Try-it-out Exercises - Day 1 v1.0.2

Monday, March 29, 2021 11:02 AM

Try-it-out Exercises - Day 1

8 hours Free ★★★★★ Rate Lab

aws training and certification

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

Overview

During the course, you use this environment to interact with the AWS Management Console and some of the services that the course discusses. The environment will be available to you throughout the day but will be reset for the following day's class.

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)

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

Start Lab

1. At the top of your screen, launch your lab by choosing Start Lab

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

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

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

2. Open your lab by choosing Open Console

   This opens an AWS Management Console sign-in page.

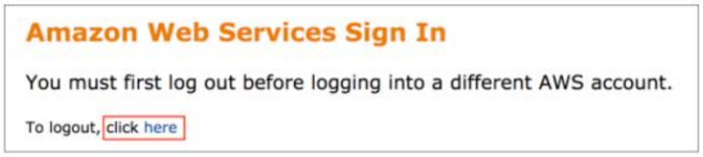
3. On the sign-in page, configure:

   - **IAM user name:** `awsstudent`
   - **Password:** Paste the value of **Password** from the left side of the lab page
   - Choose Sign In

Do not change the Region unless instructed.

Common Login Errors

Error: You must first log out



Amazon Web Services Sign In

You must first log out before logging into a different AWS account.

To logout, click here

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

- Choose **click here**
- Close your browser tab to return to your initial lab window
- Choose Open Console again

Module 2

Use the Amazon API Gateway console to build an HTTP API integrated with an AWS Lambda function

In this task, you build an HTTP API with a GET route that is integrated with an Lambda function called **bakeryFunction**. When you call the API endpoint in your browser, the function will randomly suggest a snack pick that you might buy from the bakery.

Create the API

4. Open the API Gateway console, and choose Build to build an HTTP API.

5. Enter the API name of your choice (the examples provided use the API name `Bakery`). Choose Review and Create and then Create

Create a route

6. From the **Navigation** menu, choose **Routes**, and then choose Create

7. Set the method to **GET**, name the path whatever you like (for example, `snackpick`), and choose Create

Add the Lambda integration

8. Choose the **GET** route you just created, and then choose Attach integration and

then Create and attach an integration

9. Set the **Integration type** to **Lambda function**.

10. In the **Integration details** section, under **Lambda function**, choose **bakeryFunction**.

    Note Notice the **Invoke permissions** section with the option to **Grant API Gateway permission to invoke your Lambda function**. Leave this turned on. This automatically creates the permissions your API needs to invoke the **bakeryFunction** function.

11. Choose Create so that your method is automatically deployed to the **$default** stage. Under **Deploy** for the API, navigate to the **Stages** section to find the **Invoke URL** for the **$default** stage.

    Test the endpoint

12. Copy the URL into your browser, and add your path name to the end of it (for example, https://4o8lnbzky4.execute-api.us-west-2.amazonaws.com/snackpick). When you invoke it, you should get a response that says something like the following:

```
Here is my next snack pick: Doughnut
(Refresh the page for the next pick)
```

    Note You can manage multiple versions of your API using stages. When your HTTP API is created, the **$default** stage is created and set to autodeploy.

    Usually, the **$default** stage is your production API, so you may turn off autodeploy on the **$default** stage and create another stage (for example, **dev**) for testing your updates before you deploy the production stage. For the lab, leave **$default** set to autodeploy.

13. On the **Stages** panel, choose Create to add a new stage. Give your new stage a name, turn on the **Enable automatic deployment** option, and choose **Create**.

    Note the difference in the **Invoke URL** for the **dev** stage versus the **$default** stage.

    Review the Lambda permissions associated with your API

14. Go to the Lambda console, and choose **bakeryFunction**. In the **Function overview** section, you should find the API Gateway trigger listed.

15. If you go to the **Configuration** tab, choose **Permissions** in the left navigation pane, and scroll to the **Resource-based policy** section, you should find the policy that was created automatically when you created the Lambda integration in your API. To view the policy, choose View policy document

    It includes a policy statement similar to the following example.

```
...
"Effect": "Allow",
"Principal": {
  "Service": "apigateway.amazonaws.com"
},
"Action": "lambda:InvokeFunction",
"Resource": "arn:aws:lambda:us-west-
2:123456789012:function:bakeryFunction",
"Condition": {
  "ArnLike": {
    "AWS:SourceArn": "arn:aws:execute-api:us-west-
2:123456789012:4o8lnbzky4/*/*/snackpick"
  }
}
...
```

Optional task: Build and deploy a REST API from an example, and then generate the SDK for the API

You can do this task later on your own AWS account.

16. From the API Gateway console, choose **Create API**

17. On the REST API panel, choose the **Build** button. Then choose the **Example API** radio button, and choose **Import**

    The PetStore API is created in your account, and you can look at the details in the **Resources** section.

18. REST APIs do not autodeploy, and there is not a **$default** stage. To deploy the example API, choose **Actions** and then **Deploy API**. Choose **[New Stage]** for the **Deployment stage**, give your stage a name such as `dev` and choose **Deploy**

19. To generate an SDK for this API, choose the stage you just deployed (for example, **dev**), select the **SDK Generation** tab, and choose a platform. Choose **Generate SDK** and save the generated file.

Module 5

Use the Amazon EventBridge console to build a custom event bus and add a custom event rule

In this task, you set up an event bus that could be used to route events to different Lambda functions based on the type of snack indicated in the event. You will set up a rule that routes events about the snack type **Pie** to a Lambda function called **pieFunction** and add a POST route to the HTTP API you set up earlier that is attached to your custom EventBridge bus (serverless-bus). Finally, you'll test it via the command line in AWS Cloud9 and verify the results using Amazon CloudWatch Logs.

Create the event bus

Go to the EventBridge console, and create a new event bus called **serverless-bus**.

20. From the EventBridge console menu, choose **Event buses**.

21. In the **Custom event bus** panel, choose **Create event bus**

22. Name the event bus `serverless-bus` and choose **Create**

Create the rule

The rule should route only events that include the snack type **Pie** in the detail of the event to the **pieFunction** function. Events without the snack type **Pie** will not be routed to the function.

23. From the EventBridge console menu, choose **Rules**. In the **Select event bus** section, make sure **serverless-bus** is selected, and then choose **Create rule**

24. Configure the following information to set up the rule:

    - For **Name**, enter `pie-rule`
    - Select **Event pattern**.
    - Select **Custom pattern**.
    - In the **Event pattern** text area, copy and paste the following rule code, and choose **Save**

```
{
  "source": [
    "Bakery Store"
  ],
  "detail-type": [
    "snacks"
```

{ "source": ["Bakery Store"], "detail-type": ["snacks"], "detail": { "snack-type": ["Pie"] } }

```
    ],
    "detail": {
      "snack-type": [
        "Pie"
      ]
    }
  }
```

- In the **Select event bus** section, choose **Custom or partner event bus**, and choose **serverless-bus**.
- In the **Select targets** section, choose **Lambda function** as the target, and choose **pieFunction**.
- Choose **Create**

Create the API route

If you ended this lab at any point during the day and lost the API you built in Module 2, you can use the prebuilt **Bakery-Sample-Api** for the following steps.

25. Open your **Bakery** API Gateway HTTP endpoint, and add a new route with method type **POST** and a path name of **newsnack**.

26. With POST selected, choose **Attach integration** and then **Create and attach an integration**

27. In the **Integration type** dropdown list, select **Amazon EventBridge**.

28. Select **PutEvents** for the Integration action, and then configure the following information:

    - **Detail:** Enter `$request.body.Detail`
    - **Detail Type:** Enter `snacks`
    - **Source:** Enter `Bakery Store`
    - **Invocation role:** From the left navigation of the lab page, copy the **APIGatewayInvocationRole** and paste the value into the field.

    Under **Advanced settings**, configure the following information:

    - **Event bus name:** Enter `serverless-bus`

29. Choose **Create**

    This update is automatically deployed to the **$default** stage.

Test the rule

30. Open your AWS Cloud9 environment.

31. Copy the **Invoke URL** for the **$default** stage of your API.

32. In the following command, replace *(API-GATEWAY-ENDPOINT-URL)* with the **Invoke URL**. Run the updated command in the AWS Cloud9 terminal.

```
curl --location --request POST '(API-GATEWAY-ENDPOINT-URL)/newsnack' \
--header 'Content-Type: application/json' \
--data-raw '{"Detail": {"snack-type": "Pie","snack-name": "Apple Pie"}}'
```

The above call to the API directly invokes EventBridge, which in turn invokes the **pieFunction** function. Your command line response should look similar to the following:

```
{"Entries":[{"EventId":"29f6c069-4891-5188-e5a6-
69a514eb30db"}],"FailedEntryCount":0}
```

33. In the Lambda console, open the **pieFunction** function.

34. Choose the **Monitor** tab, and then select **View logs in CloudWatch**.

35. Choose the most recent log stream, and look for the event details that you included

35. Choose the most recent log stream, and look for the event details that you included in your POST request.

36. Test the rule with another API call, but this time change the snack type to something other than **Pie**. This change should not result in an invocation of the Lambda function. Verify within CloudWatch Logs.

Use the Amazon SNS console to subscribe a Lambda function to a topic using attribute filtering

In this task, you set up an Amazon Simple Notification Service (Amazon SNS) topic that could be used to provide subscribers with snack event information. You'll subscribe the **pieFunction** function to the topic and configure attribute filtering on the subscription so that only events that include the snack type of **Pie** are delivered to the **pieFunction** subscriber. You'll also use the Amazon SNS console to create test messages and test the subscription filter policy.

Create an Amazon SNS topic with a Lambda function subscriber and a filter policy

37. Open the Amazon SNS console. From the console navigation menu, choose **Topics**, and then choose **Create topic**

38. For the topic type, select **Standard**, give your topic a name, and choose **Create topic**

39. From the topic details page, choose **Create subscription**

40. Under **Protocol**, select **AWS Lambda**, and then choose the **pieFunction** function as the endpoint.

41. Under **Subscription filter policy**, add the following filter:

```
{
  "type": [
    "Pie",
    "pie",
    "Pies",
    "pies"
  ]
}
```

42. Choose **Create subscription**

Test the filter policy

First, publish a message that doesn't have attributes that match the filter.

43. On the topic you created, choose **Publish message** and then configure the following information:

- **Subject**: Enter `New Bakery Items`
- **Message body**: Enter `A new donut flavor has been added in the bakery store`

Under **Message attributes**, configure the following information:

- **Type:** Choose **String** from the dropdown list
- **Name:** Enter `type`
- **Value:** Enter `Donut`

44. Choose **Publish message**

Now, publish a message with attributes that match the filter.

45. On the topic you created, choose **Publish message** and then configure the following information:

- **Subject**: Enter `New Bakery Items`
- **Message body**: Enter

[illegible]

`A new pie flavor has been added in the bakery store`

Under **Message attributes**, configure the following information:

- **Type:** Choose **String** from the dropdown list
- **Name:** Enter `type`
- **Value:** Enter `Pie`

46. Choose **Publish message**

47. Go to CloudWatch Logs for the **pieFunction** function, and open the most recent log stream. You should find the invocation for the message that included **Pie** in the attribute but not for the message with the **Donut** attribute.

Module 6

Use the Amazon SQS console to set up a queue as a Lambda event source

In this task, you use the Amazon Simple Queue Service (Amazon SQS) console to set up a queue as a Lambda event source and use the console features to perform basic testing with a dead-letter queue.

Set up the queue and its dead-letter queue

48. Open the Amazon SQS console, and choose **Create queue**

49. Name the queue `donut-queue` and choose **Create queue**

50. Choose **Edit**

51. Scroll to the **Dead-letter queue** section, and switch the radio button to **Enabled**.

52. In the **Queue ARN** dropdown list, select **donut-dlq**.

53. Set **Maximum receives** to **2**.

54. Choose **Save**

Test the dead-letter queue

Note If you were to configure the Lambda function trigger before you run this test, Lambda would pick up and process the messages before you could interact with them on the console.

55. With your **donut-queue** details on the console, select **Send and receive messages**

56. Enter any simple message, and choose **Send message**

57. Modify each message body to distinguish them (for example, **test message 2** and **test message 3**), and add a couple more messages.

58. Scroll to the **Receive messages** section, and choose **Poll for messages**

Your messages will appear in the list. Note the receive count.

59. Choose **Poll for messages** again. The receive count increments.

60. Choose **Poll for messages** a third time.

The list of messages in the queue should exclude any messages with a maximum receive count higher than two. You can now find those messages in the dead-letter queue.

Configure and test the Lambda trigger on the donut-queue

61. Open the **donut-queue** queue, select the **Lambda triggers** tab, and choose Configure Lambda function trigger
62. From the dropdown list, choose **donutFunction**.
63. Choose Save
64. On the **donut-queue**, choose the **Send and receive messages** option again, and send another message.
65. Find the CloudWatch logs for the **donutFunction**, and verify that the function was invoked with the message from the Amazon SQS queue.

End Lab

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

66. Return to the AWS Management Console.
67. On the navigation bar, choose **awsstudent@<AccountNumber>**, and then choose **Sign Out**.
68. Choose End Lab
69. Choose OK
70. (Optional):

- Select the applicable number of stars ☆
- Type a comment
- Choose **Submit**

  - 1 star = Very dissatisfied
  - 2 stars = Dissatisfied
  - 3 stars = Neutral
  - 4 stars = Satisfied
  - 5 stars = Very satisfied

You may close the window if you don't want to provide feedback.

For more information about AWS Training and Certification, see http://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*.