Corrections, feedback, or other questions? Contact us at https://support.aws.amazon.com/#/contacts/aws-training.

Contents

aws training and certification

Lab 1: Deploying a Simple Serverless Application

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*.

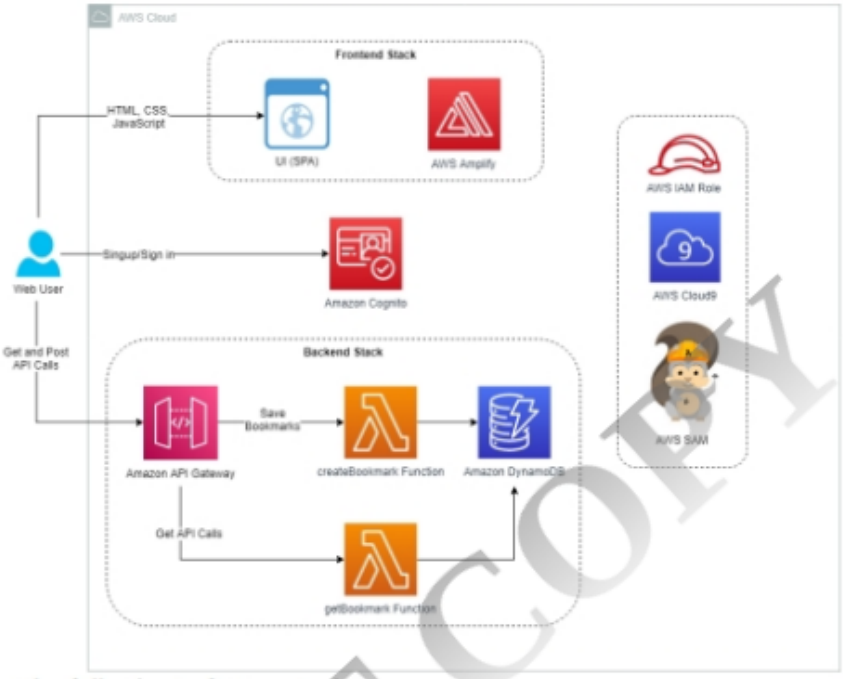
Overview

Your technical support team has a partially homegrown, highly customized ticketing system. Between development team members and tech support, you are often pointing people to links to help resolve issues, but there are no real strategies for managing the resources. You have a team knowledge base, which is maintained by the development managers, but new suggestions must be emailed to a shared mailbox for review before they are published. So submissions, reviews, and updates happen sporadically.

A recent hackathon produced a partial solution for saving resource links with relevant information. Your product manager has asked your team to build the first iteration of this application as an opportunity to address the business need while learning about and piloting a serverless architecture.

In this lab, you deploy the backend application and front-end application in various steps, and update the front-end configurations.

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



This lab uses the following software stack:

- AWS Amplify
- AWS Serverless Application Model (AWS SAM)
- Amazon Cognito
- Vue JavaScript framework
- AWS Cloud9
- Swagger API
- Amazon DynamoDB

Objectives

After completing this lab, you will be able to:

- Configure authentication through an Amazon Cognito user pool
- Deploy your backend code using AWS SAM
- View API documentation using the Swagger Editor
- Update your front-end configuration and run the build through AWS Cloud9 to test it prior to deployment
- Deploy your front-end application using Amplify

Prerequisites

This lab requires:

- Access to a notebook computer with Wi-Fi and Microsoft Windows, macOS X, or Linux (Ubuntu, SUSE, or Red Hat)
- For Microsoft Windows users, administrator access to the computer
- An internet browser such as Chrome, Firefox, or Internet Explorer 9 (previous versions of Internet Explorer are not supported)
- A text editor

⚠ **Note** The lab environment is not accessible using an iPad or tablet device, but you can use these devices to access the lab guide.

Duration

This lab requires approximately **60 minutes** to complete.

Start lab

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

This starts the process of provisioning the lab resources. An estimated amount of time to provision the lab resources is displayed. You must wait for the resources to be provisioned before continuing.

ⓘ If you are prompted for a token, use the one distributed to you (or credits you have purchased).

2. To open the lab, choose Open Console.

The **AWS Management Console** sign-in page opens in a new web browser tab.

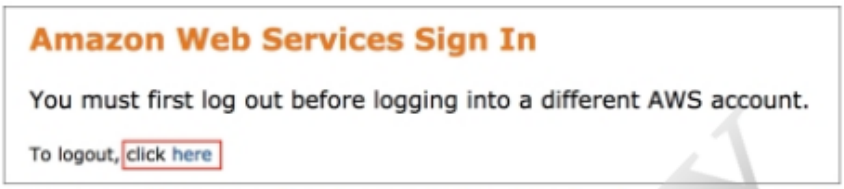
3. On the **Sign in as IAM user** page:

- For **IAM user name**, enter `awsstudent`.
- For **Password**, copy and paste the **Password** value listed to the left of these instructions.
- Choose Sign in.

⚠ **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.

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: Understanding key services and the environment setup

In this task, you look at the different AWS services that are used in this development. Take a few minutes to navigate the console for each service. Next, you download your source code, including both backend and front-end code, from an Amazon Simple Storage Service (Amazon S3) bucket to your AWS Cloud9 environment.

- **AWS Amplify** is a set of tools and services that enables mobile and front-end web developers to build secure, scalable full stack applications powered by AWS. Amplify includes an open-source framework with use-case-centric libraries and a powerful toolchain to create and add cloud-based features to your application, and a web-hosting service to deploy static web applications.
- **AWS SAM** is an open-source framework for building serverless applications. It provides shorthand syntax to express functions, APIs, databases, and event source mappings. With just a few lines per resource, you can define the application you want and model it using YAML. During deployment, AWS SAM transforms and expands the AWS SAM syntax into AWS CloudFormation syntax, enabling you to build serverless applications faster.
- **Amazon Cognito** lets you add user sign-up, sign-in, and access control to your web and mobile apps quickly and easily. Amazon Cognito scales to millions of users and supports sign-in with social identity providers, such as Facebook, Google, and Amazon, and enterprise identity providers via SAML 2.0.
- **Vue JavaScript framework** is a progressive framework for building user interfaces. Unlike other monolithic frameworks, Vue is designed to be incrementally adoptable. The core library focuses on the view layer only and is easy to pick up and integrate with other libraries or existing projects. Vue is also perfectly capable of powering sophisticated single-page applications when used in combination with modern tooling and supporting libraries.
- **AWS Cloud9** is a cloud-based integrated development environment (IDE) that lets you write, run, and debug your code with just a browser. It includes a code editor, debugger, and terminal. AWS Cloud9 makes it easy to write, run, and debug serverless applications. It pre-configures the development environment with all the SDKs, libraries, and plugins needed for serverless development.
- **Swagger API** is an open-source software framework backed by a large ecosystem of tools that help developers design, build, document, and consume RESTful web services. Swagger also allows you to understand and test your backend API specifically.
- **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.

To set up your environment, open the pre-provisioned AWS Cloud9 environment. From there, download and unzip your source code so that you can deploy the AWS SAM application in an upcoming task.

4. Choose Services and select **Cloud9**.
5. In the left navigation pane, choose **Your environments**.
6. For the **BookmarkAppDevEnv** environment, choose Open IDE

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

Note If the browser is running in an incognito session, a pop-up window with an error message will be displayed when the Cloud9 instance is opened. Choose the **OK** button to continue. Browser in a non incognito mode is recommended.

7. To download your source code, run the following commands:

```
cd ~/environment
wget https://us-west-2-tcprod.s3-us-west-2.amazonaws.com/courses/ILT-TF-200-SVDVSS/v1.0.29/lab-1-Bookmarks/scripts/app-code.zip
unzip app-code.zip
cd app-code
```

8. Run the commands below to add more disk space to your AWS Cloud9 environment.

- **Command:** Run the command below to add more disk space:

```
bash resize.sh 50
```

Expected output:

```
******************************
**** This is OUTPUT ONLY. ****
******************************

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    19  100    19    0     0   1526      0 --:--:-- --:--:-- --:--:--  1583
{
    "VolumeModification": {
        "VolumeId": "vol-0bf4348702c6f53d3",
        "ModificationState": "modifying",
        "TargetSize": 50,
        "TargetIops": 100,
        "TargetVolumeType": "gp2",
        "TargetMultiAttachEnabled": false,
        "OriginalSize": 10,
        "OriginalIops": 100,
        "OriginalVolumeType": "gp2",
        "OriginalMultiAttachEnabled": false,
        "Progress": 0,
        "StartTime": "2022-03-25T20:35:25+00:00"
    }
}
```

```
CHANGED: partition=1 start=4096 old: size=20967391 end=20971487 new: size=31453151
end=31457247
meta-data=/dev/xvda1              isize=512    agcount=6, agsize=524159 blks
         =                        sectsz=512   attr=2, projid32bit=1
         =                        crc=1        finobt=1 spinodes=0
data     =                        bsize=4096   blocks=2620923, imaxpct=25
         =                        sunit=0      swidth=0 blks
naming   =version 2               bsize=4096   ascii-ci=0 ftype=1
log      =internal                bsize=4096   blocks=2560, version=2
         =                        sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                    extsz=4096   blocks=0, rtextents=0
data blocks changed from 2620923 to 3931643
```

Note: There is more information about the **resize.sh** script in the **Additional Resources** section at the end of this lab.

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

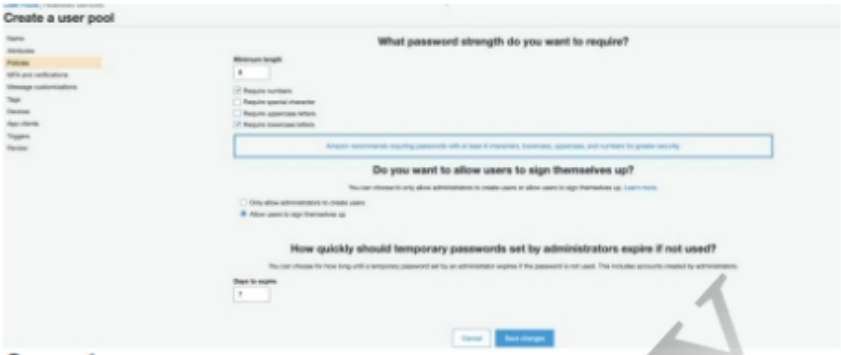
Task 2: Setting up authentication with Amazon Cognito

In this task, you set up your Amazon Cognito user pool for authentication.

9. From the AWS Management Console, choose Services and open **Amazon Cognito** in a new browser tab.
10. Choose Manage User Pools
11. Choose Create a user pool
12. For **Pool name**, enter `bookmark-app-userpool`
13. Choose Review **defaults**.

From here, modify a few of the default settings.

14. On the left side of the page, choose **Policies**.
15. Under **What password strength do you want to require?**, clear the check boxes for **Require special character** and **Require uppercase letters**. Leave the other boxes selected to keep the password simple during account creation. Refer to the following screenshot.



16. Choose **Save changes**.
17. On the left side of the page, choose **Triggers**.
18. From the **Pre sign-up** dropdown list, choose the **PresignupLambda-** AWS Lambda function.

ⓘ **Note** This Lambda function automatically verifies your email and phone number during the sign-up process.

19. Choose **Save changes**.
20. On the left side of the page, choose **App clients**.
21. Choose **Add an app client**.
22. For **App client name**, enter `AppClientForBookmarkUserPool`
23. Clear the box for **Generate client secret**.
24. Choose Create app client
25. Choose **Return to pool details**
26. Choose Create pool

Task 3: Deploying the backend application using AWS SAM

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

The following are a few 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.

27. Go back to the AWS Cloud9 browser tab.
28. On the left side of the AWS Cloud9 IDE, choose the dropdown arrow next to the **app-code** folder, and then the **backend** folder.
29. 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.

30. 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 **createBookmark** folder.
31. Choose and open the **index.js** file to view the code within this file.

This file contains your actual Lambda handler logic for the **createBookmark** Lambda function. Review the other Lambda functions in the **deleteBookmark**, **getBookmark**, **listBookmarks**, and **updateBookmark** folders.

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

This is the configuration file for the project. To replace *BUCKET_NAME*, *AWS_REGION* and *LAMBDA_ROLE_ARN* with actual values in the **samconfig.toml** file, you will run a set of CLI commands to update these values before deploying.

33. In the AWS Cloud9 terminal, run the following AWS CLI and bash commands to update the values in the **samconfig.toml** file.

```
sudo yum -y install jq
cd ~/environment/app-code/backend
export BUCKET_NAME=$(aws s3api list-buckets --query "Buckets[?contains(Name,
'bookmark')].Name" --output text)
sed -Ei "s|<BUCKET_NAME>|${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=$(aws iam  list-roles --query "Roles[?contains(RoleName,
'LambdaDeployment')].Arn" --output text)
sed -Ei "s|<LAMBDA_ROLE_ARN>|${LAMBDA_ROLE_ARN}|g" samconfig.toml
cd ..
```

34. 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 Amazon S3 bucket from where the code is deployed, the role needed for AWS SAM to deploy, and the Region. The file contains the parameters for the CloudFormation stack to be deployed.

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

```
cd backend
sam deploy
```

This command deploys your application to the AWS Cloud. The command takes the deployment artifacts you build with the **sam build** command, packages and uploads them to an Amazon 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 Cloud9. For more information about installing the AWS SAM CLI, see 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.

36. Go to your browser tab with the AWS Management Console. Choose Services and open **CloudFormation** in a new browser tab.

37. Choose the **sam-bookmark-app** stack.

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

38. To view the AWS resources that were created within this stack, choose the **Resources** tab.

39. To open the DynamoDB resource, choose the **Physical ID** of the **bookmarksTable**.

This is the **bookmarksTable** table that was created when deploying your backend code via AWS SAM.

40. Choose **Explore table items** at the top-right corner of the page.

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

41. Choose Services and select **API Gateway**.

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

42. Choose the **Bookmark App**.

43. On the left side of the page, choose **Stages**.

44. In the **Stages** pane, choose the ▶ **dev** stage.

45. Choose the **Export** tab.

46. Choose **Export as Swagger + API Gateway Extensions**, and select the JSON or YAML option.

This saves the file to your local directory for you to open and view.

View the API documentation using Swagger Editor

47. Choose the following link: https://editor.swagger.io/

48. Choose **File** and **Import file** to import the file that you just saved.

The right side of the console displays all deployed APIs.

49. Expand each API to check the parameters and the functionality.

Learn more For more information about integrating Amazon API Gateway with Swagger, see API Gateway Integration.

Task 4: Reviewing the front-end application code and updating the configuration file

Before deploying the bookmark application through Amplify, you can view the source code and test the front-end application from AWS Cloud9. You can also make changes to the code by reworking some of the source files within AWS Cloud9.

50. Return to the Cloud9 IDE browser tab.

51. To switch to your **frontend** directory and install packages and dependencies, run the following commands:

```
cd ../frontend
npm install
```

52. To run the dev server, run the following command:

```
npm run dev
```

53. At the top of the Cloud9 IDE, choose **Preview**.

54. In the **Preview** dropdown, select **Preview Running Application**.

Within Cloud9, a new browser tab will pop up showing you a local version of the bookmark application.

55. To stop the bookmark application from running, select the terminal window and press **Ctrl + C**.

56. 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.

57. 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 along with **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.

58. Choose and open the **aws-exports.js** file.

59. Copy and paste the following code block into the file:

```
const awsmobile = {
    "aws_project_region": "<AWS_REGION>",
    "aws_cognito_region": "<AWS_REGION>",
    "aws_user_pools_id": "<COGNITO_USER_POOL_ID>",
    "aws_user_pools_web_client_id": "<APP_CLIENT_ID>",
    "oauth": {},
    "aws_cloud_logic_custom": [
        {
            "name": "Bookmark App",
            "endpoint": "<API_GATEWAY_URL>",
            "region": "<AWS_REGION>"
        }
    ]
};

export default awsmobile;
```

60. Save the **aws-exports.js** file.

61. 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=$(aws apigateway get-rest-apis --query 'items[?name==`Bookmark
App`].id' --output text)
export AWS_REGION=$(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=$(aws cognito-idp list-user-pools --query
"UserPools[?contains(Name, 'bookmark-app-userpool')].Id"  --max-results 1 --output
text)
sed -Ei "s|<COGNITO_USER_POOL_ID>|${COGNITO_USER_POOL_ID}|g" aws-exports.js
export APP_CLIENT_ID=$(aws cognito-idp list-user-pool-clients --user-pool-id
${COGNITO_USER_POOL_ID}  --query "UserPoolClients[?contains(ClientName,
'AppClientForBookmarkUserPool')].ClientId"  --output text)
sed -Ei "s|<APP_CLIENT_ID>|${APP_CLIENT_ID}|g" aws-exports.js
cd ..
```

62. Save the **aws-exports.js** file.

The changed file should look like the following image:

ⓘ **Note** The parameters provided in the **aws-exports.js** file enable the front-end of the application to connect to the backend. The URL provided is the backend URL for the application that was deployed through AWS SAM.

63. 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.

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

```
cd dist
ls
```

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

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


```
zip -r app.zip *
```

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

```
export BUCKET_NAME=$(aws s3api list-buckets --query "Buckets[?contains(Name,
'bookmark')].Name" --output text)
aws s3 cp app.zip s3://${BUCKET_NAME}
```

Task 5: Deploying the bookmark application using Amplify

In this task, you deploy the bookmark application using the Amplify console.

67. From the AWS Management Console, choose Services and select **AWS Amplify**.

68. Choose the menu ☰ icon at the top-left corner of the page, and then choose **All apps**.

69. Choose New app ▼ and select **Host web app** from the dropdown list.

70. Choose **Deploy without Git provider**, and then choose Continue

71. For **Manual deploy**, configure the following information:

- **App name:** Enter `BookmarkApp`
- **Environment name:** Enter `dev`
- **Method:** Choose **Amazon S3**
- **Bucket:** Choose the bucket with the name **bookmarkbucket** in it
- **Zip file:** Select **app.zip** (When the **Bucket** is selected, this dropdown menu auto-populates.)

72. Choose Save and deploy

That's it! You just deployed your front-end code to AWS via Amplify. You should see a green **Deployment successfully completed** bar in the middle of your page.

Task 6: Testing the bookmark application

In this task, you test the bookmark application that was deployed in Amplify in the previous task. You start by creating an account, and then you add bookmarks to the application. Verify the addition by checking the DynamoDB table.

73. From the Amplify console, choose the URL under **Domain**.

This opens the bookmark application.

74. From the bookmark application page, choose **Create account**

75. Fill in the fields with your information, and choose CREATE ACCOUNT
76. To add a bookmark, choose the plus icon at the top-right corner of the page.
77. For **Add New Bookmark**, configure the following information:

- **Name:** Enter `aws`
- **Description:** Enter `aws cloud training`
- **Bookmark URL:** Enter `https://www.aws.training/`

78. Choose ADD BOOKMARK
79. Add another bookmark of your choice.

When you add these bookmarks on the front end, DynamoDB adds this data to the **bookmarksTable** table.

80. From the AWS Management Console, choose Services and select **DynamoDB**.
81. On the left side of the page, choose **Tables**.
82. Choose **bookmarksTable**.
83. Choose **Explore table items** at the top-right corner of the page.

The bookmarks that you just added are listed in the **items returned** section.

84. Go back to the bookmark application browser tab, and choose the trash can icon to delete all of your bookmarks.

When the bookmarks are deleted, DynamoDB deletes this data from the **bookmarksTable** table.

85. Go to the DynamoDB console browser tab.
86. Choose **Run** to refresh the **Items returned** section.

You can see that the deleted bookmarks are no longer in the table.

Conclusion

Congratulations! You now have successfully:

- Configured authentication through an Amazon Cognito user pool
- Deployed your backend code using AWS SAM
- Viewed API documentation using the Swagger Editor

- Updated your front-end configuration file and ran the build through AWS Cloud9 to test it prior to deployment
- Deployed your front-end application using Amplify

End lab

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

87. Return to the **AWS Management Console**.
88. At the upper-right corner of the page, choose **awsstudent@<AccountNumber>**, and then choose **Sign out**.
89. Choose End Lab.
90. Choose Submit.
91. (Optional):

- Select the applicable number of stars to rate your lab experience.
  - 1 star = Very dissatisfied
  - 2 stars = Dissatisfied
  - 3 stars = Neutral
  - 4 stars = Satisfied
  - 5 stars = Very satisfied
- Enter a comment.
- Choose **Submit**.

You can close the window if you don't want to provide feedback.
For more information about AWS Training and Certification, see *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*.

Additional resources

- For more information about Swagger, see https://swagger.io/tools/swagger-ui/.
- For more information about the Vue JavaScript framework, see https://vuejs.org/.

aws training and certification

Lab 2: Message Fan-Out with Amazon EventBridge

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

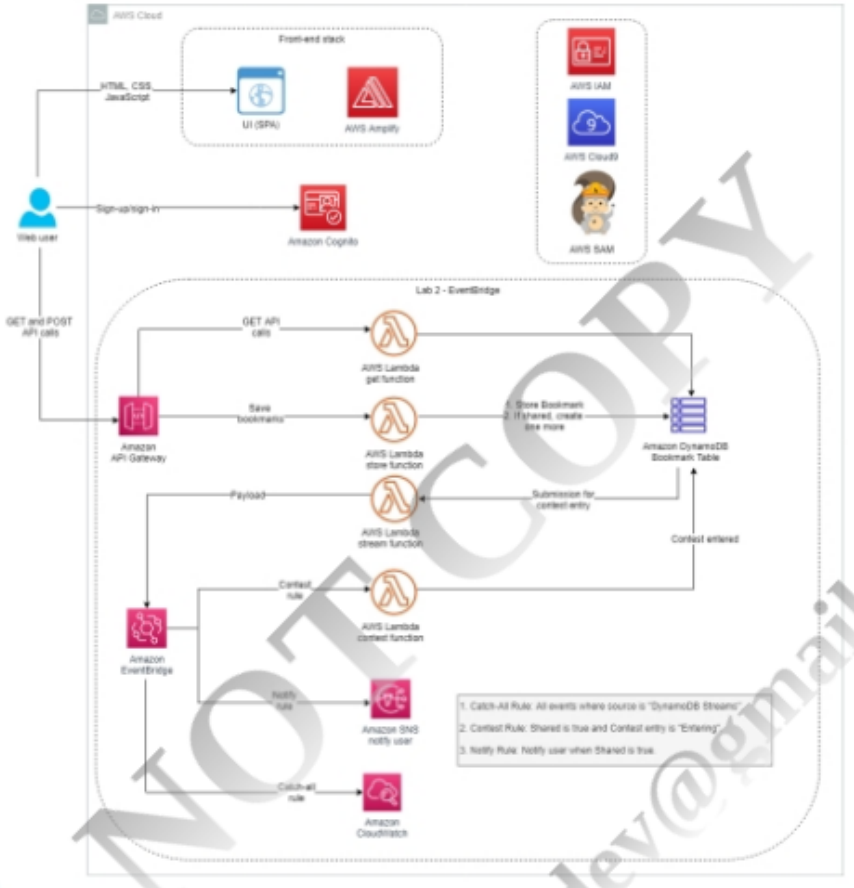
Corrections, feedback, or other questions? Contact us at *AWS Training and Certification*.

Overview

Now that you have verified your serverless proof of concept with the first iteration, you need to add the capability for sharing bookmarks with the team. To get the managers on board with your new application, you need to align to the existing knowledge base process. To get the team on board, your product manager has authorized a contest so that each new submission through the application enters the submitter into a drawing to win a prize.

In this lab, you add a new feature to the bookmark application to address the contest use case using Amazon EventBridge. When someone shares a bookmark, the bookmark application should automatically notify the mailbox that is being used to monitor submissions and enter the bookmark into the shareable bookmarks contest.

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



Objectives

After completing this lab, you will be able to:

- Enable Amazon DynamoDB Streams as an event source for an AWS Lambda function that is invoked when new items are added to a DynamoDB table
- Configure an EventBridge event bus with a Lambda function as its event source and Lambda, Amazon Simple Notification Service (Amazon SNS), and Amazon CloudWatch as targets
- Configure EventBridge rules that route events to your targets based on the criteria that you specify
- Configure an SNS topic that notifies an email subscriber

Prerequisites

This lab requires:

- Access to a notebook computer with Wi-Fi and Microsoft Windows, macOS X, or Linux (Ubuntu, SUSE, or Red Hat)

- For Microsoft Windows users, administrator access to the computer
- An internet browser such as Chrome, Firefox, or Internet Explorer 9 (previous versions of Internet Explorer are not supported)

⚠ **Note** The lab environment is not accessible using an iPad or tablet device, but you can use these devices to access the lab guide.

Duration

This lab requires approximately **60 minutes** to complete.

Start lab

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

This starts the process of provisioning the lab resources. An estimated amount of time to provision the lab resources is displayed. You must wait for the resources to be provisioned before continuing.

ⓘ If you are prompted for a token, use the one distributed to you (or credits you have purchased).

2. To open the lab, choose Open Console.

The **AWS Management Console** sign-in page opens in a new web browser tab.

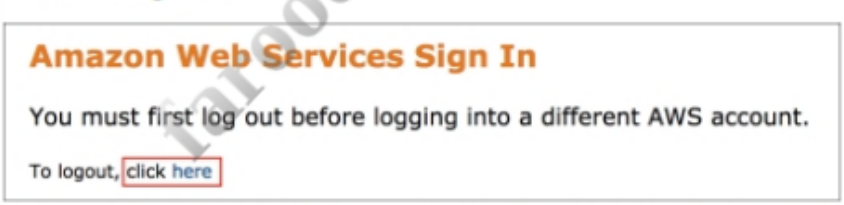
3. On the **Sign in as IAM user** page:

- For **IAM user name**, enter `awsstudent`.
- For **Password**, copy and paste the **Password** value listed to the left of these instructions.
- Choose Sign in.

⚠ **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.

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: Understanding key services and application setup

In this task, you open the AWS Cloud9 integrated development environment (IDE) and download the application code. Once the code is unzipped, a startup script automates the launching of the front-end and backend application code. You then launch and deploy the application via the AWS Amplify console.

- **Amazon EventBridge** makes it easy to build event-driven applications because it takes care of event ingestion, delivery, security, authorization, and error handling for you. To achieve the promises of serverless technologies with event-driven architecture, such as being able to individually scale, operate, and evolve each service, the communication between the services must happen in a loosely coupled and reliable environment. Event-driven architecture is a fundamental approach for integrating independent systems or building up a set of loosely coupled systems that can operate, scale, and evolve independently and flexibly. In this lab, you use EventBridge to address the contest use case.
- **Amazon DynamoDB Streams** is an ordered flow of information about changes to items in a DynamoDB table. When you enable a stream on a table, DynamoDB captures information about every modification to data items in the table.
- **Amazon Simple Notification Service (Amazon SNS)** is a fully managed messaging service for both system-to-system and app-to-person communication. It enables you to communicate between systems through publish/subscribe (pub/sub) patterns that enable messaging between decoupled microservice applications or to communicate directly to users via SMS, mobile push, and email. The system-to-system pub/sub functionality provides topics for high-throughput, push-based, many-to-many messaging. Using SNS topics, your publisher systems can fan out messages to a large number of subscriber systems or customer endpoints, including Amazon Simple Queue Service (Amazon SQS) queues, Lambda functions, and HTTP and HTTPS, for parallel processing. The app-to-person messaging functionality enables you to send messages to users at scale using either a pub/sub pattern or direct-publish messages using a single API.

4. Choose Services and select **Cloud9**.
5. For the **BookmarkAppDevEnv** environment, choose Open IDE

Within a few seconds, the AWS Cloud9 environment launches.

Note If the browser is running in an incognito session, a pop-up window with an error message will be displayed when the Cloud9 instance is opened. Choose the **OK** button to continue. Browser in a non incognito mode is recommended.

6. In the AWS Cloud9 terminal, run the following commands. These commands download and run the startup script, which contains the application code:

```
wget https://us-west-2-tcprod.s3-us-west-2.amazonaws.com/courses/ILT-TF-200-SVDVSS/v1.0.29/lab-2-EventBridge/scripts/app-code.zip
unzip app-code.zip
```



```
cd app-code
chmod +x resize.sh
chmod +x startupscript.sh
./startupscript.sh
```

⚠ **Note** The script takes a couple of minutes to run. Once it is finished, you will deploy your bookmark application through Amplify. **Be sure to let the script finish running before moving on to the next step.**

What the script is doing: This script modifies the **samconfig.toml** file within the backend portion of the application code. The script replaces values such as AWS Region, stack name, and role Amazon Resource Name (ARN). Next, the script updates the **aws-exports.js** file with the Amazon Cognito metadata that was launched in the lab's AWS CloudFormation template. The script then runs the build, deploys the bookmark application, and uploads the **app.zip** file to the **samserverless** bucket.

7. In the AWS Management Console, choose Services and select **AWS Amplify**.
8. Choose the menu ☰ icon at the top-left corner of the page, and then choose **All apps**.
9. Choose New app ▼ and select **Host web app** from the dropdown list.

You can choose either **New app** button on the page.

10. Select **Deploy without Git provider**, and then choose Continue
11. On the **Manual deploy** page, configure the following information:

- **App name:** Enter `BookmarkApp`
- **Environment name:** Enter `dev`
- **Method:** Select **Amazon S3**
- **Bucket:** Select the bucket name that includes **samserverless**
- **Zip file:** Select **app.zip** (When the **Bucket** is selected, this dropdown menu auto-populates.)

12. Choose Save and deploy
13. Once you see the message **Deployment successfully completed** in the Amplify console, choose the **Domain** URL to open the bookmark application.
14. From the bookmark application page, choose **Create account**
15. Fill in the fields with your information, and choose CREATE ACCOUNT

Note Leave this browser tab open.

Task 2: Enabling DynamoDB Streams and setting up a Lambda trigger

In this task, you enable DynamoDB Streams on **bookmarksTable**.

16. From the AWS Management Console, choose Services and select and open **DynamoDB** in a new browser tab.
17. On the left side of the DynamoDB dashboard, choose **Tables**.
18. Choose **bookmarksTable**.
19. Choose **Exports and streams** tab.
20. In the **DynamoDB stream details** section, choose Enable
21. In the **Enable DynamoDB stream** page, choose **New and old images** in the **DynamoDB stream details** section.
22. Choose Enable stream

DynamoDB Streams helps ensure that each stream record appears exactly once in the stream. Also, for each item that is modified in a DynamoDB table, the stream records appear in the same sequence as the actual modifications to the item.

Now, create the Lambda function that is invoked by the DynamoDB table and alerts the EventBridge event bus.

23. From the AWS Management Console, choose Services and select and open **Lambda** in a new browser tab.
24. On the left side of the Lambda dashboard, choose **Functions**.
25. Choose Create function
26. On the **Create function** page, configure the following information:

- **Function name:** Enter `StreamTrigger`
- **Runtime:** Select **Node.js 14.x**

27. Under **Permissions**, expand the **Change default execution role** section, and then select **Use an existing role**.
28. From the **Existing Role** dropdown menu, choose the role name that includes **EventBridgeLambdaRole**.
29. Choose Create function
30. Choose + Add trigger

31. On the **Add trigger** page, configure the following information:

- In the **Trigger configuration** section, from the **Select a trigger** dropdown menu, choose **DynamoDB**.
- In the **DynamoDB table** search box, select the table with **bookmarksTable** in the name.
- Decrease the **Batch size** to 5

32. Leave the rest of the defaults the same, and then choose Add

33. Choose the **Code** tab to bring up the function again.

34. In the **Code source** section, select **index.js**, open the context (right-click) menu and choose **Open**.

35. Delete the existing code and paste the following code:

```
const EventBridge = require('aws-sdk/clients/eventbridge');
const ev = new EventBridge();

exports.handler = async (event) => {
    console.log(JSON.stringify(event, null, 2));
    try
    {
        for(let i=0; i< event.Records.length; i++) {
            const record = event.Records[i];
            console.log(record.eventID);
            console.log(record.eventName);
            if(record.eventName === 'INSERT' || record.eventName === 'MODIFY') {
                console.log('DynamoDB Record: %j', record.dynamodb);
                console.log('share flag:', record.dynamodb.NewImage.shared.BOOL);
                console.log('contest value: ', record.dynamodb.NewImage.contest.S);

                var pk = record.dynamodb.NewImage.id.S;
                var sharedFlag = record.dynamodb.NewImage.shared.BOOL;
                var contestValue = record.dynamodb.NewImage.contest.S;

                const bookmarkDetails = {
                    id: pk,
                    shared: sharedFlag,
                    contest: contestValue,
                    payload: record.dynamodb.NewImage
                };

                const params = {
                Entries: [
                  {
                    Source: 'DynamoDB Streams',
                    DetailType: 'Shared Bookmarks',
                    EventBusName: 'bookmarks-bus',
                    Detail: JSON.stringify(bookmarkDetails)
                  }
                ]
              };
              const response =  await ev.putEvents(params).promise();
              console.log("response:", response);
              //We can optimize the code by calling the putEvents outside of the loop
with promise all option. where all the records
```

```
                //will put in the bus in parallel.
              }
            }
        } catch (error) {
              throw new Error(JSON.stringify(error));
        }

}
```

36. Choose Deploy

You should see a message that says **Successfully updated the function StreamTrigger.**

This code sends updates to EventBridge only if there is an UPDATE or INSERT event, kicking off the next phase in the event-driven architecture.

Task 3: Subscribing to bookmark contest notifications

In this task, you create and subscribe to an SNS topic that sends notifications to you and your manager when a bookmark has been shared.

37. From the AWS Management Console, choose Services and select and open **Simple Notification Service** in a new browser tab.
38. On the right side of the page, in the **Create topic** box, for **Topic name**, enter `BookmarkTopic`.
39. Choose Next step
40. Confirm that **Type** is set to *Standard* and choose Create topic
41. Choose Create subscription
42. On the **Create subscription** page, configure the following information:

- **Topic ARN:** Confirm that it is the ARN with **BookmarkTopic** in the name
- **Protocol:** Select **Email**
- **Endpoint:** Enter a valid email address

43. Choose Create subscription

After a few moments, you should receive an email to confirm the subscription. You must confirm the subscription to activate it.

44. To confirm the subscription, choose the **Confirm subscription** link in the email that you receive.

Note For this scenario, this email address is considered the email address for the manager who will receive the notifications for the bookmarking contest.

You can move onto the next task as you await the Amazon SNS email confirmation.

Task 4: Setting up an event bus and configuring rules

In this task, you create rules in EventBridge to attach to an event bus. This event bus receives events from a Lambda stream trigger and then matches them to the applicable rules.

Rules watch for specific types of events. When a matching event occurs, the event is routed to the targets that are associated with the rule. A rule can be associated with one or more targets.

45. In the AWS Management Console, choose Services and select **Amazon EventBridge**.
46. On the left side of the page, choose the menu ☰ icon.
47. Choose **Event buses**.
48. Choose Create event bus
49. On the **Create event bus** page, in the **Name** field, enter `bookmarks-bus`
50. Choose Create
51. On the left side of the page, choose the menu ☰ icon.
52. Choose **Rules**.

The first rule that you need to create is the **catch-all-rule**. This rule sends the event payload to Amazon CloudWatch Logs after being invoked by the **StreamTrigger** Lambda function and passing through the event bus.

53. Choose Create rule
54. On the **Define rule detail** page, in the **Rule detail** section, configure the following information:

- **Name:** Enter `catch-all-rule`.
- **Event bus:** Select **bookmarks-bus** from the dropdown menu.
- **Rule Type:** Select **Rule with an event pattern**.

55. Choose Next
56. In the **Build event pattern** page, configure the following information:

- **Event source:** Select **Other**.
- **Event pattern:** Select **Custom patterns**.
- In the **Event pattern** code box, copy and paste in the following code:

```
{
  "source": [
    "DynamoDB Streams"
  ],
```



```
  "detail-type": [
    "Shared Bookmarks"
  ]
}
```

57. Choose Next

ⓘ **Note** EventBridge rules use event patterns to match AWS events on an event bus. When a pattern matches, the rule routes that event to a target. This event pattern is using DynamoDB Streams as the source and identifying the **Shared Bookmarks** value as the detail to invoke the CloudWatch catch-all log.

58. In the **Select target(s)** page, configure the following information:

- **Target types:** Select **AWS service**
- **Select a target:** Select **CloudWatch log group** from the dropdown menu
- **Log Group:** Select **/aws/events/** and then enter `catch-all`

59. Choose Next

60. In the **Configure tags - optional** page, choose Next

61. In the **Review and create** page, choose Create rule

The next rule that you need to create is the **Notification rule**. When invoked, this rule sends a notification via Amazon SNS to the email address that you used earlier to register with the bookmarks site.

62. Choose Create rule

63. On the **Define rule detail** page, in the **Rule detail** section, configure the following information:

- **Name:** Enter `notify-rule`
- **Event bus:** Select **bookmarks-bus** from the dropdown menu
- **Rule Type:** Select **Rule with an event pattern**

64. Choose Next

65. In the **Build event pattern** page, configure the following information:

- **Event source:** Select **Other**
- **Event pattern:** Select **Custom patterns**
- In the **Event pattern** code box, copy and paste in the following code:

```
{
  "source": [
    "DynamoDB Streams"
```

```
  ],
  "detail-type": [
    "Shared Bookmarks"
  ],
  "detail": {
    "shared": [
      true
    ],
    "contest": [
      {
        "anything-but": [
          "Entering"
        ]
      }
    ]
  }
}
```

66. Choose Next

ⓘ **Note** This event pattern is again using DynamoDB Streams as the source and **Shared Bookmarks** as the detail-type. The third level to this pattern is **detail: shared: true** along with **contest: anything-but:Entering**, which sends an Amazon SNS message when someone shares a bookmark. The condition without **Entering** will send only one message that the bookmark has been entered into the contest.

67. In the **Select target(s)** page, configure the following information:

- **Target types:** Select **AWS service**
- **Select a target:** Select **SNS topic** from the dropdown menu
- **Topic:** Select **BookmarkTopic** from the dropdown menu

68. Choose Next

69. In the **Configure tags - optional** page, choose Next

70. In the **Review and create** page, choose Create rule

The final rule that you need to create is the **contest-rule**. This rule invokes the **contest** Lambda function, which adds the relevant item into the **sam-bookmark-app-bookmarksTable**.

71. Choose Create rule

72. On the **Define rule detail** page, in the **Rule detail** section, configure the following information:

- **Name:** Enter `contest-rule`
- **Event bus:** Select **bookmarks-bus** from the dropdown menu
- **Rule Type:** Select **Rule with an event pattern**

73. Choose Next

74. In the **Build event pattern** page, configure the following information:

- **Event source:** Select **Other**
- **Event pattern:** Select **Custom patterns**
- In the **Event pattern** code box, copy and paste in the following code:

```
{
  "source": [
    "DynamoDB Streams"
  ],
  "detail-type": [
    "Shared Bookmarks"
  ],
  "detail": {
    "shared": [
      true
    ],
    "contest": [
      "Entering"
    ]
  }
}
```

75. Choose Next

ⓘ **Note** This event pattern uses the same structure and details but invokes the Lambda contest function instead of the Amazon SNS message.

76. In the **Select target(s)** page, configure the following information:

- **Target types:** Select **AWS service**
- **Select a target:** Select **Lambda function** from the dropdown menu
- **Function:** Select **contest** from the dropdown menu.

77. Choose Next

78. In the **Configure tags - optional** page, choose Next

79. In the **Review and create** page, choose Create rule

Task 5: Testing EventBridge rules

In this task, you add and share bookmarks. This kicks off EventBridge and the corresponding rules.

Note Before continuing, make sure that you have confirmed your email registration and email subscriptions from earlier in the lab.

80. Go to the browser tab with the bookmark application.

81. Choose the plus **+** icon at the top-right corner of the page.

82. On the **Add New Bookmark** page, add and share a bookmark of your choice. Make sure that the **Share Bookmark** toggle is set to **On**.

83. Choose **ADD BOOKMARK**.

ⓘ **Note** When a user shares a bookmark, the bookmark application navigates to the shared bookmark page that lists all of the bookmarks that have been shared. The bookmark application also updates the value of the **shared** column for that particular row from false to true in the **bookmarks-app-bookmarksTable** in DynamoDB.

84. Check your email for the Amazon SNS notification.

Note Within the payload of the Amazon SNS email, you should see **"shared":true**. This email confirms that the SNS topic and **notify-rule** worked as intended.

85. From the AWS Management Console, choose Services and select **CloudWatch**.

86. On the left side of the page, choose **Log groups**.

87. Locate and open the **/aws/events/catch-all** log group.

88. Open the most recent **Log stream**.

89. Next to the timestamp for the log stream, expand the message.

Here you can see important details from the catch-all rule that you created earlier, such as time, ARN of the user who added the bookmark, account ID, and username.

90. From the AWS Management Console, choose Services and select **DynamoDB**.

91. In the left navigation pane, choose **Tables**.

92. Choose the **bookmarksTable** table.

This is the **bookmarksTable** table that was created when deploying your backend code via the AWS Serverless Application Model (AWS SAM).

93. Choose **Explore table items** at the top-right corner of the page.

Here are the bookmarks that have been added. Notice the **shared** column, which shows either true or false, and the **contest** column, which shows **Entered**.

This information confirms that all three EventBridge rules worked and that the event-driven architecture was a success!

Conclusion

Congratulations! You now have successfully:

- Enabled DynamoDB Streams as an event source for a Lambda function that is invoked when new items are added to a DynamoDB table

- Configured an EventBridge event bus with a Lambda function as its event source and Lambda, Amazon SNS, and CloudWatch as targets
- Configured EventBridge rules that route events to your targets based on the criteria that you specify
- Configured an SNS topic that notifies an email subscriber

End lab

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

94. Return to the **AWS Management Console**.
95. At the upper-right corner of the page, choose **awsstudent@<AccountNumber>**, and then choose **Sign out**.
96. Choose End Lab.
97. Choose Submit.
98. (Optional):

- Select the applicable number of stars to rate your lab experience.
  - 1 star = Very dissatisfied
  - 2 stars = Dissatisfied
  - 3 stars = Neutral
  - 4 stars = Satisfied
  - 5 stars = Very satisfied
- Enter a comment.
- Choose **Submit**.

You can close the window if you don't want to provide feedback.
For more information about AWS Training and Certification, see *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.*

Additional resources

- For more information about EventBridge, see https://pages.awscloud.com/Deep-Dive-on-Amazon-EventBridge_2019_0919-SRV_OD.html.

- For more information about DynamoDB Streams, see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Streams.html.

aws training and certification

Lab 3: Workflow Orchestration Using AWS Step Functions

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

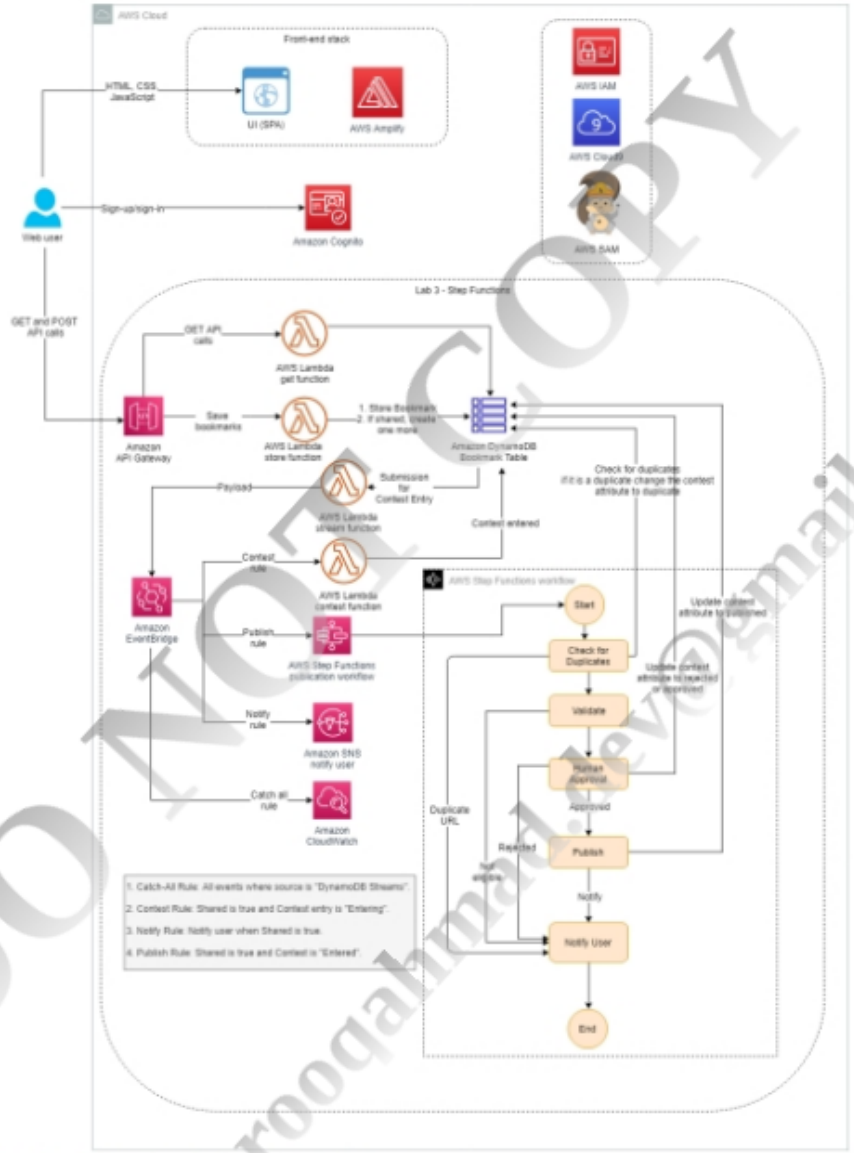
Corrections, feedback, or other questions? Contact us at *AWS Training and Certification*.

Overview

You deployed your serverless bookmark application, and then updated it to support bookmark sharing, routing of requests to the shared mailbox, and a contest feature. The managers who review the knowledge base submissions recognize that submissions are going to increase a lot with the application, so they want you to scale their submissions process by automating as much of it as possible. As part of their manual review, the managers check to make sure the submission isn't a duplicate, validate the URL and make sure it meets eligibility requirements to be shared, and then publish the resource. Then, the managers let the submitter know whether or not the item was added. When the item is a duplicate or is invalid, the submitter does not get a contest entry for that item, so the managers need to inform the contest administrator of duplicates and bad submissions. Given the variety of sources and type of materials that might be shared, the managers still want to maintain a manual approval step so that they can review the material and make sure it follows practices and technologies that they want to promote.

In this lab, you use AWS Step Functions to incorporate a workflow to automatically evaluate and publish new submissions, while still including a manual approval step as part of the process.

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



This lab uses the following software stack:

- AWS Amplify
- AWS Serverless Application Model (AWS SAM)
- Amazon Cognito
- Vue JavaScript framework
- AWS Cloud9
- Swagger API

- Amazon DynamoDB
- Amazon EventBridge
- Amazon Simple Notification Service (Amazon SNS)
- AWS Step Functions
- AWS Lambda
- Amazon CloudWatch
- Amazon API Gateway

Objectives

After completing this lab, you will be able to:

- Configure EventBridge to target a Step Functions workflow
- Use a Step Functions Standard Workflow to orchestrate tasks
- Use Lambda for tasks within a Step Functions state machine

Prerequisites

This lab requires:

- Access to a notebook computer with Wi-Fi and Microsoft Windows, macOS, or Linux (Ubuntu, SUSE, or Red Hat)
- For Microsoft Windows users, administrator access to the computer
- An internet browser such as Chrome, Firefox, or Internet Explorer 9 (previous versions of Internet Explorer are not supported)
- A text editor

⚠ **Note** The lab environment is not accessible using an iPad or tablet device, but you can use these devices to access the lab guide.

Duration

This lab requires approximately **60 minutes** to complete.

Start lab

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

This starts the process of provisioning the lab resources. An estimated amount of time to provision the lab resources is displayed. You must wait for the resources to be provisioned before continuing.

ⓘ If you are prompted for a token, use the one distributed to you (or credits you have purchased).

2. To open the lab, choose Open Console.

The **AWS Management Console** sign-in page opens in a new web browser tab.

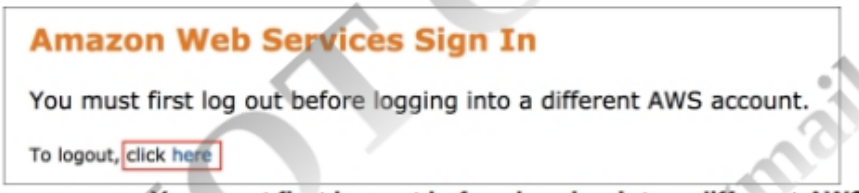
3. On the **Sign in as IAM user** page:

- For **IAM user name**, enter `awsstudent`.
- For **Password**, copy and paste the **Password** value listed to the left of these instructions.
- Choose Sign in.

⚠ **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.

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: Understanding key services and the environment setup

In this task, you look at the different AWS services that are used in this lab. The services include Step Functions and Amazon SNS.

- **AWS Step Functions** is a serverless function orchestrator that makes it easy to sequence Lambda functions and multiple AWS services 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. Orchestrating a series of individual serverless applications, managing retries, and debugging failures can be challenging. As your distributed applications become more complex, the complexity of managing them also grows. Step Functions automatically manages error handling, retry logic, and state. With its built-in operational controls, Step Functions manages sequencing, error handling, retry logic, and state, removing a significant operational burden from your team.
- **Amazon Simple Notification Service (Amazon SNS)** is a fully managed messaging service for both system-to-system and app-to-person (A2P) communication. The service enables you to communicate between systems through publish/subscribe (pub/sub) patterns that enable messaging between decoupled microservice applications or to communicate directly to users via SMS, mobile push, and email. The system-to-system pub/sub functionality provides topics for high-throughput, push-based, many-to-many messaging. Using Amazon SNS topics, your publisher systems can fan out messages to a large number of subscriber systems or customer endpoints including Amazon Simple Queue Service (Amazon SQS) queues, Lambda functions, and HTTP/S, for parallel processing. The A2P messaging functionality enables you to send messages to users at scale using either a pub/sub pattern or direct-publish messages using a single API.

In this task, you use the AWS Cloud9 integrated development environment (IDE) and download the application code. The front-end and backend code combine the workflows and resources from Labs 1 and 2. The code also contains new Lambda functions, a new API Gateway endpoint, and an SNS topic for the manual approval of the bookmark publishing process.

After downloading the code, you launch and deploy the application via the Amplify console.

4. In the AWS Management Console, on the Services menu, choose **Cloud9**.
5. For the **BookmarkAppDevEnv** environment, choose Open IDE

Within a few seconds, the AWS Cloud9 environment launches.

Note If the browser is running in an incognito session, a pop-up window with an error message will be displayed when the Cloud9 instance is opened. Choose the **OK** button to continue. Browser in a non incognito mode is recommended.

6. In the terminal pane, run the following commands to download the application code and run the startup script:

```
wget https://us-west-2-tcprod.s3-us-west-2.amazonaws.com/courses/ILT-TF-200-
SVDVSS/v1.0.29/lab-3-Step-Functions/scripts/app-code.zip
unzip app-code.zip
cd app-code
chmod +x resize.sh
chmod +x startupscript.sh
./startupscript.sh
```

The script takes a couple of minutes to run. When it is finished, you will deploy your bookmark application through Amplify.

What is the script doing?

- This script modifies the **samconfig.toml** file within the backend portion of the application code.
- The script replaces values such as AWS Region, stack name, and role Amazon Resource Name (ARN).
- The script then updates the **aws-exports.js file** with the Amazon Cognito metadata that was launched in the lab AWS CloudFormation template.
- As part of the AWS SAM deployment, a new API Gateway resource and a few Lambda functions are deployed. These functions are needed for the Step Functions workflow.
- The script then runs **npm build**, deploys the bookmark application, and uploads the **app.zip** file to the **samserverless** bucket.

Review the resources that have been deployed

7. Return to the AWS Management Console tab.
8. On the Services menu, choose **Amazon EventBridge**.
9. In the left navigation pane, choose **Rules**.

Note To expand the left navigation pane, choose the menu icon (☰).

10. From the Event **bus** dropdown menu, choose **bookmarks-bus**.

Review the rules that have been created under **bookmarks-bus**.

11. On the Services menu, choose **Lambda**.

Review the functions with **bookmark** in their names. These functions were deployed to create and share the bookmark URLs.

Note To review the code for a function, choose the name of the function, and scroll down to the **Code source** section.

12. On the Services menu, choose **API Gateway**.
13. In the **APIs** list, choose the name of **Bookmark App**.
14. In the **Resources** pane, under the **/approval** endpoint, choose **GET**.

This endpoint invokes a Lambda function that handles the human approval process.

15. On the Services menu, choose **Simple Notification Service**.
16. In the left navigation pane, choose **Topics**.

Review the topics that have been deployed.

Task 2: Creating SNS subscriptions, and deploying the bookmark application

In this task, you create subscriptions for the SNS topics, update the EventBridge rule to create CloudWatch logs, and deploy the front-end application using Amplify.

Create subscriptions for the SNS topics

17. In the Amazon SNS console, under **Topics**, choose the name of the **BookmarkTopic** topic.
18. Choose Create subscription
19. In the **Details** section, configure the following:

- **Protocol:** Choose **Email**
- **Endpoint:** Enter an email address where you can receive messages

20. Choose Create subscription
21. Check your email and confirm the subscription.

Note It may take a few minutes for the subscription confirmation to arrive. This subscription sends an email with the contents of a bookmark as it is entered in the contest.

22. In the left navigation pane, choose **Topics**.
23. Choose the name of the **ContestTopic** topic.
24. Choose Create subscription
25. In the **Details** section, configure the following:

- **Protocol:** Choose **Email**

- **Endpoint:** Enter an email address where you can receive messages

26. Choose Create subscription
27. Check your email and confirm the subscription.

Note It may take a few minutes for the subscription confirmation to arrive. This subscription sends an email with a link for human approval or rejection of a bookmark.

Deploy the front-end application using Amplify

28. On the Services menu, choose **AWS Amplify**.
29. In the left navigation pane, choose the menu icon (☰), and then choose **All apps**.
30. Choose New app ▼ and select **Host web app** from the dropdown list.
31. Choose **Deploy without Git provider**, and then choose Continue
32. On the **Manual deploy** page, configure the following:

- **App name:** Enter `BookmarkApp`
- **Environment name:** Enter `dev`
- **Method:** Choose **Amazon S3**
- **Bucket:** Choose **xxxx-samserverless-xxxx**
- **Zip file:** Choose **app.zip**

33. Choose Save and deploy

Create a user, add and share a bookmark, and check the EventBridge rules

34. When the deployment has successfully completed, choose the **Domain** URL in the middle of the Amplify console page.
35. To create an account for the bookmark application, choose **Create account**
36. Fill in the provided fields with your information and choose CREATE ACCOUNT
37. To add a bookmark, choose the plus **+** icon at the top right of the page.
38. Enter the **Name**, **Description**, and **Bookmark URL** of any website. Ensure the **Share Bookmark** toggle is set to **On**.
39. Choose **ADD BOOKMARK**.

Now, check that the CloudWatch logs are configured for the **catch-all** EventBridge rule.

40. Return to the AWS Management Console tab.

41. On the Services menu, choose **CloudWatch**.

42. In the left navigation pane, choose **Log groups**.

43. In the log groups list, choose the name of the **/aws/events/catch-all** log group.

Notice the log streams listed for this log group. This shows that logs are being generated for the **catch-all** EventBridge rule.

Check your email for an Amazon SNS notification with the details of the bookmark that you entered in the bookmark contest.

Task 3: Deploying the Lambda functions for the Step Functions state machine

In this task, you create a Lambda function that the Step Functions state machine needs. You also review the remaining Lambda functions that were set up with the AWS SAM deploy process.

44. In the AWS Management Console, on the Services menu, choose **Lambda**.

45. Choose Create function

46. Choose **Author from scratch**, and configure the following:

- **Function name:** Enter `duplicateBookmarkCheck`
- **Runtime:** Choose **Node.js 14.x**

47. Expand ▶ **Change default execution role**, and configure:

- **Execution role:** Choose **Use an existing role**
- **Existing role:** Choose **LambdaDeploymentRole**

48. Choose Create function

49. In the **Code source** section, select **index.js**, open the context(right-click) menu and choose **Open**.

50. Delete the existing code and paste the following code:

```
const AWS = require('aws-sdk');
const docClient = new AWS.DynamoDB.DocumentClient();
const dynamodb = new AWS.DynamoDB();

exports.handler = async message => {
   console.log(message);
   let bookmark = message;
```

```
    const bookmarkDetails = JSON.stringify(bookmark);
    console.log("bookmarkDetails are "+bookmarkDetails);
    const bookmarkItem = JSON.parse(bookmarkDetails);
    console.log("bookmarkItem "+bookmarkItem);
    console.log("url is "+bookmarkItem.detail.payload.bookmarkUrl.S);
    var exists = false;
    try{
        if(message != null)
         {
             var params = {
               TableName: process.env.TABLE_NAME,
               IndexName: process.env.INDEX_NAME,
               KeyConditionExpression: "bookmarkUrl = :keyurl",
               ExpressionAttributeValues: {
                ":keyurl": {"S": bookmarkItem.detail.payload.bookmarkUrl.S}
               }
             };
            console.log("exists "+exists);
            var result = await dynamodb.query(params).promise();
            console.log("result is "+JSON.stringify(result.Items));
            var data = JSON.parse(JSON.stringify(result.Items));

             data.forEach(function(item) {
                 console.log("db username", item.username.S+"
"+bookmarkItem.detail.payload.username.S);
                 if (item.username.S != bookmarkItem.detail.payload.username.S)
                     exists = true;
                 });

            console.log(exists);
            if (exists === true)
            {
              console.log("in here");
              var updateParams = {
              TableName: process.env.TABLE_NAME,
              Key:{
                  "id": bookmarkItem.detail.payload.id.S
              },
              UpdateExpression: "set contest=:c",
              ExpressionAttributeValues:{
                  ":c": "duplicate"
              },
              ReturnValues:"UPDATED_NEW"
            };
            await docClient.update(updateParams, function(err, data) {
              if (err) {
                console.log("Unable to update item. Error JSON:", JSON.stringify(err,
null, 2));
              }
              else {
                console.log("UpdateItem succeeded:", JSON.stringify(data, null, 2));
              }
            }).promise();
            return "Duplicate";
            }
        }
     }

    catch(e){
      console.log(e);
    }
     return "NotDuplicate";
```

```
};
```

51. Choose Deploy

You should see a message that says **Successfully updated the function duplicateBookmarkCheck.**

52. Choose the **Configuration** tab to configure the environment variables.

53. In the left navigation pane, choose **Environment variables**.

54. In the **Environment variables** section, choose Edit

55. In the **Edit environment variables** page, configure the following details:

- Choose Add environment variable
  - **Key:** Enter `INDEX_NAME`
  - **Value:** Enter `bookmarkUrl-index`
- Choose Add environment variable
  - **Key:** Enter `TABLE_NAME`
  - **Value:** Enter `bookmarksTable`

56. Choose Save

Note This Lambda function is the first function that runs as part of the Step Functions state machine. The function checks whether a new bookmark URL is a duplicate.

57. In the breadcrumbs at the top left of the page, choose **Functions**.

Review the following functions, which are used in the Step Functions workflow:

- **validateURL:** This function validates if the user entered a valid URL.
- **userApprovalEmail:** This function generates a URL for human approval or rejection and notifies the **ContestTopic**.
- **userApproval:** This function gets the approve or reject value after the user chooses the URL generated from the **userApprovalEmail** function.
- **publishApproval:** If the URL is approved, this function updates the DynamoDB table with the approved value.

Note To see all of these functions, you may need to go to the second page of the functions list.

Task 4: Setting up the state machine workflow invoked by the EventBridge rule

In this task, you set up a Step Functions state machine and an EventBridge rule. The EventBridge rule will invoke the state machine that you create.

58. In the AWS Cloud9 IDE, choose the ⊕ button and choose **New File**.

59. Copy the following Step Functions definition into the **New File**.

```
{
  "Comment": "Publish Rule workflow",
  "StartAt": "CheckDuplicates",
  "States": {
    "CheckDuplicates": {
      "Type": "Task",
      "Resource": "<duplicateBookmarkCheckArn>",
      "InputPath":"$",
      "ResultPath":"$.extractedMetadata",
      "Catch": [
          {
            "ErrorEquals": [ "States.ALL" ],
            "Next": "NotifyUser"
          }
        ],
      "Next": "DuplicateChoiceState"
    },
    "DuplicateChoiceState": {
        "Type": "Choice",
        "Choices": [
          {
            "Variable": "$.extractedMetadata",
            "StringEquals": "NotDuplicate",
            "Next": "NotDuplicate"
          },
          {
            "Variable": "$.extractedMetadata",
            "StringEquals": "Duplicate",
            "Next": "Duplicate"
          }
        ]
      },
    "NotDuplicate": {
      "Type": "Pass",
      "Next": "ValidateURL"
    },
    "Duplicate": {
      "Type": "Pass",
      "Next": "NotifyUser"
    },
    "ValidateURL":{
      "Type": "Task",
      "Resource": "<validateURLArn>",
      "InputPath":"$",
      "ResultPath":"$.extractedMetadata",
      "Catch": [
          {
            "ErrorEquals": [ "States.ALL" ],
            "Next": "NotifyUser"
          }
```

```
        ],
      "Next": "ValidateChoiceState"
      },
     "ValidateChoiceState": {
        "Type": "Choice",
        "Choices": [
          {
            "Variable": "$.extractedMetadata",
            "StringEquals": "Valid",
            "Next": "Valid"
          },
          {
            "Variable": "$.extractedMetadata",
            "StringEquals": "Invalid",
            "Next": "Invalid"
          }
        ]
      },
    "Valid": {
      "Type": "Pass",
      "Next": "UserApprovalEmail"
    },
    "Invalid": {
      "Type": "Pass",
      "Next": "NotifyUser"
    },
    "UserApprovalEmail": {
      "Type": "Task",
      "Resource": "arn:aws:states:::lambda:invoke.waitForTaskToken",
      "InputPath":"$",
      "ResultPath":"$.extractedMetadata",
      "Parameters": {
        "FunctionName": "<userApprovalEmailArn>",
        "Payload": {
          "ExecutionContext.$": "$$",
          "APIGatewayEndpoint": "https://<API_GATEWAY_ID>.execute-
api.<AWS_REGION>.amazonaws.com/dev/bookmarks/approval"
        }
      },
      "Next": "ManualApprovalChoiceState"
      },
     "ManualApprovalChoiceState": {
      "Type": "Choice",
      "Choices": [
        {
          "Variable": "$.extractedMetadata.Status",
          "StringEquals": "Approved",
          "Next": "PublishApproval"
        },
        {
          "Variable": "$.extractedMetadata.Status",
          "StringEquals": "Rejected",
          "Next": "NotifyUser"
        }
      ]
    },
   "PublishApproval":{
    "Type": "Task",
    "Resource": "<publishApprovalArn>",
    "InputPath": "$",
    "ResultPath": "$.extractedMetadata",
    "End": true
    },
```

```
    "NotifyUser": {
        "Type": "Task",
         "Resource": "arn:aws:states:::sns:publish",
         "InputPath": "$",
         "Parameters": {
           "TopicArn": "<ContestTopicArn>",
           "Message.$": "$",
           "MessageAttributes": {
             "msg": {
               "DataType": "String",
               "StringValue": "additional instructions!"
             }
           }
         },
         "End": true
        }
   }
}
```

60. At the top of the page, choose **File > Save** to save the file as `statemachine.json` in the **app-code** folder.

61. Review the Step Functions definition and see how the Lambda functions and the other components are plugged in.

62. In the AWS Cloud9 terminal, run the following AWS CLI and bash commands to replace the Lambda Function ARNs, the API Gateway Endpoint URL and the SNS Topic ARN in the above Step Functions definition.

```
export duplicateBookmarkCheckArn=$(aws lambda get-function --function-name
duplicateBookmarkCheck | jq '.Configuration.FunctionArn' | tr -d '"')
export validateURLArn=$(aws lambda get-function --function-name validateURL | jq
'.Configuration.FunctionArn' | tr -d '"')
export userApprovalEmailArn=$(aws lambda get-function --function-name
userApprovalEmail | jq '.Configuration.FunctionArn' | tr -d '"')
export publishApprovalArn=$(aws lambda get-function --function-name publishApproval |
jq '.Configuration.FunctionArn' | tr -d '"')
export API_GATEWAY_ID=$(aws apigateway get-rest-apis --query 'items[?name==`Bookmark
App`].id' --output text)
export ContestTopicArn=$(aws sns list-topics  | jq '.Topics[1].TopicArn' | tr -d '"')
sed -Ei "s|<duplicateBookmarkCheckArn>|${duplicateBookmarkCheckArn}|g"
statemachine.json
sed -Ei "s|<validateURLArn>|${validateURLArn}|g" statemachine.json
sed -Ei "s|<userApprovalEmailArn>|${userApprovalEmailArn}|g" statemachine.json
sed -Ei "s|<publishApprovalArn>|${publishApprovalArn}|g" statemachine.json
sed -Ei "s|<API_GATEWAY_ID>|${API_GATEWAY_ID}|g" statemachine.json
sed -Ei "s|<ContestTopicArn>|${ContestTopicArn}|g" statemachine.json
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" statemachine.json
```

63. At the top of the page, choose **File > Save** to save the file.

64. In the AWS Management Console, on the Services menu, choose **Step Functions**.

65. In the left navigation pane, choose the menu icon (☰), and then choose **State machines**.

66. Choose Create state machine
67. In the **Choose authoring method** page, configure the following:

- Choose **Write your workflow in code**
- **Type:** Choose **Standard**

68. In the **Definition** section, copy the Step Functions definition from the statemachine.json file and replace the sample code. You should see the code and the workflow diagram populated like the following image:

69. Choose Next
70. On the **Specify details** page, configure the following:

- **State machine name:** Enter `PublishStateMachine`
- **Permissions:** Choose **Choose an existing role**
- **Existing roles:** Choose **EventBridgeStateMachineRole**

71. Choose Create state machine

You can also view the state machine in the Workflow studio.

72. Choose Edit at the top most corner of the console.
73. In the **Definition** pane of the console, choose **Workflow Studio** New
74. Choose **Center** on the top to see the workflow in the middle pane of the console. It should look similar to the following image.