Building a Serverless API

**SPL-BE-100-CEBSAP-1 - Version 1.0.2**

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

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

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

**Lab overview**

In this Amazon Web Services (AWS) lab, you gain hands-on experience in deploying a serverless application programming interface (API) by using the AWS Serverless Application Model (AWS SAM). You are guided through the process of understanding the purpose of each file in the Python-based source code, deploying a basic AWS Lambda function, and ultimately implementing a more advanced function to read the contents of an Amazon Simple Storage Service (Amazon S3) object.

OBJECTIVES

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

* Access the preconfigured AWS Cloud9 instance with the preloaded application code.
* Review the three application files (app.py, requirements.txt, template.yaml) and their purposes.
* Build and deploy the application by using the *sam build* and *sam deploy --guided* commands.
* Visit the API endpoint in a browser and used client URL (curl) commands to view the initial static text response.
* Modify the *app.py* file so that it uses the AWS SDK for Python (Boto3) to retrieve the contents from the *object1* file and include the file contents in the Lambda response.
* Redeploy the updated application and verify that it returns the contents of the object1 file.

TECHNICAL KNOWLEDGE PREREQUISITES

To successfully complete this lab, you should be:

* Familiar with the basic navigation of the AWS Management Console.
* Versed in editing and running scripts by using an AWS Cloud9 code editor and terminal.
* Have a basic understanding of AWS SAM, Amazon API Gateway, AWS Lambda, AWS CodePipeline, AWS CodeDeploy, and AWS CloudFormation.

DURATION

This lab requires *30* minutes to complete.

ICON KEY

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

* **Command:** A command that you must run.
* **Expected output:** A sample output that you can use to verify the output of a command or edited file.
* **Note:** A hint, tip, or important guidance.
* **Consider:** A moment to pause to consider how you might apply a concept in your own environment or to initiate a conversation about the topic at hand.
* **Task complete:** A conclusion or summary point in the lab.

**Start lab**

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

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

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

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

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

COMMON SIGN-IN ERRORS

**Error: You must first sign out**



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

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

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

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

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

**Task 1: Review the application files**

In this task, you review the three application files to learn what they do and how they are used in the overall application.

TASK 1.1: CONNECT TO AWS CLOUD9

AWS Cloud9 is a cloud-based integrated development environment (IDE) that you can use to write, run, and debug your code with only a browser. It includes a code editor, debugger, and terminal. AWS Cloud9 comes prepackaged with essential tools for popular programming languages, including JavaScript, Python, PHP, and more. You don’t need to install files or configure your development machine to start new projects.

In this task, you connect to the AWS Cloud9 environment that’s provisioned as part of this lab.

1. From the **Lab Information** section to the left of these instructions, copy the **Cloud9url** URL link and in a new browser tab, paste the link.

The browser takes you to the AWS Cloud9 environment that you use during this lab.

You don’t need the **AWS Cloud9 Welcome screen** or any of the other default tabs that appear when you first launch **AWS Cloud9**.

1. Close each tab by choosing the **X**.

This section of the IDE is where you view and update various files throughout this lab.

**Consider:** Take a moment to familiarize yourself with the **AWS Cloud9** IDE interface.

* In the middle of the screen, a single terminal session is open in the editor. You can open multiple tabs in this window to edit files and run terminal commands.
  + The file navigator is on the left side of the screen.
  + A gear icon is on the right side of the screen. Choosing this icon opens the AWS Cloud9 **Settings** panel.

**Note:** Every *AWS Cloud9* workspace is automatically assigned *AWS Identity and Access Management (IAM)* credentials. These credentials provide the workspace with limited access (based on your federated role) to some AWS services in your account. These are known as AWS managed temporary credentials.

TASK 1.2: REVIEW THE APPLICATION FILES

In this task, you review the application files, and learn about the main sections and what they do.

The *terminal pane* is at the bottom of the AWS Cloud9 IDE. You can expand the pane halfway up the screen to have more visibility when you run commands. You can also close it and open a new terminal session from the top menu. (To open a new terminal session, choose the  icon and choose *New Terminal*.)

1. From the file tree, expand the **LabFunction** folder.
2. To review the code, open **app.py**.

This Python code is an AWS Lambda function that uses the SDK for Python library to interact with Amazon S3, a cloud storage service. Your challenge is to see if you can spot the update that it needs to read the contents of an object.

* It imports the *boto3* library, which is the Amazon Web Services (AWS) SDK for Python.
* It creates an Amazon S3 client by using *boto3.client(‘s3’)*, which is used to interact with the Amazon S3 service.
* It defines a function called *lambda\_handler* that takes two arguments: *event* and *context*. This function is invoked when the Lambda function is run.
* Inside the *lambda\_handler* function, it retrieves an object from the Amazon S3 bucket with the name that starts with *sourcefiles* and the key *object1*.
* It reads the data stored in the *contents* variable, which is only static text at this time.
* The function returns a dictionary that contains the key “result” and the data stored in the *contents* variable.

In summary, this Lambda function currently retrieves an object from an Amazon S3 bucket, and it also retrieves the contents of a specific variable. The function then returns the static text as a result.

**Consider:** Does this scenario appear unusual to you? You might be questioning the reason why a Lambda function fetches an object from an Amazon S3 bucket and returns the contents of a variable with static text, instead of the object1 file’s contents. If so, you’ve identified the section of this script that needs to be updated. This update is performed in an upcoming task.

1. To review the requirements for the application, open **requirements.txt**.

In this lab, the *requirements.txt* file contains *boto3* as a requirement because the lab uses SDK for Python *boto3* library to interact with AWS resources.

*boto3* is the official AWS SDK for Python. You can use it to access and manage AWS services such as Amazon Simple Storage Service (Amazon S3), Amazon Elastic Compute Cloud (EC2), Amazon DynamoDB, and many others. By including *boto3* in the *requirements.txt* file, the *boto3* library is installed and available when you run the code in the lab. Because it’s already installed, *boto3* library makes it possible for you to use AWS resources seamlessly within the Python code.

1. To review the resources that are created for this application, open **template.yaml**.

This template is a YAML Ain’t Markup Language (YAML) file that defines an AWS CloudFormation stack by using the AWS Serverless Application Model (AWS SAM) extension. The template creates a serverless application with the following resources:

*AWSTemplateFormatVersion*: This setting specifies the AWS CloudFormation template version that the template conforms to.

*Transform*: This directive instructs CloudFormation to apply the specified transform to the template. In this case, it’s using the AWS SAM transform.

*Resources*:

* *LabFunction*: This is a Lambda function that’s defined using the AWS::Serverless::Function resource type. The properties of this function are:
  + *CodeUri*: This specifies the location of the Lambda function’s source code.
  + *Handler*: This specifies the entry point of the Lambda function.
  + *Runtime*: This specifies the runtime environment for the Lambda function (Python 3.9).
  + *Architectures*: This specifies the architecture for the Lambda function (x86\_64).
  + *Policies*: This specifies the IAM policies (which are attached to the Lambda function), that allow the function to be read from the specified Amazon S3 bucket.
  + *Events*: Specifies the event sources for the Lambda function. In this case, it’s an HTTP API event source with a GET method.
* *HttpApi*: This is an HTTP API that’s defined using the AWS::Serverless::HttpApi resource type. The property **StageName** is set to “$default”.
* *Outputs*: This contains the output values that you can import into other stacks or return in the AWS Management Console.
  + *HttpApiUrl*: Provides the URL of the API endpoint, which is a combination of the HttpApi resource, AWS Region, and AWS URL suffix.

**Task complete:** You have successfully connected to the AWS Cloud9 environment and familiarized yourself with the interface. You reviewed the three application files that help make up the application to understand what they do and how they are used. You also identified the update that’s required to read the contents of the object1 file.

**Task 2: Build and deploy the application**

After examining the application code and learning about the functionality of the files in the development process, you are now ready to use the AWS Serverless Application Model (AWS SAM) to construct the application package and subsequently deploy the application within the AWS infrastructure. Both *sam build* and *sam deploy* are commands that you can use in the AWS SAM command line interface (CLI), a tool that helps you manage and deploy serverless applications on AWS. These commands perform specific tasks in the development and deployment processes of your serverless application.

*sam build:* The *sam build* command is used to compile and package your serverless application’s source code and its dependencies. It processes your AWS SAM template, which defines the application’s resources and configuration, and creates a build directory with the compiled artifacts (e.g., Lambda functions, layers, etc.) and transformed template.

The build process includes the following steps:

* Reading the AWS SAM template and identifying the resources to build.
* Building each resource (such as a Lambda function) by downloading the required dependencies, compiling the source code (if necessary), and creating a deployment package (such as a .zip file).
* Generating a new AWS SAM template with the updated resource references (for example, pointing to the built artifacts).

After running *sam build*, your serverless application is ready for deployment or local testing by using *sam local* commands.

*sam deploy:* The *sam deploy* command is used to deploy your serverless application to AWS. It takes the built artifacts and transformed template that’s generated by *sam build*, and creates or updates the necessary AWS resources.

The deployment process includes the following steps:

* Packaging the built artifacts and uploading them to an S3 bucket. This step can be skipped if the artifacts are already packaged and uploaded by using the *sam package* command.
* Deploying the application by using AWS CloudFormation. It creates a new CloudFormation stack, or updates an existing one with the resources that are defined in the transformed AWS SAM template. This step creates or updates AWS resources, such as Lambda functions, API Gateway endpoints, DynamoDB tables, and others.

During the deployment, you can specify various options, such as the stack name, deployment Region, parameter overrides, and more.

TASK 2.1 BUILD THE DEPLOYMENT PACKAGE

The first step is to compile and build source code into a deployment package. To compile the deployment package, the terminal session needs to be in the root environment folder.

1. **Command:** If your terminal isn’t already in the **~/environment** directory, change directories by running the following command:

cd ~/environment

**Expected output:**

*None, unless an error occurs.*

1. **Command:** Build the deployment package by running the following command:

sam build

**Expected output:** Output has been truncated.

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

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

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

Building codeuri: /home/ec2-user/environment/LabFunction runtime: python3.9 metadata: {} architecture: x86\_64 functions: LabFunction

Running PythonPipBuilder:ResolveDependencies

Running PythonPipBuilder:CopySource

Build Succeeded

TASK 2.2: DEPLOY THE APPLICATION

Now that you’ve created the deployment package, deploy the application.

1. **Command:** Deploy the application with the following command:

sam deploy --guided

You are prompted with a series of questions.

1. Respond to the questions with the following options:

* **Command:** For the **Stack Name [sam-app]:**, accept the default value by pressing **Enter**.

**Warning**

The *stack name* needs to be set to *sam-app* because the lab policies are created based on this name. If you use a different name, the remaining lab steps won’t work properly.

* **Command:** For **AWS Region [us-west-2]:**, accept the default value by pressing **Enter**.
* **Command:** For **Confirm changes before deploy [y/N]:** accept the default answer by pressing **Enter**.
* **Command:** For **Allow SAM CLI IAM role creation [Y/n]:**, accept the default answer by pressing **Enter**.
* **Command:** For **Disable rollback [y/N]:**, accept the default answer by pressing **Enter**.
* **Command:** For **LabFunction may not have authorization defined, Is this okay? [y/N]:**, enter

y

 and press **Enter**.

* **Command:** For **Save arguments to configuration file [Y/n]:**, accept the default answer by pressing **Enter**.
* **Command:** For **SAM configuration file [samconfig.toml]:**, accept the default value by pressing **Enter**.
* **Command:** For **SAM configuration environment [default]:**, accept the default value by pressing **Enter**.

**Expected output:** Output has been truncated.

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

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

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

Configuring SAM deploy

======================

Looking for config file [samconfig.toml] : Not found

Setting default arguments for 'sam deploy'

=========================================

Stack Name [sam-app]:

AWS Region [us-west-2]:

#Shows you resources changes to be deployed and require a 'Y' to initiate deploy

Confirm changes before deploy [y/N]:

#SAM needs permission to be able to create roles to connect to the resources in your template

Allow SAM CLI IAM role creation [Y/n]:

#Preserves the state of previously provisioned resources when an operation fails

Disable rollback [y/N]:

LabFunction may not have authorization defined, Is this okay? [y/N]: y

Save arguments to configuration file [Y/n]:

SAM configuration file [samconfig.toml]:

SAM configuration environment [default]:

Looking for resources needed for deployment:

Managed S3 bucket: aws-sam-cli-managed-default-samclisourcebucket-1u3fu7o3gelcq

A different default S3 bucket can be set in samconfig.toml

Saved arguments to config file

Running 'sam deploy' for future deployments will use the parameters saved above.

The above parameters can be changed by modifying samconfig.toml

Learn more about samconfig.toml syntax at

https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/serverless-sam-cli-config.html

Uploading to sam-app/6ef47c0da3e463d20af67dde5dfed4ef 11580021 / 11580021 (100.00%)

Deploying with following values

===============================

Stack name : sam-app

Region : us-west-2

Confirm changeset : False

Disable rollback : False

Deployment s3 bucket : aws-sam-cli-managed-default-samclisourcebucket-1u3fu7o3gelcq

Capabilities : ["CAPABILITY\_IAM"]

Parameter overrides : {}

Signing Profiles : {}

CloudFormation outputs from deployed stack

--------------------------------------------------------

Outputs

--------------------------------------------------------

Key HttpApiUrl

Description URL of your API endpoint

Value https://8874ni1cuc.execute-api.us-west-2.amazonaws.com/labfunction

--------------------------------------------------------

Successfully created/updated stack - sam-app in us-west-2

1. From the output of the *sam deploy* command, copy the **endpoint URL** value.
2. In a text editor, paste and save the URL so you can retrieve it later.

**Note:** You use this **URL** to test the API in the next task.

**Task complete:** You have successfully created the deployment package that encompasses your application’s source code. You have also deployed the application into the AWS environment by using AWS SAM commands.

**Task 3: Test the application**

In this task, you visit the API endpoint in a browser tab to see that the application has been successfully deployed. You also test the application from a terminal, using a *curl* command. Using a *curl* command to test an AWS API Gateway endpoint allows you to send HTTP requests to the API Gateway and receive a response. This can help you verify if the API Gateway is functioning correctly, if the integration with backend services (like AWS Lambda) is working as expected, and if the API is returning the expected response data. Both methods of testing the application should return a response of *These are not the contents you are looking for.* if the deployment was completed successfully.

Test the functionality of the application by using a web browser.

1. Copy the **endpoint URL** you saved earlier, open a new browser tab, paste the **URL** into the address bar and press the **Enter** key.

The browser returns the following text:

*{“result”: “These are not the contents you are looking for.”}*

Test the functionality of the application by using a *curl* command.

1. Return back to the browser tab that’s opened to the AWS Cloud9 environment.
2. **Command:** Test the application by running the following echo and curl commands. Be sure to replace the API\_URL\_HERE placeholder with the actual endpoint URL you copied earlier:

echo; curl API\_URL\_HERE; echo; echo

**Example Command:**

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

\*\*\*\* This is an EXAMPLE ONLY. \*\*\*\*

\*\*\*\*\*\*\*\*\*\* DO NOT COPY \*\*\*\*\*\*\*\*\*\*\*

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

echo; curl https://onw4cvuz5k.execute-api.us-west-2.amazonaws.com/labfunction; echo; echo

**Expected output:**

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

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

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

{"result": "These are not the contents you are looking for."}

**Note:** The echo commands were added to make reading the output easier, and aren’t required outside of aesthetics.

**Task complete:** You have successfully tested the functionality of the application by using a browser and a curl command to access the API endpoint. Both test methods return the data that’s stored in the *contents* variable, which is currently only static text.

**Task 4: Update the application and test the changes**

In this task, you update the application to read the contents of the object1 file’s body. The application then populates the *contents* variable with this code snippet, instead of the static text that’s currently being set in the *contents* variable.

**Consider:**

* *s3.get\_object()* is a method from the SDK for Python that retrieves an object from an S3 bucket. The *Bucket* and *Key* parameters specify the bucket name and the object key, respectively. In this case, the bucket name starts with *sourcefiles* and the object key is *object1*.
* The *get\_object()* method returns a dictionary that contains the object’s metadata and its body as a *botocore.response.StreamingBody* object. The *obj* variable holds this dictionary.
* *obj[“Body”]* accesses the StreamingBody object that’s associated with the ‘Body’ key in the *obj* dictionary.
* The *read()* method is called on the StreamingBody object to read the contents of the object.

CHALLENGE A - UPDATE THE APPLICATION CODE

Now that you know the application works, your challenge is to update the value for the *contents* variable with minimal instruction and no hints. Make the update in the best way that you can. Don’t worry, because the full solution is provided for you to refer to and verify against your own update.

1. If not already opened, open the **app.py** file.
2. Update the **contents** variable to use

obj["Body"].read()

 instead of the text that reads, **These are not the contents you are looking for.**

Once you made the edit to the **app.py** file, review the following solution to verify if you made the correct update.

**Solution**

1. Save the changes to the **app.py** file.

**Task complete:** You have successfully updated the app.py file to read the contents of the object1 file in the S3 bucket and update the *contents* variable with that code snippet. When the application runs, it replaces the value of the *contents* variable from the code snippet to the data from the object1 file.

TASK 4.1 - BUILD, DEPLOY, AND TEST THE UPDATED APPLICATION

You have made the necessary changes to the application. You can now run the *sam build* and *sam deploy* commands to deploy the latest version of the application, and then test again.

1. **Command:** To package the latest changes into a deployment file, run the following command:

sam build

**Expected output:** Output has been truncated.

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

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

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

Building codeuri: /home/ec2-user/environment/LabFunction runtime: python3.9 metadata: {} architecture: x86\_64 functions: LabFunction

Running PythonPipBuilder:ResolveDependencies

Running PythonPipBuilder:CopySource

Build Succeeded

Because you accepted the option to save the arguments to a configuration file, you can deploy the update by running only the *sam deploy* command. The AWS SAM CLI doesn’t ask you the series of questions again (like it did the first time you ran the command).

1. **Command:** Deploy the updates with by using the following command:

sam deploy

**Expected output:** Output has been truncated.

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

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

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

Successfully created/updated stack - sam-app in us-west-2

Test the functionality of the application using a web browser.

1. Copy the **endpoint URL** you saved earlier, open a new browser tab, paste the **URL** into the address bar and press the **Enter** key.

The browser returns the following text:

*{“result”: “Welcome to AWS Lambda!”}*

Test the functionality of the application using a *curl* command.

1. Return back to the browser tab that’s opened to the AWS Cloud9 environment.
2. **Command:** Test the application by running the following echo and curl commands. Be sure to replace the API\_URL\_HERE placeholder with the endpoint URL:

echo; curl API\_URL\_HERE; echo; echo

**Example:**

echo; curl https://onw4cvuz5k.execute-api.us-west-2.amazonaws.com/labfunction; echo; echo

**Expected output:**

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

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

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

{"result": "Welcome to AWS Lambda!"}

**Note:** The echo commands were added to make reading the output easier and aren’t required outside of aesthetics.

CHALLENGE B - VERIFY THE CONTENTS OF THE OBJECT1 FILE

You have updated the application code, and it now returns different output. Now, your challenge is to determine if it is the correct output. You can achieve this through a couple of different ways. Challenge yourself to see if you can figure it out on your own. If you get stuck, refer to the following hints.

1. The first way to verify the contents of the **object1** file is to use the Amazon S3 console to download the file and open it with a text editor.

**Hint**

The second option to verify the contents of the *object1* file is to copy the file using an AWS CLI s3 command.

**Learn more**

For more information about how to use the **cp** command as part of the AWS CLI, see *AWS CLI Command Reference: s3 cp* in the **Additional resources** section.

1. Try to figure out what command you can run to download the **object1** file and then open it to view its contents.

If you get stuck, refer to the following hint.

**Hint**

**Task complete:** You have successfully updated the application to retrieve the contents of the *object1* file. You verified the updates by using a browser and a curl command to access the API endpoint. Both testing methods return the data that’s stored in the *contents* variable, which is now the data from the *object1* file.

**Conclusion**

You now have successfully:

* Accessed the preconfigured AWS Cloud9 instance with the preloaded application code.
* Reviewed the three application files (app.py, requirements.txt, template.yaml) and their purposes.
* Built and deployed the application by using the *sam build* and *sam deploy --guided* commands.
* Visited the API endpoint in a browser and used client URL (curl) commands to view the initial static text response.
* Modified the *app.py* file so that it uses the AWS SDK for Python (Boto3) to retrieve the contents from the *object1* file and include the file contents in the Lambda response.
* Redeployed the updated application and verified that it returns the contents of the *object1* file.

**End lab**

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

1. Return to the **AWS Management Console**.
2. At the upper-right corner of the page, choose **AWSLabsUser**, and then choose **Sign out**.
3. Choose **End lab** and then confirm that you want to end your lab.

**Additional resources**

[AWS CLI Command Reference: s3 cp](https://docs.aws.amazon.com/cli/latest/reference/s3/cp.html)

For more information about AWS Training and Certification, see [*https://aws.amazon.com/training/*](https://aws.amazon.com/training/).

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