Lab Information

Contents

AWS Services are loading.

aws training and certification

Lab 5: Develop Solutions Using Amazon API Gateway

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* .

Duration

This lab will require **60 minutes** to complete.

Prerequisites

This lab requires:

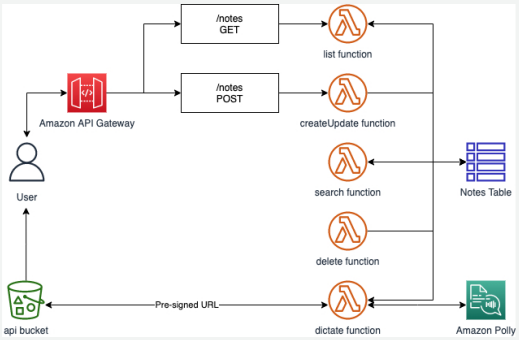
- Access to a Microsoft Windows or MacOS notebook computer with a WI-FI connection.
- An Internet browser such as Chrome, Firefox, or IE9+.
- **Note:** Previous versions of Internet Explorer are not supported.
- **Note:** You can use an iPad or tablet device to access these directions in the lab console.
- **Additional information:** Review additional lab environment specific details in the Appendix.

Overview

Now that you have the core backend set up for storage, database, and compute processing, you need a way for users to access the AWS Lambda functions over the internet.

In this lab, you will create an Amazon API Gateway resource for your application. You will then configure two methods for that resource to allow access to two of your AWS Lambda functions. With these methods you will validate the requests and transform the responses returned from the Lambda functions. Finally, you will configure Cross-Origin Request Sharing (CORS) headers to allow access for your web application that will be added in the next lab.

Once you have completed this lab, your deployment will look like the following diagram:



OBJECTIVES

After completing this lab, you will be able to:

- Create RESTful API resources and configure CORS for your application.
- Integrate API methods with AWS Lambda functions to process application data.
- Configure mapping templates to transform the pass-through data during method integration.
- Create a request model for API methods to ensure that the pass-through data format complies with application rules.
- Deploy the API to a stage and validate the results using the API endpoint.

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.

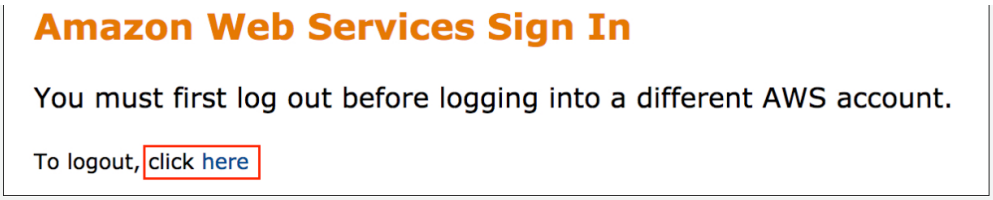
2. To open the lab, choose Open Console .

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: Create a REST API and resource

In this task you will develop a REST API and resource for your application.

Consider: This lab is designed for both experienced and newer developers:

- For more experienced developers who enjoy a challenge, there are **High-Level Instructions** before each task that should provide you enough information to help you complete the task.
- Once you complete the updates test your code to ensure it works, troubleshoot if needed, and then move on to the next task.
- For newer developers, there are **Detailed Instructions** to guide you through each step of the lab.

High-Level Instructions

- Create a new `PollyNotesAPI` Regional REST API
- Create a `/notes` resource with CORS enabled

Detailed Instructions

TASK 1.1: CREATE THE AMAZON API GATEWAY AND RESOURCE

There are several different types of gateways that you can create with Amazon API Gateway. For this application, you will create a REST API.

An Amazon API Gateway REST API is a collection of HTTP resources and methods that are integrated with backend HTTP endpoints, Lambda functions, or other AWS services. You can deploy this collection in one or more stages. Typically, API resources are organized in a resource tree according to the application logic. Each API resource can expose one or more API methods that have unique HTTP verbs supported by API Gateway.

3. From the **AWS Management Console**, use the **AWS search bar** to search for `API Gateway` and then choose the service from the list of results.

Note: This lab uses the **latest** API Gateway interface. If you find a dialog prompting you to switch to the new console, choose **Try out the new console**.

4. For **Choose an API type**, on the **REST API** card, choose **Build**.

CAUTION: Be sure not to choose the option reading **REST API** Private.

5. Make the following selections:

- **API details:** **New API**
- **API name:** `PollyNotesAPI`

6. Leave all other values as the default and choose **Create API**.

7. Choose **Create Resource** and set the following values:

- Untoggle **Proxy resource:** option, if it is not already disabled.
- **Resource Name:** `notes`
- **CORS (Cross Origin Resource Sharing):** (Select the checkbox.)

8. Choose **Create Resource**.

Congratulations! You now have an API and resource that you can build your connections on.

Task 2: Configure the GET method and add a mapping template to transform responses

The first Lambda function that you need to allow access to is the list-function. Whenever a request is sent to the API Gateway resource, using a GET method, this function should be invoked and the response returned to the requester. If a UserId is specified in the event, the Lambda function will only return items that match that UserId. Using the integration request, you will create a mapping template to hard code that parameter for now.

After the request is configured to return items for a single user, it is redundant to include the UserId in every item that is returned. Only returning responses with the information needed, and no more, is important to increase performance and reduce data transfer costs as your application scales.

High-Level Instructions

- Configure a GET method for the notes resource, map it to the **list-function**, and test the output.
- Add a mapping template to manually specify `{"UserId":"student"}` and confirm the output.
- Create a response mapping template to transform the response to only return the note id and note text.

Detailed Instructions

TASK 2.1: CONFIGURE GET METHOD

There are several methods that can be configured for your new resource, the GET method is the default method that a web browser requests when you browse to a URL. The GET method is used to retrieve data.

9. Select the **/notes** resource, choose the **Create Method**.

10. In the method details page, make the following selections:

- For **Method type**, choose **GET**.
- For **Lambda Function**, select the ARN that contains `list-function`.

11. Leave all other as the default and choose **Create Method**.

12. In the **/notes - GET - Method Execution** panel, choose **Test** tab.

- Choose **Test**.

Note: You may have to scroll to see the **Test** button.

The Lambda function is successful and returns a **Status** of **200** . The **Response Body** now returns a JSON array with all items in the DynamoDB table. You can examine the entire **Log** for more information on how the request was processed. As the dataset grows larger in production, this would quickly become a performance issue. If the notes are private, it could also be a security issue if every user can download the notes of all users.

TASK 2.2: UPDATE THE GET METHOD INTEGRATION REQUEST TO PASS A USERID VARIABLE

The integration request is the internal interface of a REST API method in API Gateway. In the integration request, you map the body of a route request, or the parameters and body of a method request, to the formats required by the backend.

Additional information: The mapping templates are a script in Velocity Template Language (VTL) that transforms a request or response body to the needed data format. Mapping templates can be specified in the integration request or in the integration response. They can reference data made available at runtime as context and stage variables.

The mapping can be as simple as an identity transform that passes the headers or body through the integration as-is from the client to the backend for a request. The same is true for a response, in which the payload is passed from the backend to the client.

13. At the top of the page, choose **Integration Request** .

14. Choose **Edit**

- Scroll down to **Mapping Templates** section.
- Choose **Add mapping template** .
- In the **Content type** box, enter `application/json` .

For now, hard code the UserId parameter, in the next lab you will use the current username.

- In the **Template body** box, enter `{"UserId":"student"}`
- Choose **Save** .

15. In the **/notes - GET - Method Execution** panel, choose **Test** tab.

- Choose **Test** .

Once again the Lambda function is successful and the **Status** it returns is **200** . The **Response Body** now returns a JSON array with only notes where the **UserId** is *student*. You can examine the entire Execution log for more information on how the request was processed.

Note: The list-function performs a DynamoDB query operation if a UserId was passed in the event, otherwise it performs a scan.

TASK 2.3: LIMIT THE RESPONSE DATA RETURNED FROM THE API METHOD

Now that you are specifying the UserId in the request, you can reduce the amount of redundant data returned in the response. You will specify a mapping template for the integration response.

The integration response is the internal interface of a REST API method in API Gateway. You use the integration response to map the status codes, headers, and payload, that are received from the backend, to the response format that is returned to a client app.

16. At the top of the page, choose **Integration Response** .

17. Choose **Edit**

- Scroll down to **Mapping Templates** section.
- In the **Content type** box, enter `application/json` .

18. **Copy/Paste:** In the **Template body** box, choose the correct choice and insert it.

- Choice A:

```
#set($inputRoot = $input.path('$'))
[
    {
        "NoteId" : "$elem.NoteId",
        "Note" : "$elem.Note"
    }
]
```

- Choice B:

```
#set($inputRoot = $input.path('$'))
[
    #foreach($elem in $inputRoot)
    {
    "NoteId" : "$elem.NoteId",
    "Note" : "$elem.Note"
    }
    #if($foreach.hasNext),#end
    #end
]
```

Answer: You can find the solution to this step in the TODO 1 solution section at the bottom of these instructions.

- Choose **Save** .

Now that the integration response has been configured, you can confirm the output by testing the method.

19. At the top of the page, in the **/notes - GET - Method Execution** panel, choose **Test** tab.

- Choose **Test** .

The response is now a JSON array with just the **NoteId** and **Note** properties in each object. Since the UserId is not needed in this response you are able to reduce the amount of data returned. As more notes are entered into the system this will exponentially reduce the size of the responses. Smaller responses will lead to better application performance and reduced data transfer costs.

Task 3: Configure the POST method and add a request validation model

You now have a way to retrieve data from the database. You need a way to add or update data in the database as well. The next Lambda function to provide access to is the **createUpdate-function** . To access this function, you will use the **POST** method. When you post a new note to the notes resource, you will include the data to be added (or updated), in the body of your request.

By default, the entire body is passed to the backend as is. One of the features of the API Gateway is that it can verify the data structure using a Model.

A Model is a data schema specifying the data structure of a request or response payload. A Model is required for generating a strongly typed SDK of an API. It is also used to validate payloads.

By verifying the data structure before it is passed to the backend, you can make sure that the input is formatted correctly before invoking your Lambda function. If the input does not match the Model schema, API Gateway will immediately respond with an error message and not invoke your Lambda function.

High-Level Instructions

- Configure the POST method, map it to the **createUpdate-function** , and test the default behavior with a new note object.
- Create a Model that specifies the note schema above. (The Schema can be found in task 3.2 if you need a hint.)
- Configure validation on the request body, enforce the schema, and test a valid and invalid submission.

Detailed Instructions

TASK 3.1: CONFIGURE THE POST METHOD AND ADD A REQUEST VALIDATION MODEL

In this task, you configure the **POST** method to pass requests to the **createUpdate-function**. You will then test the method to make sure it works as expected.

20. Select the **/notes** resource, choose the **Create Method**.

21. In the method details page, make the following selections:

- For **Method type**, choose **POST**.
- For **Lambda Function**, select the ARN that contains `createUpdate-function`.

22. Leave all other as the default and choose **Create Method**.

This method relies on information passed in the body of the request, so you can test it as is.

23. In the **/notes - POST - Method Execution** panel, choose **Test** tab.

- **Copy/Paste:** In the **Request Body** code block, paste the following JSON object:

```
{
    "Note": "This is your new note added using the POST method",
    "NoteId": 3,
    "UserId": "student"
}
```

- Choose **Test**.

The response body just returns the **NoteId**. You can also examine the entire Execution log for more information on what happened. Using the **GET** method, confirm that the new note was added.

24. In the **Resources** panel, for the **/notes** resource, choose **GET**.

25. In the **/notes - GET - Method Execution** panel, choose **Test** tab.

- Choose **Test**.

The **Response Body** now has the note you added and the placeholder notes that were already in the table.

TASK 3.2: ENFORCE A SCHEMA FOR THE POST REQUEST BODY

The **POST** function works as is, but you can make it better. For instance, if the note isn't submitted or all required fields are not included, there is no need to pass the request to the Lambda function. Using an API Gateway Model, validate that there is information in the body of the request and choose what fields are passed to the function in the body.

26. In the left navigation panel, under **API: PollyNotesAPI**, choose **Models**.

27. Choose **Create model**.

- For **Name**, enter `NoteModel`
- For **Content type**, enter `application/json`

28. **Copy/Paste:** For **Model schema**, choose the correct choice and insert it.

- Choice A:

```
{
    "title": "Note",
    "type": "array",
    "properties": {
        "UserId": {"type": "string"},
        "NoteId": {"type": "integer"},
        "Note": {"type": "string"}
    },
    "required": ["UserId", "NoteId", "Note"]
}
```

- Choice B:

```
{
    "title": "Note",
    "type": "object",
    "properties": {
        "UserId": {"type": "string"},
        "NoteId": {"type": "integer"},
        "Note": {"type": "string"}
    },
    "required": ["UserId", "NoteId", "Note"]
}
```

Answer: You can find the solution to this step in the TODO 2 solution section at the bottom of these instructions.

29. Choose **Create**.

Note: If the **Create model** button is greyed out, add the Model schema code first, then add the content type.

30. In the left navigation panel, under **API: PollyNotesAPI**, choose **Resources**.

31. In the **Resources** panel, for the **/notes** resource, choose **POST**.

32. In the **/notes - POST - Method Execution** panel, choose **Method Request**.

- In **Method request settings**, choose **Edit**.
- For **Request validator**, choose **Validate body**.
- Expand **Request Body**
- Choose **Add model**
- For **Content type**, enter `application/json`
- For **Model**, select **NoteModel**.
- Choose **Save**.

TASK 3.3: TEST THE REQUEST VALIDATOR

You can now test the validator. First, send a note in the payload without the correct schema, and then one that uses the correct schema.

33. In the **/notes - POST - Method Execution** panel, choose **Test** tab.

- **Copy/Paste:** In the **Request Body** code block, paste the following json object.

```
{
    "Note": "This is your updated note using the Model validation",
    "UserId": "student",
    "id": 3
}
```

- Choose **Test** .

Notice that the test was not successful, there is a **Status** of **400** and the **Response Body** clearly states that there was an **Invalid request body** . If you review the **Logs** , there is more detail. *"Request body does not match model schema for content type application/json: [object has missing required properties ([\"NoteId\"])]"*

- **Copy/Paste:** In the **Request Body** code block, replace what is there with the following json object.

```
{
    "Note": "This is your updated note using the Model validation",
    "UserId": "student",
    "NoteId": 3
}
```

- Choose **Test** .

With a valid body that matches the enforced schema, the request succeeds and is processed by the Lambda function.

Task 4: Deploy the API with CORS configurations

Cross-Origin Resource Sharing (CORS) is an HTTP-header based mechanism that allows a server to indicate any other origins (domain, scheme, or port) than its own from which a browser should permit loading of resources. CORS also relies on a mechanism by which browsers make a "preflight" request, to the server hosting the cross-origin resource, in order to check that the server will permit the actual request. In that preflight, the browser sends headers that indicate the HTTP method and headers that will be used in the actual request.

An example of a cross-origin request: the front-end JavaScript code served from https://domain-a.com uses XMLHttpRequest to make a request for https://domain-b.com/data.json.

In later labs, your web application will be accessed with an Amazon S3 domain name, and the API will have an Amazon API Gateway domain name, so this configuration is required for the labs. There are many reasons to host backend APIs with different domains, configuring CORS is a common requirement in API development.

After configuring CORS, you will create a new stage and deploy the API. This creates an endpoint that can be accessed from the internet.

High-Level Instructions

- Enable CORS on all methods to allow any origin.
- Deploy the API to a Prod stage.
- Test the API with your web browser.

Detailed Instructions

TASK 4.1: CONFIGURE CROSS-ORIGIN REQUEST SHARING (CORS)

When you first created the API, you instructed the service to enable CORS. That created and configured the **OPTIONS** method for the resource. The **OPTIONS** method is requested when browsers perform the "preflight" request. You have created methods, so you need to configure CORS for those methods.

In this task, you add **Response Headers** to the methods you have configured. You can configure this manually but the instructions below use the API Gateway console to automate the configuration.

34. In the **Resources** panel, choose **/notes** .
35. Choose the **Enable CORS** .

- For **Gateway Responses** , check the box for **Default 4XX** and **Default 5XX** . This is to allow you to view 4xx and 5xx errors from your browser in the event you need to troubleshoot.
- For **Access-Control-Allow-Methods** , check the box for **GET** , **OPTIONS** and **POST** .

36. Choose **Save** .

For each existing method, this step added **Header Mappings** and **Response Headers** for each status code your methods have configured. Now your web browser will allow CORS for these API methods. You can modify this configuration to make it more secure in the future when your web front-end and API use known domain names.

TASK 4.2: DEPLOY THE API

Your API has been configured, but before you can access this configuration with the API endpoint, you need to create a stage and deploy the configuration to that stage.

A stage is a logical reference to a lifecycle state of your API. API stages are identified by API ID and stage name. You could create a "dev" and a "prod" stage, and all changes could be deployed to dev and fully tested before you allow them to be deployed to prod.

Another common strategy is to have a new stage for each major or minor version of your API. Anytime you make a change that could impact legacy versions, of the front end applications that use your API, you could create a new stage (for example, "v2"). Your users would have to change the API endpoint URL to take advantage of the new features after they have confirmed that their code works with the new version.

An API deployment is a point-in-time snapshot of your API Gateway API. To be available for clients to use, the deployment must be associated with one or more API stages.

37. In the **Resources** panel, choose **/notes** .
38. At the upper-right corner of the page, choose the **Deploy API** .

- For **Stage** , choose ***New Stage*** .
- For **Stage name** , enter `Prod`

39. Choose **Deploy** .
40. Copy the **Invoke URL** and paste it to a new web browser tab, append `/notes` to the end of the URL and press Enter.

Expected Output:

The expected output should be a JSON array like the one below. Depending on your web browser, the output may be formatted in a way that is easier to read.

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

[

{

"NoteId" : "1",

"Note" : "DynamoDB is NoSQL"

},

{

"NoteId" : "2",

"Note" : "A DynamoDB table is schemaless"

},

{

"NoteId" : "3",

"Note" : "This is your updated note using the Model validation"
```

```
"Note" : "This is your updated Note using the Model Validation"
}
]
```

Summary

Congratulations! You can access your AWS Lambda functions with the Amazon API Gateway. As you can see, your API is open to the world at this point. In the next lab, you will secure it before adding any more API methods or resources.

You can now:

- Create RESTful API resources and configure CORS for your application.
- Integrate API methods with AWS Lambda functions to process application data.
- Configure mapping templates to transform the pass-through data during method integration.
- Create a request model for API methods to ensure that the pass-through data format complies with application rules.
- Deploy the API to a stage and validate the results using the API endpoint.

End lab

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

41. Return to the **AWS Management Console** .

42. At the upper-right corner of the page, choose **AWSLabsUser** , and then choose **Sign out** .

43. Choose End lab and then confirm that you want to end your lab. 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

- Amazon API Gateway Developer Guide
- Velocity Template Language (VTL)
- Cross-Origin Resource Sharing (CORS)

Code Challenge Solutions

TODO 1 SOLUTION

- Choice A is incorrect because it does not parse each object.
- **Choice B is correct code snippet.** In this case, your response received is list of objects and you need to parse each object.

```
#set($inputRoot = $input.path('$'))
[
    #foreach($elem in $inputRoot)
    {
        "NoteId" : "$elem.NoteId",
        "Note" : "$elem.Note"
    }
    #if($foreach.hasNext),#end
    #end
]
```

Return to the instructions

TODO 2 SOLUTION

- Choice A is incorrect because the type property is set to an **array** .
- **Choice B is correct code snippet.** In this scenario, you are receiving the request as an object, not an array.

```
{
    "title": "Note",
    "type": "object",
    "properties": {
        "UserId": {"type": "string"},
        "NoteId": {"type": "integer"},
        "Note": {"type": "string"}
    },
    "required": ["UserId", "NoteId", "Note"]
}
```

Return to the instructions

Appendix

AWS SERVICES NOT USED IN THIS LAB

AWS services that are not used in this lab are deactivated in the lab environment. In addition, the capabilities of the services used in this lab are limited to what the lab requires. Expect errors when accessing other services or performing actions beyond those provided in this lab guide.

ICON KEY

Various icons are used throughout this lab to call attention to different types of instructions and notes. While not all of the icons will be used, 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 note, tip, or important guidance.

- **Additional information:** Where to find more information.
- **Caution:** Information of special interest or importance (not so important to cause problems with the equipment or data if you miss it, but it could result in the need to repeat certain steps).
- **WARNING:** An action that is irreversible and could potentially impact the failure of a command or process (including warnings about configurations that cannot be changed after they are made).
- **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.
- **Copy/Paste:** A code block that displays the contents of a script or file you need to copy and paste that has been pre-created for you. When you need to copy only a certain part of a code block, there will be numbered **TODO** comments in the code.
- **Knowledge check:** An opportunity to check your knowledge and test what you have learned.
- **Security:** An opportunity to incorporate security best practices.
- **Refresh:** A time when you might need to refresh a web browser page or list to show new information.
- **Copy command:** A time when copying a command, script, or other text to a text editor (to edit specific variables within it) might be easier than editing directly in the command line or terminal.
- **Hint:** A hint to a question or challenge.
- **Answer:** An answer to a question or challenge.
- **Group effort:** A time when you must work together with another student to complete a task.

Return to the instructions