, the actions you can perform are limited to only those that are required to complete the lab tasks and learning objectives. Access to AWS services that are not needed to complete the lab tasks is restricted. Expect errors if you attempt to perform actions beyond those provided in the lab guide.

**Task 1: Setting up a Lambda function to generate presigned URLs**

In this task, you create a Lambda function that generates presigned URLs, then captures user-provided album metadata and stores it to DynamoDB.

**Note:** This application requires several Lambda functions. Most of these were created for you during the lab provisioning process.

1. At the top of the AWS Management Console, in the search bar, search for and choose

Lambda

.

1. Choose **Create function**.
2. On the **Create function** page, configure the following:

* Choose **Author from scratch**.
* For **Function name**, enter

PresignedUrlFunction

.

* For **Runtime**, choose **Python 3.12**.

1. Expand **Change default execution role**, and then configure the following:

* For **Execution role**, choose **Use an existing role**.
* For **Existing role**, choose **DigitalBooks/BackendProcessingLambdaRole**.

1. Choose **Create function**.
2. In the **Code source** section, choose **Upload from**, then choose **Amazon S3 location**.
3. In the **Upload a file from Amazon S3** popup window, configure the following:

* For **Amazon S3 link URL**, copy and paste the **PresignedLambdaSourceCode** value that is listed to the left of these instructions.
* Choose **Save**.

1. Choose the **Configuration** tab.
2. In the configuration navigation pane, choose **Environment variables**.
3. Choose **Edit**.
4. To configure the first environment variable, on the **Edit environment variables** page, choose **Add environment variable**, and then configure the following:

* For **Key**, enter

imageMetadataTableName

.

* For **Value**, copy and paste the **ImageMetadataTable** value that is listed to the left of these instructions.

1. To configure a second environment variable, choose **Add environment variable**, and then configure the following:

* For **Key**, enter

s3BucketName

.

* For **Value**, copy and paste the **UserImagesS3Bucket** value that is listed to the left of these instructions.

1. Choose **Save**.
2. Choose **General configuration**.
3. Choose **Edit**.
4. On the **Edit basic settings** page, configure the following:

* For **Memory**, enter

512

 MB.

* **Timeout**, enter

10

 sec.

* Choose **Save**.

**Task complete:** You have successfully created the Lambda function to generate presigned URLs and create the album metadata.

**Task 2: Creating a presigned API resource**

All Amazon S3 objects and buckets are private by default. Presigned URLs are useful if you want a user to be able to upload a specific object to your bucket, but you don’t want to require them to have AWS security credentials or permissions. When you create a presigned URL, you must provide your security credentials and then specify a bucket name, object key, HTTP method (PUT for uploading objects), and expiration date and time. Presigned URLs are valid only for the specified duration.

In this task, you create a new API resource on an existing API Gateway. This API endpoint generates presigned URLs to upload the images directly to the Amazon S3 bucket for further processing.

**Note:** The API Gateway endpoint and other API resources were created during the lab provisioning process.

1. At the top of the AWS Management Console, in the search bar, search for and choose

API Gateway

.

1. Choose the link for the API that starts with **DigitalBookPrintsAPI-**.
2. On the **Resources** page, choose **Create resource**.
3. On the **Create resource** page, configure the following:

* For **Resource Path**, leave it as

/

 (it is auto populated with **/presigned** after creation).

* For **Resource Name**, enter

presigned

.

1. Choose **Create resource**.
2. On the **Resources** panel, choose the **/presigned** resource.
3. In the **Methods** section, choose **Create method**.
4. On the **Create method** page, in the **Method details** section, configure the following:

* For **Method type**, choose **POST**.
* For **Integration type**, choose **Lambda function**.
* For **Lambda Proxy integration**, set the toggle to the **on** position.
* For **Lambda Region**, select the region that matches the **AwsRegionCode** value that is listed to the left of these instructions.
* For **Lambda Function**, search for and choose the

PresignedUrlFunction

 function.

* For **Request body**, choose **Add model** and then enter

application/json

 in Content type.

1. Choose **Create method**.
2. At the upper-right corner of the page, choose **Deploy API**.
3. In the **Deploy API** pop-up window, for **Stage**, choose **stage**.
4. Choose **Deploy**.
5. On the **Stages** page, expand **stage**, **/**, **/presigned** and choose **POST**.
6. Copy the **Invoke URL** to a text editor to use in a later task.

**Task complete:** You have successfully created a new API resource on an existing API Gateway and integrated it with a backend Lambda function. Your presigned API and Lambda functions are ready for you to upload a few images.

**Task 3: Uploading a sample image**

In this task, you upload a sample picture to include in the photo book.

1. At the top of the AWS Management Console, in the search bar, search for and choose

Cloud9

.

1. To start a terminal session to the Cloud9 environment, for the **DigitalBooks-xxxx** environment, choose the **Open** link .

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

1. Close the **Welcome** tab.
2. To download sample images, in the AWS Cloud9 terminal, run the following commands:

cd ~/environment

wget https://us-west-2-aws-training.s3.amazonaws.com/courses/spl-254/v1.0.18.prod-3e7dc13d/scripts/sample-images.zip

unzip sample-images.zip

cd sample-images

**Expected output:** A *sample-images.zip* file is downloaded from an S3 bucket and unzipped into the *sample-images* directory.

There is now a *sample-images* directory with several sample images (for example, *sample-image*). In the next steps, you get presigned URLs to upload these images to generate a photo book.

The following command is used to get a specific number of presigned URLs to use when uploading the images:

*curl -d ‘{“userName”:“USER\_NAME”,“albumName”:“ALBUM\_NAME”,“message”:“CUSTOM\_MESSAGE”,“numberOfImages”:“INTEGER\_VALUE”}’ -H “Content-Type: application/json” -X POST INVOKE\_URL | python -m json.tool*

Here is an explanation of the command arguments:

* **USER\_NAME**: The username of customer requesting adding the images to the album.
* **ALBUM\_NAME**: The name of album where images are added.
* **CUSTOM\_MESSAGE**: A description of the album.
* **INTEGER\_VALUE**: The number of presigned URLs requested. This should be equal to the number of images to be uploaded.
* **INVOKE\_URL**: The URL to invoke the presigned API which you created in the previous step.

*When running the command, it invokes the presigned API. The API invokes a backend Lambda function that generates a number of presigned URLs to use when uploading images to an album.*

Throughout the lab, the following arguments are used when running the commands:

* **USER\_NAME**: Jane Doe
* **ALBUM\_NAME**: Hawaii Vacation
* **CUSTOM\_MESSAGE**: Our trip to Hawaii in 2019
* **INTEGER\_VALUE**: 1 (This generates one presigned URL to upload one image)
* **INVOKE\_URL**: This is the **Invoke URL** value you copied at the end of task 2

1. To get a presigned URL to upload an image, run the following command:

* Replace the **INVOKE\_URL** placeholder value with the **Invoke URL** value that you copied at the end of Task 2.

curl -d '{"userName":"Jane Doe","albumName":"Hawaii Vacation","message":"Our trip to Hawaii in 2019","numberOfImages":"1"}' -H "Content-Type: application/json" -X POST INVOKE\_URL | python -m json.tool

**Expected output:** The output includes the file name, bucket name, and a presigned URL for each image in the photo book.

In the next step, you upload one of the sample images to the presigned URL from the output. Because you are in the *sample-images* directory, the path to each image is the file name (for example, *sample-image.jpg*).

**Note:** Here is a list of the other available images which you can upload:

no\_face.jpg

,

backlit-bonding-casual.jpg

,

business-meeting.jpg

,

city-community-crossing

,

pittsburgh.jpg

,

vegas.jpg

,

multiple\_faces.jpg

,

seattle.jpg

,

sunglass\_face.jpg

,

busy-street-crowd-crowded.jpg

, or

victoria.jpg

.

1. To upload an image, run the following command:

* Replace **PATH\_TO\_IMAGE\_NAME** with one of the sample image file names (for example,

sample-image.jpg

).

* Replace **PRESIGN\_URL** with the outputted **presignUrl** from the previous step (keep the quotation marks).

curl -X PUT -T PATH\_TO\_IMAGE\_NAME "PRESIGN\_URL"

Example:

curl -X PUT -T sample-image.jpg "https://xxxxxx-userimagess3bucket-xxxxxx.s3-us-west-2.amazonaws.com/Incoming/Jane\_Doe/Hawaii\_Vacation/Jane\_Doe\_Hawaii\_Vacation\_0.jpg?AWSAccessKeyId=xxxxxxxxxx&Signature=xxxxxxxxxx&x-amz-security-token=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxExpires=xxxxxxxxxx"

**Expected output:** None, unless there is an error.

**Caution:** In this lab, the timeout value is limited for the Lambda functions that are used to process the images. If you attempt to upload more than one image, you will receive an error when you invoke the book binding state machine in a future step.

1. Return to your web browser tab with the AWS Management Console.
2. At the top of the AWS Management Console, in the search bar, search for and choose

S3

.

1. Navigate to the **xxxx-userimagess3bucket-xxxx** bucket, and inspect the uploaded image. The path is **/Incoming/Jane\_Doe/Hawaii\_Vacation/**. You should find the image you uploaded.

**Task complete:** You uploaded a sample picture to be part of the photo book.

**Task 4: Subscribing to user approval email notification**

During the image-processing state machine flow, once all the images are put into a digital album, an email is sent to the user for approval. The email body contains two links, an approval link and a reject link. Based on the user action, the state moves to the next step. In this task, you subscribe to the Amazon SNS email topic to receive emails.

1. At the top of the AWS Management Console, in the search bar, search for and choose

Simple Notification Service

.

1. In the navigation pane, choose **Topics**.
2. Choose the link for the **SNSTopicForUserCommunication** topic.
3. Choose **Create subscription**.
4. On the **Create subscription** page, configure the following:

* For **Protocol**, choose **Email**.
* For **Endpoint**, enter a valid email address.

1. Choose **Create subscription**.

After a brief delay, you receive an email to confirm the subscription. You must confirm before the subscription is activated.

1. To confirm the subscription, choose the **Confirm subscription** link in the email.

**Task complete:** You subscribed to the Amazon SNS email topic to receive emails.

**Task 5: Making the digital album**

You have uploaded images, and they are waiting to be processed. In this task, using Step Functions, each image goes through a series of steps starting with metadata extraction, type check, image content validation, resize, and watermarking. The final step is stitching all the images into a single PDF document in the **/Book** folder. This PDF is the final album for you to approve or reject before it is sent for printing. In this task, you process the images and then download the album to see how it looks.

1. Return to your web browser tab with the **AWS Cloud9** terminal.
2. To invoke the Step Functions state machine, run the following commands in the AWS Cloud9 terminal window after making the following changes:

* Replace the **CREATE\_BOOK\_BINDING\_URL** placeholder value with the **CreateBookBindingUrl** value that is listed to the left of these instructions.

**Note:** Keep the quotation marks around the values when you replace them.

curl -d '{"userName":"Jane Doe","albumName":"Hawaii Vacation"}' -H "Content-Type: application/json" -X POST CREATE\_BOOK\_BINDING\_URL

**Expected output:** The output displays a message stating the request has been received, similar to the following:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* EXAMPLE OUTPUT \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

"Your book printing request has been received, once the digital book is available, you receive an email to validate the pdf book and a link to approve or reject the physical book printing."

1. Return to your web browser tab with the AWS Management Console.
2. At the top of the AWS Management Console, in the search bar, search for and choose

S3

.

1. Navigate to the **xxxx-userimagess3bucket-xxxx** bucket, and inspect the various folders. The **Book/Jane\_Doe/Hawaii\_Vacation** path contains the final PDF digital album for approval. You can open or download the PDF digital album to review it.
2. At the top of the AWS Management Console, in the search bar, search for and choose

Step Functions

.

1. Choose the link for the **ImageProcessStateMachine** state machine.
2. On the **Executions** tab, choose the link for the most recent state.
3. In the **Graph view** section, notice that the **UserApproval** step is currently **In progress**. This particular step requires human intervention.

You should receive an email to approve a Step Functions invocation.

1. To confirm the invocation, choose the **Approval Link** in the email.
2. Inspect the state machine workflow again.

The PDF album was approved, and now a message has been placed into an Amazon SQS queue for the third-party vendor to pick up for printing the the physical book.

1. At the top of the AWS Management Console, in the search bar, search for and choose

Simple Queue Service

.

1. For the **BookPrintQueue** queue, note the value in the **Messages available** column. There should be one message available.
2. Choose the link for **BookPrintQueue**.
3. Choose **Send and receive messages**.
4. On **Send and receive messages** page, in the **Receive messages** section, choose **Poll for messages**.
5. In the **Message** section, choose the link for the displayed message ID to see the Details, Body, and Attributes of the message.

**Task complete:** You processed the images, verified the album, and approved printing it.

**Task 6: Setting up the book printing state machine**

This task walks you through setting up a Step Functions state machine. This state machine reads the messages from the Amazon SQS print queue from the previous task, sends the messages to the third-party print vendor, and finally sends an email to the customer saying the book has shipped.

1. At the top of the AWS Management Console, in the search bar, search for and choose

Step Functions

.

1. Choose **Create state machine**.
2. Choose the **+ Blank** template under **All**.
3. Choose **Select**.
4. At the upper-left corner of the page, choose the  **Edit state machine name** button.
5. On the **Details** page, configure the following:

* For **State machine name**, enter

BookprintStateMachine

* For **Type**, choose **Standard**.

1. In the **Permissions** section, configure the following:

* For **Execution role**, under **Choose an existing role**, choose **StateMachineRole**.

1. At the top of the page, choose the **Code** tab.
2. Replace the sample code snippet with the following code:

* Replace the **PRINT\_VENDOR\_LAMBDA\_ARN** placeholder value with the **PrintVendorLambdaArn** value that is listed to the left of these instructions.
* Replace the **SNS\_TOPIC\_FOR\_USER\_COMMUNICATION\_ARN** placeholder value with the **SNSTopicForUserCommunicationArn** value that is listed to the left of these instructions.

{

"Comment": "This state machine is to send request to 3rd party",

"StartAt": "RequestSentTo3rdParty",

"States": {

"RequestSentTo3rdParty": {

"Type": "Task",

"Resource": "PRINT\_VENDOR\_LAMBDA\_ARN",

"InputPath": "$",

"ResultPath": "$.extractedMetadata",

"Next": "UpdateFrom3rdParty"

},

"UpdateFrom3rdParty": {

"Type": "Choice",

"Choices": [

{

"Variable": "$.extractedMetadata.shippingStatus",

"StringEquals": "Shipped",

"Next": "Shipped"

},

{

"Variable": "$.extractedMetadata.shippingStatus",

"StringEquals": "FailedForPayment",

"Next": "FailedToProcess"

}

]

},

"Shipped": {

"Type": "Pass",

"Next": "NotifyUser(SNS)"

},

"FailedToProcess": {

"Type": "Pass",

"Next": "NotifyUser(SNS)"

},

"NotifyUser(SNS)": {

"Type": "Task",

"Resource": "arn:aws:states:::sns:publish",

"Parameters": {

"TopicArn": "SNS\_TOPIC\_FOR\_USER\_COMMUNICATION\_ARN",

"Message.$": "$.extractedMetadata.message",

"MessageAttributes": {

"msg": {

"DataType": "String",

"StringValue": "additional instructions!"

}

}

},

"End": true

}

}

}

1. Choose **Create**.
2. On the **BookprintStateMachine** page, in the **Details** section, copy the **Arn** value and paste it in a text editor to use later.

The ARN looks similar to the following: *arn:aws:states:us-west-2:xxxx:stateMachine:BookprintStateMachine*.

The state machine setup is now complete. In the next few steps, you map this state machine ARN to the *PrintVendorInvokeFunction* Lambda function.

1. At the top of the AWS Management Console, in the search bar, search for and choose

Lambda

.

1. In the search box, type

PrintVendorInvokeFunction

, and then press **Enter**.

1. Choose the link for the **PrintVendorInvokeFunction** function.
2. Choose the **Configuration** tab.
3. In the configuration navigation pane, choose **Environment variables**.
4. Choose **Edit**.
5. Replace **BOOK\_PRINT\_STATE\_MACHINE\_ARN** with the **BookprintStateMachine ARN** that you copied in a previous step.
6. Choose **Save**.

There is one step left before pushing the books to the third-party vendor. At this time, the Amazon SQS queue is holding messages, but there is no consumer to pick up those messages. Next, you enable the consumer so that the BookprintStateMachine activates and sends the books to the third-party vendor.

1. At the top of the AWS Management Console, in the search bar, search for and choose

Simple Queue Service

.

1. Choose the link for **BookPrintQueue**.
2. On the **BookPrintQueue** page, choose the **Lambda triggers** tab.
3. Choose **Configure Lambda function trigger**.
4. On the **Trigger AWS Lambda Function** page, for **Specify an AWS Lambda function available for this queue** menu, choose **PrintVendorInvokeFunction**.
5. Choose **Save**.
6. At the top of the AWS Management Console, in the search bar, search for and choose

Step Functions

.

1. Choose the link for **BookprintStateMachine**.
2. On the **Executions** tab, check for any successful executions.

**Refresh:** You might need to choose  refresh to see the latest state machine executions.

After approximately two minutes, you should receive an email indicating that your photo album has shipped.

**Learn more:** Refer to the [Appendix](https://labs.skillbuilder.aws/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A470679935125%3Ablueprintversion%2Fspl-254%3A1.0.18-2f2fc8ec/en-US#appendix) section to learn about AWS Lambda layers. This lab used Lambda layers in a few of its Lambda functions.

**Task complete:** You successfully configured and verified setting up the book printing state machine.

**Conclusion**

You have successfully done the following:

* Understood an event-driven architecture.
* Understood how Step Functions is configured to orchestrate serverless applications.
* Taken advantage of Amazon SQS and SNS.
* Created and configured Lambda functions and API Gateway resources.
* Made configuration updates to restore API functionality.

**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.

**Appendix**

AWS LAMBDA LAYERS

You can configure your Lambda function to pull in additional code and content as layers. A layer is a ZIP archive that contains libraries, a custom runtime, or other dependencies. With layers, you can use libraries in your function without needing to include them in your deployment package. Layers let you keep your deployment package small, which makes development easier. You can avoid errors that can occur when you install and package dependencies with your function code. For Node.js, Python, and Ruby functions, you can develop your function code in the Lambda console as long as you keep your deployment package under 3 MB.

This lab used Lambda layers in the following functions:

* **WatermarkFunction**
* **ResizeFunction**
* **ExtractImageMetadataFunction**

Open these Lambda functions and notice the use of one common AWS Lambda layer, **image-magick-layer-xxxx**. You can use the same layer across many Lambda functions to keep your code modular and concise.

For more information about Lambda layers, see the AWS Lambda Developer Guide at [*https://docs.aws.amazon.com/lambda/latest/dg/configuration-layers.html*](https://docs.aws.amazon.com/lambda/latest/dg/configuration-layers.html).