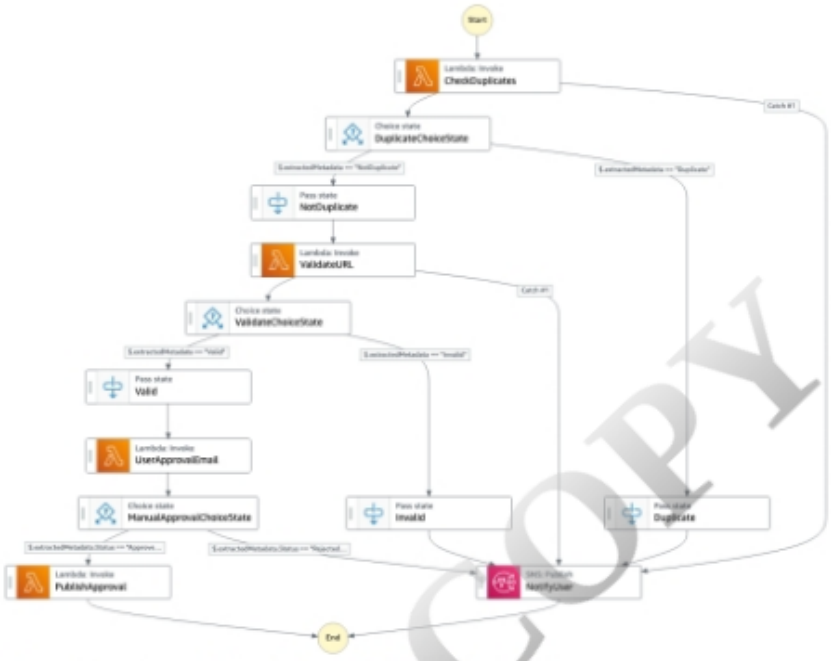


75. Choose **Cancel** to return to the **Edit PublishStateMachine** page.

76. Choose **Cancel** to return to the **PublishStateMachine** page.

The state machine setup is now complete. Now, create a new rule in EventBridge to invoke the new state machine.

77. In the AWS Management Console, on the Services menu, choose **Amazon EventBridge**.

78. In the left navigation pane, choose **Rules**.

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

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

80. Choose Next

81. 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:

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

82. Choose Next

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

- **Target types:** Select **AWS service**
- **Select a target:** Select **Step Functions state machine** from the dropdown menu
- **State machine:** Choose **PublishStateMachine**
- **Execution role:** Choose **Use existing role**
- Choose **EventBridgeStateMachineRole** from the dropdown menu

84. Choose Next

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

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

Task 5: Reviewing the deployment and running test cases using the application

In this task, you take a quick overview of all of the components that have been deployed so far. You then run a few test cases to test the Step Functions workflow and the EventBridge rule. Note the following:

- The backend and front end have been integrated for end-to-end testing.
- You created a Lambda function and a state machine manually.
- You reviewed the Lambda functions that the AWS SAM deploy process created for the Step Functions state machine workflow.
- You updated the state machine with the Lambda function ARNs, API Gateway endpoint, and SNS topic ARN.

87. Open and log in to the bookmark application, if you are not already logged in.

Test case 1: Add a new bookmark without the shared flag

In this test case, add a new *unshared* bookmark. Check the Step Functions state machine to see if it was invoked or not. Then, share the new bookmark and approve it.

88. In the bookmark application, add a new bookmark with the **Share Bookmark** toggle set to **Off**. Make sure you enter a valid bookmark URL.

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

90. Choose the name of the **PublishStateMachine** state machine.

No jobs are listed in the **Running** state because the unshared bookmark did not invoke the **PublishStateMachine** workflow.

Test case 2: Add a new bookmark with the shared flag

In this test case, add a new *shared* bookmark. Then, approve the new bookmark.

91. In the bookmark application, add a new bookmark with the **Share Bookmark** toggle set to **On**. Make sure you enter a valid bookmark URL.

This invokes the **PublishStateMachine** workflow.

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

93. Choose the name of the **PublishStateMachine** state machine.

94. Choose the name of the **Running** state.

Scroll down to the **Graph inspector** section, and review the state machine steps.

Because this is a new bookmark and no duplicates are found, the **duplicateBookmarkCheck** function sends a **NotDuplicate** message and invokes the **validateURL** function.

Because this is a valid URL, the **validateURL** function sends a **Valid** message and invokes the **userApprovalEmail** function.

95. Check your email for approve and reject links from the **userApprovalEmail** function.

96. Choose the approval URL, and check the Step Functions workflow.

Because you manually approved the URL, the **publishApproval** function is invoked, and the contest value in the DynamoDB table is updated to **Approved**.

Test case 3: Add a bookmark with an invalid URL

In this test case, add a new *shared* bookmark that is *invalid*. Then, share the new bookmark.

97. In the bookmark application, add a new bookmark URL **without** any of the generic top level domains (".edu" or ".gov" or ".com" or ".info" or ".mil" or ".net" or ".org") such as `aws.training` in the URL field. Make sure the **Share Bookmark** toggle is set to **On**.

This invokes the **PublishStateMachine** workflow.

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

99. Choose the name of the **PublishStateMachine** state machine.

100. Choose the name of the most recently started state.

Scroll down to the **Graph inspector** section, and review the state machine steps.

Because this is a new bookmark and no duplicates are found, the **duplicateBookmarkCheck** function sends a **NotDuplicate** message and invokes the **validateURL** function.

Because this is *not* a valid URL, the **validateURL** function sends an **Invalid** message and invokes the **notifyUser** function.

Test case 4: Add a duplicate bookmark as a different user

In this test case, add a *new* user account in the bookmark application. Then, add a *shared* bookmark that is a duplicate of a bookmark (URL) that another user added.

101. In the bookmark application, create a second user account.

102. Add a bookmark that is a duplicate of a bookmark that another user added. Make sure the **Share Bookmark** toggle is set to **On**.

Note Only the URL field value is checked for duplicate.

This invokes the **PublishStateMachine** workflow.

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

104. Choose the name of the **PublishStateMachine** state machine.

105. Choose the name of the most recently started state.

Scroll down to the **Graph inspector** section, and review the state machine steps.

Because this is a duplicate bookmark that was already shared, the **duplicateBookmarkCheck** function sends a **Duplicate** message and invokes the **notifyUser** function to send an email saying the bookmark is a duplicate.

Note If you have lab time remaining, run a few test cases for the human rejection process.

Conclusion

Congratulations! You now have successfully:

- Configured EventBridge to target a Step Functions workflow
- Used a Step Functions Standard workflow to orchestrate tasks
- Used Lambda for tasks within a Step Functions state machine

End lab

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

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

107. At the upper-right corner of the page, choose **awsstudent@<AccountNumber>**, and then choose **Sign out**.

108. Choose End Lab.

109. Choose Submit.

110. (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 asynchronous messaging for microservices, see https://aws.amazon.com/blogs/compute/understanding-asynchronous-messaging-for-microservices/.
- For more information about Step Functions, see https://docs.aws.amazon.com/step-functions/latest/dg/welcome.html.

aws training and certification

Lab 4: Observability and Monitoring

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 now have a stable bookmarking application with the features you need to roll it out to the larger team. But, before it becomes your production version, you need to think about the operational data that you will need to know if it's working as expected and to understand how users are accessing it. You need to build observability into your application.

Before you go any further, you need to take a closer look at your code and the potential points of failure. Make sure your error handling is appropriate before you start getting higher levels of production traffic when this becomes the permanent way for sharing knowledge base links. After analyzing existing logs, you realize that you need to pay attention to how you are coding your logs, and you need more insights for better observability.

This lab focuses on observability and monitoring techniques to protect your systems from unforeseen events that have a negative impact. Observability in production is a requirement when you design complex systems.

This lab has prebuilt code that is meant to send errors and occasionally fail, allowing you to set up proper observability and monitoring services to handle these occasions. You will add AWS X-Ray tracing and work with Amazon CloudWatch to monitor and troubleshoot your application.

Three pillars of observability

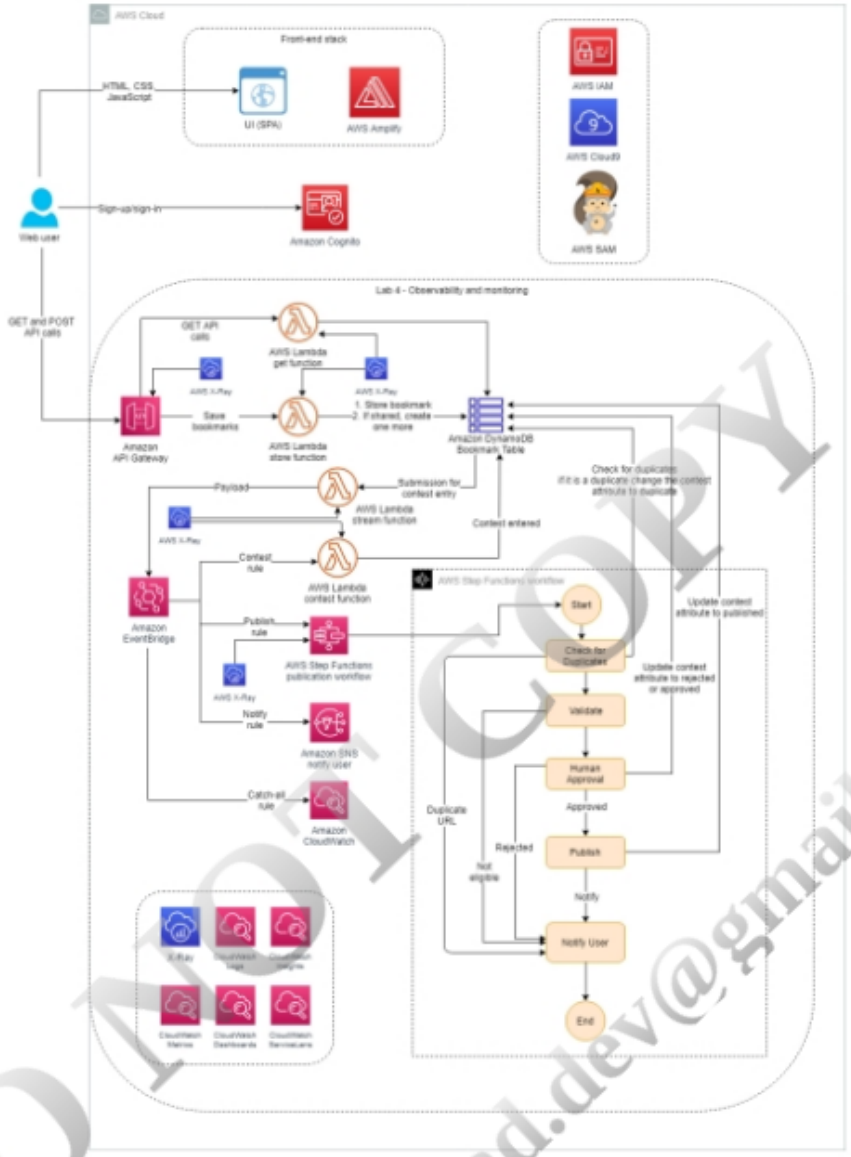
Logs, metrics, and distributed tracing are often known as the three pillars of observability. These are powerful tools that, if well understood, can unlock the ability to build better systems.

Logs provide valuable insights into how you measure your application health. Event logs are especially helpful in uncovering growing and unpredictable behaviors that components of a distributed system exhibit. Logs come in three forms: plaintext, structured, and binary.

Metrics are a numeric representation of data measured over intervals of time about the performance of your systems. You can configure and receive automatic alerts when certain metrics are met.

Tracing can provide visibility into both the path that a request traverses and the structure of a request. An event-driven or microservices architecture consists of many different distributed parts that must be monitored. Imagine a complex system consisting of multiple microservices, and an error occurs in one of the services in the call chain. Even if every microservice is logging properly and logs are consolidated in a central system, it can be difficult to find all relevant log messages.

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



Objectives

After completing this lab, you will be able to:

- Update your code for better logs, applying logging best practices
- Use Amazon CloudWatch Logs and CloudWatch metrics to monitor application operations
- Use X-Ray and CloudWatch ServiceLens to troubleshoot application issues

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 **90 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.

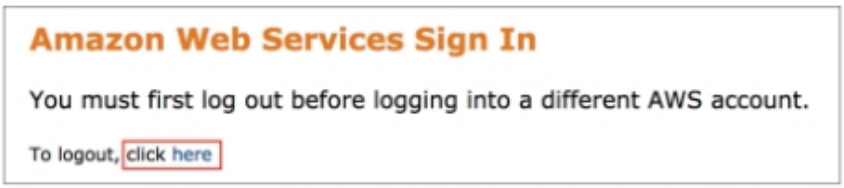
3. On the **Sign in as IAM user** page:

- 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: Looking at key services and setting up the app

In this task, you look at the different AWS services that you use in this lab. Take a few minutes to navigate the console for each service.

- **AWS X-Ray** helps developers analyze and debug production, distributed applications, such as those built using a microservices architecture. With X-Ray, you can understand how your application and its underlying services are performing to identify and troubleshoot the root cause of performance issues and errors. X-Ray provides an end-to-end view of requests as they travel through your application and shows a map of your application's underlying components. You can use X-Ray to analyze both applications in development and in production, from simple three-tier applications to complex microservices applications consisting of thousands of services.
- **Amazon CloudWatch Logs Insights** is a fully managed service that is designed to work at cloud scale with no setup or maintenance required. The service analyzes massive logs in seconds and gives you fast, interactive queries and visualizations. CloudWatch Logs Insights can handle any log format and autodiscovers fields from JSON logs.
- **Amazon CloudWatch ServiceLens** is a feature that enables you to visualize and analyze the health, performance, and availability of your applications in a single place. CloudWatch ServiceLens ties together CloudWatch metrics and logs, as well as traces from X-Ray, to give you a complete view of your applications and their dependencies. This enables you to quickly pinpoint performance bottlenecks, isolate root causes of application issues, and determine impacted users.

First, you launch the application by running a preprogrammed script in AWS Cloud9.

4. Choose Services and select **Cloud9**.
5. On the left side of the page, choose the menu ☰ icon to expand the menu, and choose **Your environments**.

If the menu is already expanded, move to the next step.

6. 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.

7. From the terminal, run the following commands to download the startup script, 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-4-ObservabilityMonitoring/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.

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.
- Then, the script runs **npm build**, deploys the bookmark application, and uploads the **app.zip** file to the **samserverless** Amazon Simple Storage Service (Amazon S3) bucket.

Note Keep this AWS Cloud9 browser tab open because you return to it later in the lab.

8. In the AWS Management Console, choose Services and open **API Gateway** in a new tab.
9. Choose the **Bookmark App**.
10. In the left navigation pane, choose **Stages**.
11. In the **Stages** pane, choose the ▶ **dev** stage.

You see the API Gateway end point has been deployed successfully.

Note Keep this Amazon API Gateway page open.

12. In the AWS Cloud9 IDE, choose the **>** arrow next to the **app-code** folder to expand it.
13. Choose the **>** arrow next to the **test** folder to expand it.
14. Open the **simple-post.yaml** file.
15. In the AWS Cloud9 terminal, run the following AWS CLI and bash commands to replace *API_GATEWAY_URL* with the actual value in the **simple-post.yaml** file.

```
cd test
export API_GATEWAY_ID=$(aws apigateway get-rest-apis --query 'items[?name==`Bookmark
App`].id' --output text) >> ~/environment/app-code/labVariables
export AWS_REGION=$(curl -s 169.254.169.254/latest/dynamic/instance-identity/document
| jq -r '.region')
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" simple-post.yaml
cd ..
```

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

Note Keep the AWS Cloud9 page open.

Task 2: Enabling X-Ray and CloudWatch

In this task, you enable X-Ray so that it collects data to analyze for errors. The bookmark application makes heavy use of API Gateway and AWS Lambda. To get full trace information, X-Ray must be enabled on both.

17. Copy the **APIGatewayCLRole** value from the left side of the lab instructions.
18. Return to the API Gateway browser and in the API Gateway console, at the bottom of the left navigation pane, choose **Settings**.
19. In the **CloudWatch log role ARN** box, paste the **APIGatewayCLRole** value that you copied from the lab page.
20. Choose Save
21. In the left navigation pane, choose **Stages**.
22. Within the ▶ **dev** stage editor, choose the **Logs/Tracing** tab.
23. Under **CloudWatch Settings**, configure the following options:

- Select **Enable CloudWatch Logs**. Make sure that the **Log level** is set to **ERROR**.
- Select **Enable Detailed CloudWatch Metrics**.

24. Under **X-Ray Tracing**, select **Enable X-Ray Tracing**.
25. Choose Save Changes
26. In the AWS Management Console, choose Services and select **Lambda**.
27. Under **Functions**, enter `createBookmark` in the filter box, and press ENTER.
28. Choose the function with **createBookmark** in the **Function name**.
29. Choose the **Configuration** tab and select **Monitoring and operations tools** in the left navigation pane.
30. In the **Monitoring and operations tools** section, choose Edit
31. Under **AWS X-Ray**, select **Active tracing**.
32. Choose Save

Now any request that the API gets from the four deployed API Gateway methods injects the tracing header into the request. This then flows to the Lambda function and allows you to see where a request is being dropped or receiving an error.

ⓘ **Note** Because this application uses several Lambda functions, X-Ray has been pre-enabled on the remaining application functions for you.

Note Keep this browser tab open because you return to it later in the lab.

Task 3: Running the load test using Artillery

You start this task by using Artillery, which is a load-testing and functionality tool, along with Faker, a simple random data generator. You run the **simple-post.yaml** file from the **test** folder in AWS Cloud9, which starts adding random bookmarks and invoking the **createBookmark** function.

33. Return to the AWS Cloud9 browser tab.

34. Run the following code to install Artillery, install Faker, and run the script:

```
cd test
npm install artillery -g
npm install faker@5.5.3
artillery run simple-post.yaml
```

💬 **Note** This script runs for 2 minutes, adding data through the API and then invoking the **createBookmark** Lambda function.

Task 4: Coding for better logging

In this task, you review the results from running the Artillery load test while also learning how to optimize CloudWatch Logs Insights for improved logging by using various queries.

The **createBookmark** Lambda function code has been designed to produce errors around 10 percent of the time when adding bookmarks. This is normal for this lab scenario.

35. Return to the browser tab with the **createBookmark** Lambda function.

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

37. Review the following three lines of code:

```
var AWSXRay = require('aws-xray-sdk');
var AWS = AWSXRay.captureAWS(require('aws-sdk'));
var dynamodb = AWSXRay.captureAWSClient(new AWS.DynamoDB());
```

ⓘ **Note** You can instrument all AWS SDK clients by wrapping your **aws-sdk require** statement in a call to **AWSXRay.captureAWS**.

To instrument individual clients, wrap your AWS SDK client in a call to **AWSXRay.captureAWSClient**. For example, to instrument an Amazon DynamoDB client, you would change the following code:

```
var dynamodb = new AWS.Dynamodb();
```

to

```
var dynamodb = AWSXRay.captureAWSClient(new AWS.DynamoDB());
```

38. Choose the **Monitor** tab.

In the **Metrics** tab you can view all of the CloudWatch metrics pertaining to this specific Lambda function.

39. Refresh until you see some metrics begin to populate.

40. Pause over each line for **Invocations**, **Duration**, and **Error count and success rate (%)**.

41. Choose the **Logs** tab to view the **CloudWatch Logs Insights** section. Choose refresh.

Here you can gain a quick glimpse and easily filter for **Recent invocations** and **Most expensive invocations in GB-seconds**.

42. Under **Recent invocations**, choose the ▶ icon for the most recent invocation to expand the details.

Here you can see some key log attributes that come by default through CloudWatch Logs Insights, such as **duration**, **maxMemoryUsed**, **memorySize**, **requestId**, and **xrayTraceId**.

43. Under **Most expensive invocations**, choose the ▶ icon for the most recent invocation to expand the details.

44. Choose the **Metrics** tab and refresh again to load and observe the rest of the data.

45. Choose View logs in CloudWatch

46. In the left navigation pane, under **Logs**, choose **Logs Insights**.

Near the top of the page is the query editor. When you first open CloudWatch Logs Insights, this box contains a default query that returns the 20 most recent log events.

47. From the **Select log group(s)** dropdown list, select **aws/lambda/createBookmark**.

When you select a log group, CloudWatch Logs Insights automatically detects fields in the data in the log group.

💬 **Note** To see these discovered fields, select **Fields** on the right side of the page.

48. In the query editor at the top of the page, leave the default query, which shows the 20 most recent log entries. Choose Run query

49. Find and select the **ERROR Invoke Error** log event to view the details.

Note If the **ERROR Invoke Error** log event doesn't appear in the 20 most recent log entries, run the query with a value of **50**. The query would look like the following:

```
fields @timestamp, @message
| sort @timestamp desc
```

```
| limit 50
```

50. At the top of the page, delete the code in the query editor. Copy and paste the following code into the query editor:

```
filter @type = "REPORT"
| fields @requestId, @billedDuration
| sort by @billedDuration desc
```

51. Choose Run query

ⓘ **Note** Every Lambda function invocation remits a log entry as the last line with the keyword REPORT. The REPORT line contains all of the standard fields and is present on every invoke.

This query tells CloudWatch Logs Insights to filter for the REPORT line only and then return two fields: **requestID** and **billedDuration**. Then, the results are sorted by **billedDuration**.

52. Choose the ▶ **1** icon to expand and observe the details.

53. Compare the **@billedDuration** to the **@duration**, and then compare **@maxMemoryUsed** to **@memorySize**.

You can see that **@maxMemoryUsed** is lower than the **@memorySize** provisioned for the Lambda function. This is a great starting point for learning more about how your function operates so you can better adjust it.

54. At the top of the page, delete the code in the query editor. Copy and paste the following code into the query editor:

```
stats count(*) by @logStream
    | limit 100
```

55. Choose Run query

56. Choose the **Visualization** tab.

57. From the Line▼ dropdown list, select **Bar**.

The results show the number of log events in the log group for each log stream.

ⓘ **Note** When you run a query that uses the **stats** function to group the returned results by the values of one or more fields in the log entries, you can view the results as a bar chart, pie chart, line graph, or stacked area graph. These options help you more efficiently visualize trends in your logs.

Learn more For more sample queries for CloudWatch Logs, see Sample Queries.

Note Keep this browser tab open because you return to it later in the lab.

Task 5: Tracing and troubleshooting

Earlier in the lab, you enabled X-Ray tracing for API Gateway and the **createBookmark** Lambda function. Now, use X-Ray tracing to help diagnose why you are getting errors and fails within the **createBookmark** function.

58. In the AWS Management Console, choose Services and open **X-Ray** in a new tab.

Note If you don't see a left navigation pane when you open X-Ray, choose Get started to open the X-Ray console, and then choose Cancel

59. In the left navigation pane, choose **Traces**.

60. From the URL▼ dropdown list, select **StatusCode**.

61. From the Last 5 minutes ▼ dropdown list, select **1 hour**.

Here you can see the percent of traces with 200 ok responses and 502 response errors.

62. From the **Trace list**, choose one trace ID with a **200** response to open in a new browser tab.

63. From the **Trace list**, choose one trace ID with a **502** response to open in a new browser tab.

64. Choose the browser tab with the **502** response to view the timeline for this trace.

The **Timeline** view shows a hierarchy of segments and subsegments. The first entry in the list is the segment, which represents all of the data that the service recorded for a single request, as shown in the following image.

Below the segment are subsegments, which should look similar to the following image.

The X-Ray SDK automatically records subsegments when you use an instrumented AWS SDK, HTTP, or SQL client to make calls to external resources. You can also tell the SDK to record custom subsegments for any function or block of code.

65. Choose each icon within the trace map to see the end-to-end path of this request.

66. Below the timeline, choose the ! **Status** icon to view the segment details.

The **Overview** tab shows information about the request and response. Notice that **Error** is **false**, but **Fault** is **true**.

67. Choose the **Resources** tab.

The **Resources** tab for a segment shows information about the AWS resources running your application and the X-Ray SDK.

68. Choose Close

69. Choose the **Raw data** tab.

Here you can access the raw trace data that the console uses to generate the timeline. This JSON document contains all of the segments and subsegments that make up the trace.

70. Take a moment to review the trace data.

71. Choose the browser tab where you opened the **200** response trace ID to view its timeline.

72. Compare this successful request with the **502** request you just viewed.

You can see within the trace map that the end-to-end path was successful from the client to API Gateway and then to the **createBookmark** Lambda function, and finally populated the DynamoDB **bookmarksTable**.

73. Select each segment and subsegment to view their details and compare what you see for this successful request.

Note Keep the X-Ray console browser tab open because you return to it later in the lab.

Task 6: Adding subsegment code

74. Return to the browser tab with the **createBookmark** Lambda function.

75. In the **Code source** section, the **index.js** file is already open.

76. Find the line that says *//Call to the sub segment annotation code goes here.*

77. Copy the following code and paste it over the line in the previous step to replace that line.

```
var segment = AWSXRay.getSegment();
await addSegment(segment,message);
```

78. Find the line that says *//Placeholder for sub segment annotation code.*

79. Copy the following code and paste it over the line in the previous step to replace that line.

```
const addSegment = (segment,message) => {

    let bookmark = JSON.parse(message.body);
    const f = async (subsegment) => {

      subsegment.addAnnotation('uid',bookmark.id);
      subsegment.addAnnotation('name',bookmark.name);
      subsegment.addMetadata('bookmarkUrl', bookmark.bookmarkUrl);
      subsegment.addMetadata('username', bookmark.username);

      subsegment.close();
    };
    AWSXRay.captureAsyncFunc("adding annotations and metadata", f, segment);
};
```

After adding the code snippets, the **index.js** file should look like the following:


```
var AWSXRay = require('aws-xray-sdk');
var AWS = AWSXRay.captureAWS(require('aws-sdk'));
var dynamodb = AWSXRay.captureAWSClient(new AWS.DynamoDB());

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

  //Fail the messages randomly to see those errors in X-Ray tracing. It's for testing
only.
  if(Math.random() < 0.3)
    throw new Error('An unknown error occurred');

  //Can you throw a different response code other than 200?

  //Timeout failures about 10%
  if(Math.random() < 0.2) {
     await new Promise(resolve => setTimeout(resolve, 10000));
  };

  if (message.body) {
    let bookmark = JSON.parse(message.body);
    let params = {
      TableName: process.env.TABLE_NAME,
      Item: {
        id: { S: bookmark.id },
        bookmarkUrl: { S: bookmark.bookmarkUrl },
        name: { S: bookmark.name },
        description: { S: bookmark.description },
        username: { S: bookmark.username },
        shared: { BOOL: bookmark.shared }
      }
    };

    var segment = AWSXRay.getSegment();
    await addSegment(segment,message);

    console.log(`Adding bookmark to table ${process.env.TABLE_NAME}`);
    await dynamodb.putItem(params).promise();
    console.log(`New bookmark added to the inventory`);
  }

  return {
    statusCode: 200,
    headers: {"Access-Control-Allow-Origin": '*'},
    body: JSON.stringify({})
  };
};

const addSegment = (segment,message) => {

    let bookmark = JSON.parse(message.body);
    const f = async (subsegment) => {

      subsegment.addAnnotation('uid',bookmark.id);
      subsegment.addAnnotation('name',bookmark.name);
      subsegment.addMetadata('bookmarkUrl', bookmark.bookmarkUrl);
      subsegment.addMetadata('username', bookmark.username);

      subsegment.close();
    };
    AWSXRay.captureAsyncFunc("adding annotations and metadata", f, segment);
};
```

The code that you added is adding annotations and metadata to the subsegments that you just reviewed within X-Ray.

80. Choose Deploy

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

Annotations are key-value pairs with string, number, or Boolean values. Annotations are indexed for use with filter expressions. Use annotations to record data that you want to use to group traces in the console or to call the **GetTraceSummaries** API.

Metadata are key-value pairs that can have values of any type, including objects and lists, but are not indexed for use with filter expressions. Use metadata to record additional data that you want stored in the trace but don't need to use with search.

Learn more For more information about adding annotations and metadata to segments with the X-Ray SDK for Node.js, see https://docs.aws.amazon.com/xray/latest/devguide/xray-sdk-nodejs-segment.html.

81. Return to the browser tab with the AWS Cloud9 IDE.

82. Run the Artillery script again by entering the following command:

```
artillery run simple-post.yaml
```

Note This script runs for 2 minutes, adding data through the API and then invoking the **createBookmark** function.

83. Return to the AWS X-Ray browser tab.

84. On the left side of the page, choose **Traces**.

85. From the URL▼ dropdown list, select **StatusCode**.

86. From the **Trace list**, select a trace ID with a **200** response.

87. Choose the **Raw data** tab.

88. Use the **Find** feature on your computer to find the word `uid` on the page.

Here you can see that annotations and metadata have now been added to this X-Ray trace.

Task 7: Monitoring and visualization

Monitoring is an important part of maintaining the reliability, availability, and performance of your AWS solutions. In this task, you see what the X-Ray service map can offer and then switch into CloudWatch ServiceLens.

89. On the X-Ray console, choose **Service map** on the left side of the page.

The service map page allows you to identify services where errors are occurring, connections with high latency, or traces for requests that were unsuccessful.

In the center of each node, the console shows the average response time and number of traces that it sent per minute during the chosen time range.

90. Choose a service node and view requests for that node.

You can use the histogram to filter traces by duration and select status codes for which you want to view traces.

The service map indicates the health of each node by color. It is based on the ratio of successful calls to errors and faults:

- **Green** is for successful calls (200).
- **Red** is for server faults (500-series errors).
- **Yellow** is for client errors (400-series errors).
- **Purple** is for throttling errors (429 Too Many Requests).

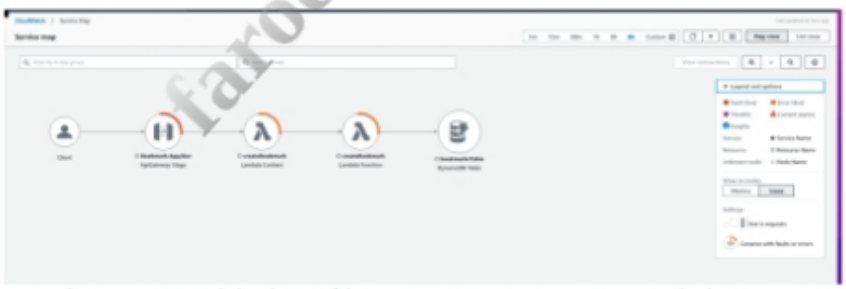
This is shown in the following image.

91. In the AWS Management Console, choose Services and open **CloudWatch** in a new browser tab.

Next, use CloudWatch ServiceLens, which gives you unified access to metrics, logs, traces, and canaries and enables performance monitoring from end user interaction to infrastructure layer insights.

92. On the left side of the page, under **Application Monitoring**, choose **ServiceLens Map**.

The following image shows the Map view within the Service Map console.



93. Choose a node to get a quick view of latency, errors, requests, and alarm summary.

94. To see latency statistics for an edge connection, choose the lines between each node that represent these connections.

95. Choose the **ApiGateway Stage** node to view its details.

96. Choose View dashboard to see more detailed metrics.

97. At the top of the page, choose the dropdown list where it shows **Bookmark App/dev**, and select the **createBookmark** function.

Notice the **Errors (4xx)** section.

98. At the top right of the page, choose View on map

99. At the top right of the page, choose View connections

This shows the incoming and outgoing connections for a node.

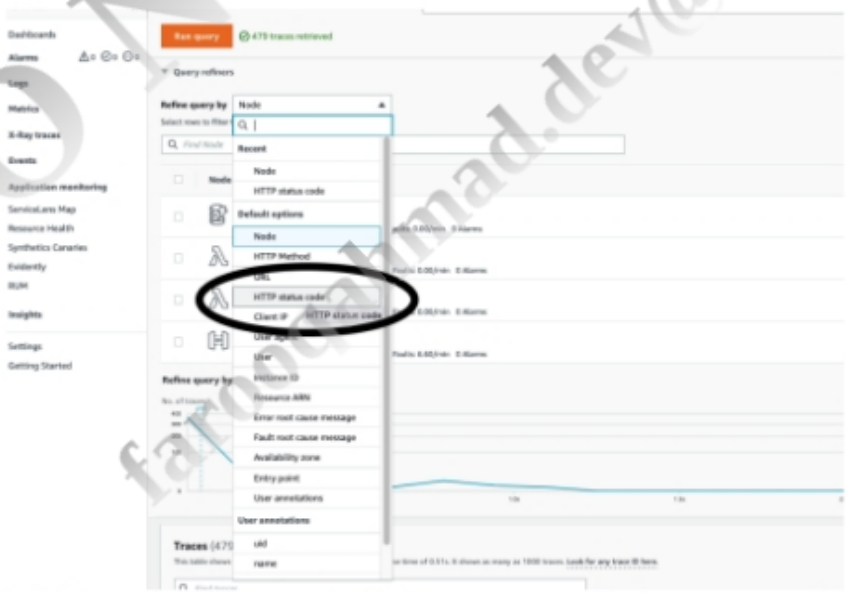
100. Choose Back to full map

101. At the top right of the page, choose List view to view the service map as a table.

102. Select **createBookmark**, with the **Type** of **Lambda Function**.

103. Choose View traces

104. Under **Query refiners**, for **Refine query by**, select **HTTP status code** from the dropdown list.



105. Navigate down the page to see the histogram representing the **response time distribution**.

106. Under the **Traces** section, select a trace ID from the list.

Here you can view the details for this specific trace.

Challenge task

For this challenge task, you troubleshoot the **createBookmark** function code to see if you can identify what is causing the 502 errors. To test to see if the code is working properly, you re-run the Artillery script and check X-Ray for errors.

107. Return to the browser tab with the **createBookmark** Lambda function.

108. In the **Code source** section, the **index.js** file is already open.

Analyze the code to determine what you can change to fix the errors. Then, update the code.

109. Once you have updated the code, choose Deploy to save your changes.

If you have successfully updated the code, you will see a message that says **Successfully updated the function createBookmark.**

110. Return to the tab with the AWS Cloud9 IDE.

111. Run the Artillery script again by entering the following:

```
artillery run simple-post.yaml
```

Note This script runs for 2 minutes, adding data through the API and then invoking the **createBookmark** function.

112. Return to the X-Ray browser tab.

113. On the left side of the page, choose **Traces**.

114. From the URL▼ dropdown list, select **StatusCode**.

115. Make sure the filter at the top right is set to **Last 5 minutes**.

If the code has been updated correctly, you should see only **200** responses populating.

Tip The section of code that you need to modify is toward the top of the function code.

Review the Solution if you have any problems.

Conclusion

Congratulations! You have successfully:

- Updated your code for better logs, applying logging best practices
- Used CloudWatch Logs and CloudWatch metrics to monitor application operations
- Used X-Ray and CloudWatch ServiceLens to troubleshoot application issues

End lab

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

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

117. At the upper-right corner of the page, choose **awsstudent@<AccountNumber>**, and then choose **Sign out**.

118. Choose End Lab.

119. Choose Submit.

120. (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 CloudWatch and X-Ray, see https://aws.amazon.com/blogs/devops/using-amazon-cloudwatch-and-amazon-sns-to-notify-when-aws-x-ray-detects-elevated-levels-of-latency-errors-and-faults-in-your-application/.
- For more information about adding annotations, see https://docs.aws.amazon.com/xray/latest/devguide/xray-sdk-nodejs-segment.html/.

Solution

Delete or comment out the following block of code to get the **createBookmark** function to work successfully.

```
  //Fail the messages randomly to see those errors in X-Ray tracing. It's for testing
only.
  if(Math.random() < 0.3)
    throw new Error('An unknown error occurred');

  //Can you throw a different response code other than 200?

  //Timeout failures about 10%
  if(Math.random() < 0.2) {
    await new Promise(resolve => setTimeout(resolve, 10000));
  };
```

aws training and certification

Lab 5: Securing Serverless Applications

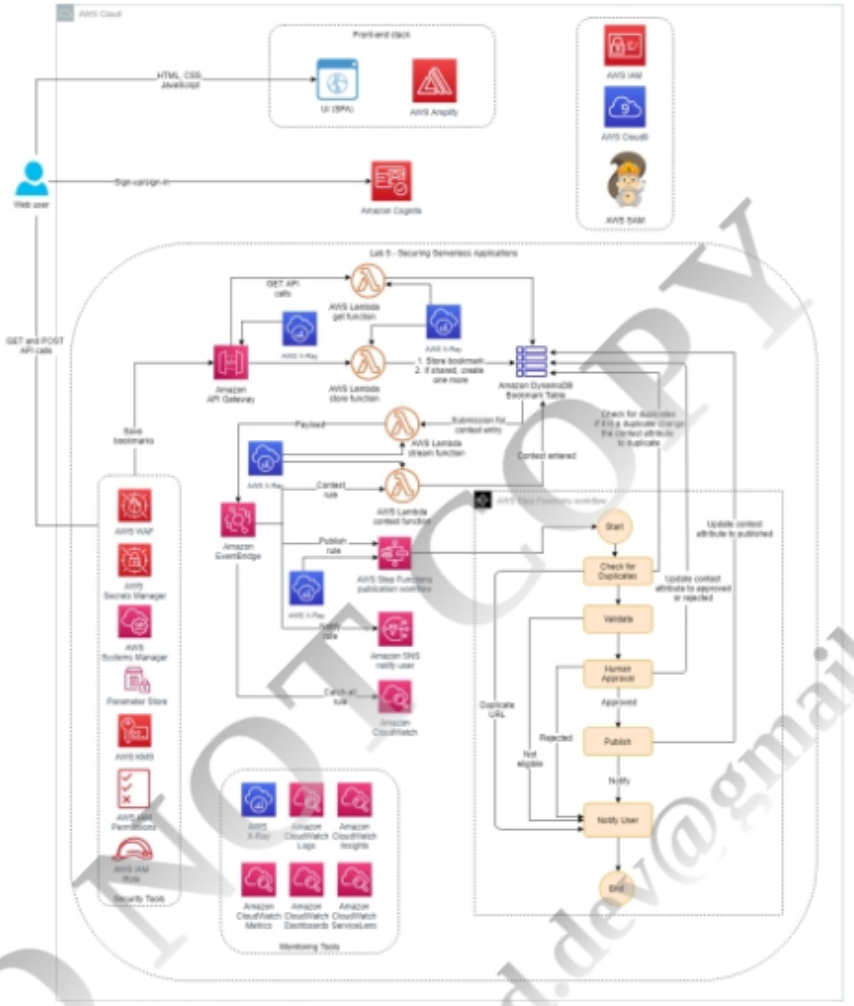
Note: Do not include any personal, identifying, or confidential information into the lab environment. Information entered may be visible to others.

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

Overview

Before you'll be able to make this application available outside of your development team, you need to review security best practices for securing access and protecting resources and data. You have successfully completed coding your application with several features and have also taken care of observability and monitoring aspects of the bookmark application. In this lab, you look into security aspects to ensure the protection of your resources and data and to avoid application outages.

The following architecture diagram shows the components that have been and will be deployed:



This lab uses the following services:

- AWS Amplify
- AWS Serverless Application Model (AWS SAM)
- Amazon Cognito
- AWS Cloud9
- Amazon DynamoDB
- Amazon EventBridge
- Amazon Simple Notification Service (Amazon SNS)
- AWS Step Functions

- AWS Lambda
- Amazon CloudWatch
- Amazon API Gateway
- AWS WAF
- AWS Key Management Service (AWS KMS)
- AWS Systems Manager Parameter Store
- AWS Secrets Manager

Objectives

After completing this lab, you will be able to:

- Secure your application with AWS WAF web ACLs
- Secure access to your API with an API Gateway resource policy
- Secure your Lambda functions and other backend services with AWS KMS, Systems Manager Parameter Store, and Secrets Manager

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 **90 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 AWS WAF and securing the application with web ACLs

AWS WAF is a web application firewall that helps protect your web applications or APIs against common web exploits that may affect availability, compromise security, or consume excessive resources. AWS WAF gives you control over how traffic reaches your applications by enabling you to create security rules that block common attack patterns, such as SQL injection or cross-site scripting, and rules that filter out specific traffic patterns you define.

You can get started quickly using AWS Managed Rules for AWS WAF, a pre-configured set of rules managed by AWS or AWS Marketplace sellers. AWS Managed Rules for AWS WAF address issues like the OWASP Top 10 security risks. These rules are regularly updated as new issues emerge. AWS WAF includes a full-featured API that you can use to automate the creation, deployment, and maintenance of security rules.

First, you launch the application by running a pre-programmed script via **AWS Cloud9**.

4. In the **AWS Management Console**, choose Services and select **Cloud9**.
5. In the left navigation pane, choose the ☰ icon to expand the menu, and choose **Your environments**.

If the menu is already expanded, move onto the next step.

6. 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.

7. In the AWS Cloud9 terminal, run the following command 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-5-Security/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.

What is the script doing?

- This script is modifying the **samconfig.toml** file within the backend portion of the app code.

- It is replacing values such as AWS Region, stack name, and role Amazon Resource Name (ARN), among others.
- It then updates the **aws-exports.js** file with the Amazon Cognito metadata that was launched in the lab AWS CloudFormation template.
- It then runs **npm build**, deploys the bookmark app, and uploads the **app.zip** file to the **samserverless** Amazon Simple Storage Service (Amazon S3) bucket.

Note Leave this page open.

Task 1.1: Securing with AWS WAF web ACLs

A web access control list (web ACL) has a capacity of 1,500. You can add hundreds of rules and rule groups to a web ACL. The total number that you can add is based on the complexity and capacity of each rule.

A rate-based rule tracks the rate of requests for each originating IP address and triggers the rule action on IPs with rates that exceed a limit. You set the limit as the number of requests per a 5-minute time span. You can use this type of rule to put a temporary block on requests from an IP address that's sending excessive requests. By default, AWS WAF aggregates requests based on the IP address from the web request origin, but you can configure the rule to use an IP address from an HTTP header, such as X-Forwarded-For, instead.

When the rule action triggers, AWS WAF applies the action to additional requests from the IP address until the request rate falls below the limit. It can take a minute or two for the action change to go into effect.

In this task, you create a web ACL to secure the API Gateway resources using AWS WAF.

Create a web ACL

8. In the AWS Management Console, choose Services and open **WAF & Shield** in a new browser tab.
9. Choose **Create web ACL**
10. In the **Web ACL details** section, configure the following details:

- **Region:** Choose the Region from the dropdown list. The Region should be the one displayed in the left column of the lab instructions.

Note If the Region is not set to the value displayed in the left column of the lab instructions, you need to change the Region before filling in the other fields. Changing the Region clears the other field values if you have entered any information before selecting the Region. If you are not sure about the Region, choose the Global Region dropdown list in the AWS Management Console.

- **Name:** Enter `BookmarkACL`
- **Description:** Enter `Block actions from the API Gateway`
- **CloudWatch metric name:** Enter `BookmarkACL` (pre-populated with the text entered for the **Name** value)
- **Resource Type:** Choose **Regional resources (Application Load Balancer, API Gateway, AWS AppSync)**

11. Choose **Next**
12. In the **Rules** section, configure the following details:

- Choose **Add rules** ▼ and select **Add my own rules and rule groups** from the dropdown list.
- Leave the default **Rule type** as **Rule builder**.

13. In the **Rule builder** section, configure the following details:

- **Name:** Enter `100ratebasedrule`
- **Type:** Select **Rate-based rule**

14. In the **Request rate details** section, configure the following details:

- **Rate limit:** Enter `100`
- **IP address to use for rate limiting:** Leave the default value as **Source IP address**
- **Criteria to count request towards rate limit:** Leave the default value as **Consider all requests**

15. In the **Then** section, leave the default value of **Action** as **Block**.
16. Choose **Add rule**
17. On the **Add rules and rule groups** page, choose **Next**
18. On the **Set rule priority** page, choose **Next**
19. On the **Configure metrics** page, choose **Next**
20. On the **Review and create web ACL** page, scroll to the bottom and choose **Create web ACL**

The web ACL has been created, and an ID has been generated for the web ACL.

Note Leave this page open.

Attach the web ACL to API Gateway

21. In the AWS Management Console, choose Services and open **API Gateway** in a new browser tab.
22. Choose the **Bookmark App**.
23. In the left navigation pane, choose **Stages**.
24. In the **Stages** pane, choose the ▶ **dev** stage.

You see the API Gateway end point has been deployed successfully.

25. In the **Web Application Firewall (WAF)** section, from the **Web ACL** dropdown list, choose **BookmarkACL**.
26. Choose **Save changes**

The web ACL has been assigned to the **dev** stage of the bookmark application.

27. If you receive an error stating, **AWS WAF couldn't retrieve the resource that you requested. Retry your request.**, close the error and save the changes again.

Note Leave this page open.

Test the web ACL using Artillery

First, you load the bookmark data using the Artillery tool and then invoke a load test.

28. In the AWS Cloud9 console, choose the **>** arrow next to the **app-code** folder to expand it.
29. Choose the **>** arrow next to the **test** folder to expand it.
30. Open the **simple-post.yaml** file.
31. In the AWS Cloud9 terminal, run the following AWS CLI and bash commands to replace the *API_GATEWAY_URL* with the actual value.

```
cd test
echo export API_GATEWAY_ID=$(aws apigateway get-rest-apis --query
'items[?name==`Bookmark App`].id' --output text) >> ~/environment/app-
code/labVariables
echo export AWS_REGION=$(curl -s 169.254.169.254/latest/dynamic/instance-
identity/document | jq -r '.region') >> ~/environment/app-code/labVariables
source ~/environment/app-code/labVariables
echo export API_GATEWAY_URL=https://${API_GATEWAY_ID}.execute-
api.${AWS_REGION}.amazonaws.com/dev >> ~/environment/app-code/labVariables
source ~/environment/app-code/labVariables
sed -Ei "s|<API_GATEWAY_URL>|${API_GATEWAY_URL}|g" simple-post.yaml
cd ..
```

Note The script is running the AWS CLI command the API Gateway ID and the AWS region to construct the API Gateway URL. This URL is then substituted in the placeholder <API_GATEWAY_URL> in the **simple-post.yaml** file.

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

33. In the AWS Cloud9 terminal, run the following code to install Artillery, install Faker, and run the script.

```
cd test
npm install artillery@1.7.9 -g
npm install faker@5.5.3
artillery run simple-post.yaml
```

This script runs for 30 seconds, adding data through the API and then invoking the createBookmark Lambda function.

In the next steps, you run a curl command to retrieve one of the bookmark details that was added in the above step.

34. In the AWS Management Console, choose Services and open **DynamoDB** in a new tab.

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

36. Choose the **bookmarksTable**.

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

You see the list of items added to the **bookmarksTable** by the artillery run.

38. In the AWS Cloud9 terminal, run the following AWS CLI and bash commands to retrieve the bookmark details of an item from the **bookmarksTable** and replace the value in the curl command:

```
source ~/environment/app-code/labVariables
echo export ID=$(aws dynamodb scan --table-name sam-bookmark-app-bookmarksTable --
query Items[0].id --output text) >> ~/environment/app-code/labVariables
source ~/environment/app-code/labVariables
curl ${API_GATEWAY_URL}/bookmarks/${ID}
```

Note The *API_GATEWAY_URL* value has been fetched in the above steps and stored in the *labVariables* file. The *labVariables* file contains all of the values fetched so far using the AWS CLI commands.

The **curl** command invokes the **getBookmark** Lambda function to retrieve the bookmark details of the provided item id.

The bookmark details are displayed in a JSON format in the terminal.

39. In the AWS Cloud9 terminal, run the following commands to test the BookmarkACL.

```
source ~/environment/app-code/labVariables
artillery quick -n 20 --count 100 ${API_GATEWAY_URL}/bookmarks
```

Note This command creates a load test of 100 users with 20 requests from each user because the goal is to reach 100 requests per minute. The 200 status code for each request indicates that the request has been successful.

40. Run the following curl command to retrieve the bookmark data and test the web ACL.

```
source ~/environment/app-code/labVariables
curl ${API_GATEWAY_URL}/bookmarks/${ID}
```

41. The following message is displayed in the console.

```
{"message": "Forbidden"}
```

This message is displayed because the web ACL you created is blocking the request to the bookmark application. Specifying conditions in a web ACL allows you to protect your application resources.

API Gateway rejects any further calls after the first 100 requests made in 1 minute. When AWS WAF sees continuous requests from the same source IP address, it blocks all future calls based on the rule. After a few minutes, AWS WAF releases the restrictions and automatically lets the calls in.

You can check the requests that are blocked by the web ACL in the AWS WAF console.

42. To check these requests, navigate to the AWS WAF console, and choose **BookmarkACL**.

43. In the **Sampled requests** pane, you see all the requests by default.

44. To see the blocked requests, choose **100ratebasedrule** from the dropdown list.

Now, you see the requests in the **BLOCK** state. As AWS WAF releases the restrictions after a few minutes, you invoke the API Gateway endpoint to see if the request is still being blocked.

Note It might take up to 5 minutes for AWS WAF to release the restrictions.

45. In the AWS Cloud9 terminal, run the following curl command to test the release:

```
source ~/environment/app-code/labVariables
curl ${API_GATEWAY_URL}/bookmarks/${ID}
```

This command displays the bookmark details for the provided **id**.

Note If you do not see the bookmark details, wait until the restrictions are released, and run the above curl command again.

Task 1.2: Securing with AWS WAF using an IP Address

In this section, you create an IP set that contains an IP address from which you want to block any requests to your application.

46. Navigate to the AWS WAF page, and in the left navigation pane, choose **IP sets**.

47. Choose Create IP set

48. In the **IP set details** section, configure the following information:

- **Region:** Choose the Region from the dropdown list. The Region should be the one displayed in the left column of the lab instructions.

Note If the Region is not set to the value displayed in the left column of the lab instructions, you need to change the Region before filling in the other fields. Changing the Region clears the other field values if you have entered any information before selecting the Region. If you are not sure about the Region, choose the Global Region dropdown list in the AWS Management Console.

- **IP set name:** Enter `IPToBlock`
- **Description:** Enter `Block this IP address`
- **IP version:** Leave the default value
- **IP addresses:** In the AWS Cloud9 terminal, run the following command to get the IP address of the server:

```
curl https://checkip.amazonaws.com/
```

Copy the IP address displayed in the terminal, and paste it into the **IP addresses** field on the **Create IP set** page. Append `/32` to the end of the IP address.

Note Without the CIDR block of `/32`, an error will be thrown if the **Create IP set** is chosen.

49. On the AWS WAF page, choose Create IP set

An IP set has been successfully created. Now, you need to attach the IP set to a web ACL.

50. In the left navigation pane, choose **Web ACLs**.

51. Choose **Create web ACL**

52. In the **Web ACL Details** section, configure the following details:

- **Region:** Choose the Region from the dropdown list. The Region should be the one displayed in the left column of the lab instructions.

Note If the Region is not set to the value displayed in the left column of the lab instructions, you need to change the Region before filling in the other fields. Changing the Region clears the

other field values if you have entered any before selecting the Region. If you are not sure about the Region, choose the Global Region dropdown list in the AWS Management Console.

- **Name:** Enter `IPsetbasedACL`
- **Description:** Enter `Blocks actions from the specified IP address`
- **CloudWatch metric name:** Enter `IPsetbasedACL` (pre-populated with the text entered for the **Name** value)
- **Resource Type:** Choose **Regional resources (Application Load Balancer, API Gateway, AWS AppSync)**

53. Choose **Next**
54. In the **Rules** section, configure the following details:

- Choose **Add rules ▼** and select **Add my own rules and rule groups** from the dropdown list.

55. In the **Rule type** section, choose **IP set**.
56. In the **Rule** section, configure the following information:

- **Name:** Enter `IPbasedrule`

57. In the **IP set** section, configure the information below:

- **IP set:** In the dropdown list, choose **IPToBlock**
- **IP address to use as the originating address:** Leave the default value as **Source IP address**
- **Action:** Leave the default value as **Block**

58. Choose **Add rule**
59. On the **Add rules and rule groups** page, choose **Next**
60. On the **Set rule priority** page, choose **Next**
61. On the **Configure metrics** page, choose **Next**
62. On the **Review and create web ACL** page, scroll to the bottom and choose **Create web ACL**

The web ACL has been created, and an ID has been generated for the web ACL.

Note Leave this page open.

Attach the web ACL to API Gateway

63. Navigate to the API Gateway page you left open earlier in this task.
64. Refresh this page so that the **IPsetbasedACL** web ACL created earlier appears in the **Web ACL** dropdown list.
65. In the **Web Application Firewall (WAF)** section, from the **Web ACL** dropdown list, select **IPsetbasedACL**.
66. Choose **Save changes**

The web ACL has been assigned to the **dev** stage of the bookmark application.

Note Leave this page open.

Test the web ACL using curl

67. In the AWS Cloud9 terminal, run the curl command you updated earlier:

```
source ~/environment/app-code/labVariables
curl ${API_GATEWAY_URL}/bookmarks/${ID}
```

68. The following command is displayed on the terminal because the web ACL is blocking the requests from your IP address:

```
{"message": "Forbidden"}
```

Before you proceed to the next task, you must remove the above web ACL because it blocks requests to the application from your IP address.

69. Navigate to the API Gateway page you left open earlier in this task.
70. Make sure that you are in the **dev** stage and the **Settings** tab for this stage. In the **Web Application Firewall (WAF)** section, from the **Web ACL** dropdown list, select **None**.
71. Choose **Save changes**

This step disassociates the web ACL from the **dev** stage of the bookmark application.

Task 2: Securing the application with API Gateway resource policies

API Gateway resource policies are JSON policy documents that you attach to an API to control whether a specified principal (typically an AWS Identity and Access Management [IAM] user or role) can invoke the API. You can use API Gateway resource policies to allow your API to be securely invoked by the following:

- Users from a specified AWS account

- Specified source IP address ranges or CIDR blocks
- Specified virtual private clouds (VPCs) or VPC endpoints (in any account)

You can attach a resource policy to an API by using the AWS Management Console, AWS Command Line Interface (AWS CLI), or AWS SDKs.

API Gateway resource policies are different from IAM policies. IAM policies are attached to IAM entities (users, groups, or roles) and define what actions those entities are capable of doing on which resources. API Gateway resource policies are attached to resources. For a more detailed discussion of the differences between identity-based (IAM) policies and resource policies, see Identity-Based Policies and Resource-Based Policies.

You can use API Gateway resource policies together with IAM policies.

In this task, you learn how to add certain IP addresses or a range of IP addresses to an allow list to access your API Gateway resources. You create a resource policy for the bookmark API that denies access to any IP address that isn't specifically allowed.

72. Choose the following URL to find the IP address of your system:

https://checkip.amazonaws.com/

Note The IP address of your system is displayed in a new browser tab.

73. Copy and paste the IP address into a text editor. You use it in the **Resource Policy** you define in the next few steps.

74. Navigate to the API Gateway page you left open earlier.

75. In the left navigation pane, choose **Resource Policy**.

76. Copy and paste the following policy into the text editor:

```
{
  "Version": "2012-10-17",
  "Statement": [{
      "Effect": "Allow",
      "Principal": "*",
      "Action": "execute-api:Invoke",
      "Resource": "execute-api:/*/*/*"
    },
    {
      "Effect": "Deny",
      "Principal": "*",
      "Action": "execute-api:Invoke",
      "Resource": "execute-api:/*/*/*",
      "Condition": {
        "NotIpAddress": {
          "aws:SourceIp": ["SOURCEIPORCIDRBLOCK"]
        }
      }
    }
  ]
```

```
}
```

77. In the last line of the policy, replace *SOURCEIPORCIDRBLOCK* with the IP address you noted earlier.

78. Choose **Save**

Now, you need to deploy the changes.

79. In the left navigation pane, choose **Resources**.

80. In the **Resources** pane, choose **Actions** ▼ and select **Deploy API**.

81. In the pop-up window, for **Deployment Stage** in the dropdown list, choose **dev**.

82. Choose **Deploy**

This step deploys the resource policy you just created. Now, you test whether the resource policy is working or not.

83. In the AWS Cloud9 terminal, run the following curl command to test the resource policy.

```
source ~/environment/app-code/labVariables
curl ${API_GATEWAY_URL}/bookmarks/${ID}
```

In the console, an error message similar to the following message is displayed:

```
{"Message":"User: anonymous is not authorized to perform: execute-api:Invoke on
resource: arn:aws:execute-
api:(AWS:Region):*******1234:(BookmarkAppID)/dev/GET/bookmarks/(bookmarkId) with an
explicit deny"}
```

Note If you do not see the expected result, wait a few seconds and try again.

This message is displayed because you created a resource policy to deny all the requests from other than your system's IP address.

Before you proceed to the next task, you should remove the above resource policy because it blocks the application from being invoked.

84. In the API Gateway console in the left navigation pane, choose **Resource Policy**.

85. Select the entire policy in the text editor, and press **Delete**.

86. Choose **Save**

Now, you need to deploy the changes.

87. In the left navigation pane, choose **Resources**.

88. In the **Resources** pane, choose **Actions** ▼ and select **Deploy API**.

89. In the pop-up window, for **Deployment Stage** in the dropdown list, choose **dev**.

90. Choose **Deploy**

This step deploys the changes you just made.

91. In the AWS Cloud9 terminal, run the following curl command to test if the policy was removed successfully.

```
source ~/environment/app-code/labVariables
curl ${API_GATEWAY_URL}/bookmarks/${ID}
```

The console now displays the bookmark details for the **id** provided in the above command returned by API Gateway.

Note If you do not see the expected result, wait a few seconds and try again. Leave this page open.

Task 3: Securing an AWS Lambda function

A highly recommended best practice is to never store your secrets or passwords in plain text or hard code them as part of the function code. They should always be encrypted to secure them from attacks. The following are a few widely used tools to manage your secrets:

- AWS KMS
- Parameter Store
- Secrets Manager

AWS KMS makes it easy for you to create and manage cryptographic keys and control their use across a wide range of AWS services and in your applications. AWS KMS is a secure and resilient service that uses hardware security modules that have been validated under FIPS 140-2, or are in the process of being validated, to protect your keys. AWS KMS is integrated with AWS CloudTrail to provide you with logs of all key use in order to help meet your regulatory and compliance needs.

Parameter Store provides secure, hierarchical storage for configuration data management and secrets management. You can store data such as passwords, database strings, Amazon Machine Image (AMI) IDs, and license codes as parameter values. You can store values as plain text or encrypted data. You can reference Systems Manager parameters in your scripts, commands, Systems Manager documents, and configuration and automation workflows by using the unique name that you specified when you created the parameter.

Secrets Manager helps you protect secrets needed to access your applications, services, and IT resources. The service enables you to easily rotate, manage, and retrieve database credentials, API keys, and other secrets throughout their lifecycle. Users and applications retrieve secrets with a call to Secrets Manager APIs, eliminating the need to hard code sensitive information in plain text. Secrets Manager offers secret rotation with built-in integration for Amazon Relational Database Service (Amazon RDS), Amazon Redshift, and Amazon DocumentDB. Also, the service is extensible to other types of secrets, including API keys and

OAuth tokens. In addition, Secrets Manager enables you to control access to secrets using fine-grained permissions and to audit secret rotation centrally for resources in the AWS Cloud, third-party services, and on premises.

In this task, you learn how to secure secrets using AWS KMS, Parameter Store, and Secrets Manager and retrieve the secrets in your Lambda code.

Task 3.1: Securing environment variables using AWS KMS

92. In the AWS Management Console, choose Services and open Key **Management Service** in a new tab.

93. In the left navigation pane, choose **Customer managed keys**.

94. Choose Create key

95. In the **Configure key** section, leave the default option for **Key type** as **Symmetric**.

96. Choose Next

97. In the **Add labels** section, configure the following details:

- **Alias:** Enter `LambdaSecrets`
- **Description:** Enter `Creating a Lambda secrets key`

98. Choose Next

99. In the **Key administrators** section, select the user or role you are logged in with.

The user or role is displayed at the top-right of your screen.

100. Choose Next

101. In the **This account** section, select the user or role you are logged in with.

102. Choose Next

Review the policy in the **Key policy** section.

103. Choose Finish

104. In the AWS Cloud9 terminal, run the following commands.

```
export KeyId=$(aws kms list-aliases --query 'Aliases[?starts_with(AliasName,
`alias/LambdaSecrets`)].TargetKeyId' --output text)
aws kms encrypt --plaintext "Key Management Service Secrets" --query CiphertextBlob --
output text --key-id $(KeyId)
```

You are encrypting the string **Key Management Service Secrets** using the AWS KMS key ID.

105. Copy and paste the base64 encoded output from the console into a text editor.

The resulting encoded output is base64 encoded and is provided as an environment variable to your Lambda function. The Lambda function decrypts the data to get the plaintext in order to actually use it.

Create a Lambda function and configure it in API Gateway to test the AWS KMS

In this section, you create a new Lambda function to test the AWS KMS secrets.

106. In the AWS Management Console, choose Services and then choose **Lambda**.

107. Choose Create function

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

- **Function name:** Enter `sam-bookmark-app-secrets-function`
- **Runtime:** Choose **Node.js 14.x**

109. Expand ▶ **Change default execution role**, and configure the following information:

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

110. Choose Create function

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

112. Delete the existing code snippet and paste the following code into the code box:

```
const aws = require('aws-sdk');

const kmsSecret = process.env.KMS_SECRET;

let decodedSecret;
let DecodedKMSSecret;

const kms = new aws.KMS();
const ssm = new aws.SSM();
const sm = new aws.SecretsManager();

exports.handler = async message => {
    console.log(message);
    let secretType = message.pathParameters.id
    console.log("Secret Type:", secretType);

    if(secretType == 'kms')
        decodedSecret = await decodeKMSSecret();
    else if (secretType == 'ssm')
        decodedSecret = await decodeSSMSecret();
    else if (secretType == 'sm') {
        var password = await decodeSMSecret(userId);
        decodedSecret = "Password is: " + password;
```

```
    }
    else
        decodedSecret = "Provide a valid secret type (kms, ssm, or sm (secrets
manager))";

    console.log(decodedSecret);
    const response = {
        statusCode: 200,
        headers: {},
        body: JSON.stringify('Plain text secret(s): ' + decodedSecret)
    };
    return response;
};

async function decodeKMSSecret() {
    if (DecodedKMSSecret) {
        return DecodedKMSSecret;
    }
    const params = {
      CiphertextBlob: Buffer.from(kmsSecret, 'base64')
    };
    const data = await kms.decrypt(params).promise();
    DecodedKMSSecret = data.Plaintext.toString('utf-8');
    return DecodedKMSSecret;
}
```

113. Choose Deploy

You should see a message that says **Successfully updated the function sam-bookmark-app-secrets-function.**

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

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

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

117. In the **Edit environment variables** page, choose Add environment variable and configure the following details:

- **Key:** Enter `KMS_SECRET`
- **Value:** Enter the base64 encoded output that you pasted into a text editor earlier

118. Choose Save

Note This Lambda function shows how to use the AWS KMS SDK to read secrets. Review the function code after it is deployed.

Note To read the secrets, an IAM permission, **kms:Decrypt**, is needed for the Lambda function. For the purposes of this lab, the permission has already been added to the **LambdaDeploymentRole**, which is assigned to the Lambda function during the pre-build lab process.

To test this Lambda function, configure it in API Gateway.

119. Navigate to the API Gateway page you left open at the end of task 2.

120. In the left navigation pane, choose **Resources**.

121. In the **Resources** pane, choose the **/** endpoint, and then choose **Actions ▼**

122. In the dropdown list, choose **Create Resource**.

123. In the right navigation pane, enter `secrets` for the **Resource Name**.

124. Choose **Create Resource**

Once the resource creation is complete, the **secrets** resource appears in the **Resources** pane.

125. In the **Resources** pane, choose the **/secrets** endpoint, and then choose **Actions ▼**

126. In the dropdown list, choose **Create Resource**.

127. In the right navigation pane, enter `{id}` for the **Resource Name**.

128. In the **Resource Path**, the value is set to **-id-** by default when you enter the **Resource Name**. Replace it with `{id}`

129. Choose **Create Resource**

Once the resource creation is complete, the **{id}** resource appears under the **secrets** resource in the **Resources** pane.

130. In the **Resources** pane, choose the **/secrets/{id}** endpoint, and then choose **Actions ▼**

131. In the dropdown list, choose **Create Method**.

132. In the dropdown menu under the **/secrets/{id}** endpoint, choose the **GET** method, and then choose .

Note If you see a warning message on top of the pop-up window indicating **Invalid model identifier specified: Empty**, choose the **x** on the window.

133. In the right navigation pane, select the check box for **Use Lambda Proxy integration**.

134. For **Lambda Function**, enter `sam-bookmark-app-secrets-function`

This is the new Lambda function name that you just created.

135. Choose **Save**

A window pops up with a message that reads **Add Permission to Lambda Function**.

136. Choose **OK**

The Lambda function has been integrated into the new API endpoint. Now, the new endpoint should be deployed in order to test the function.

137. In the **Resources** pane, choose **Actions** ▼

138. In the dropdown list, choose **Deploy API**.

139. In the **Deploy API** pop-up window, from the **Deployment stage** dropdown list, choose **dev**.

140. Choose **Deploy**

This step has successfully deployed the new endpoint to the **dev** stage.

141. In the **Stages** pane, expand the ▶ **dev** stage.

142. Under the **/secrets/{id}** endpoint, choose **GET**.

143. Copy and paste the **Invoke URL** value into a text editor.

144. Replace *{id}* with `kms` in this **Invoke URL**.

145. Copy the updated **Invoke URL**, paste it into a new browser window, and press **Enter**.

The browser displays the following message from the Lambda function code:

```
"Plain text secret(s): Key Management Service Secrets"
```

The AWS KMS secret has been successfully decoded and is displaying the plain text.

If you enter a value other than **kms**, you will see the following message:

```
"Plain text secret(s): Provide a valid secret type (kms, ssm or sm (secrets manager))"
```

Task 3.2: Storing and accessing passwords using Systems Manager Parameter Store

The Parameter Store is part of Systems Manager. It is used to store not only encrypted secrets but almost any data. IAM permissions can control who is able to access and change the data in a very fine-grained way. For each record in the Parameter Store, a history is stored, which makes it possible to know exactly when a change has occurred.

146. In the AWS Cloud9 terminal, run the following command to store secrets in the Parameter Store:

```
aws ssm put-parameter --name /db/secret --value 'Hello, Parameter Store!' --type
SecureString
```

You are storing a secret with the name **/db/secret** that has a value of **Hello, Parameter Store!**

The following output is displayed:

```
{
    "Version": 1,
```

```
    "Tier": "Standard"
}
```

View the Parameter Store in the AWS Management Console

147. In the AWS Management Console, choose Services and then choose **Systems Manager**.

148. In the left navigation pane, choose **Parameter Store**.

The **/db/secret** secret you created earlier is displayed here.

149. To view the details of the secret, choose **/db/secret**.

150. To view the value of the secret, choose **Show**.

Test the secret using a Lambda function

Update the **sam-bookmark-app-secrets-function** function code to test the new parameter.

151. In the AWS Management Console, choose Services and then choose **Lambda**.

152. Enter `sam-bookmark-app-secrets-function` into the box, and select this function.

153. In the **Code source** section, add the following to the **index.js** file:

- Add the following constant after the kmsSecret constant (line 4):

```
const ssmSecret = process.env.SSM_SECRET;
```

- Add the following code snippet to the end of the existing code:

```
async function decodeSSMSecret() {
    const params = {
        Name: ssmSecret,
        WithDecryption: true
    };
    const result = await ssm.getParameter(params).promise();
    return result.Parameter.Value
}
```

The code should look like the following after you add the previous two snippets:

```
const aws = require('aws-sdk');

const kmsSecret = process.env.KMS_SECRET;
const ssmSecret = process.env.SSM_SECRET;

let decodedSecret;
let DecodedKMSSecret;

const kms = new aws.KMS();
const ssm = new aws.SSM();
```

```
const sm = new aws.SecretsManager();

exports.handler = async message => {
    console.log(message);
    let secretType = message.pathParameters.id
    console.log("Secret Type:", secretType);

    if(secretType == 'kms')
        decodedSecret = await decodeKMSSecret();
    else if (secretType == 'ssm')
        decodedSecret = await decodeSSMSecret();
    else if (secretType == 'sm') {
        var password = await decodeSMSecret(userId);
        decodedSecret = "Password is: " + password;
    }
    else
        decodedSecret = "Provide a valid secret type (kms, ssm, or sm (secrets
manager))";

    console.log(decodedSecret);
    const response = {
        statusCode: 200,
        headers: {},
        body: JSON.stringify('Plain text secret(s): ' + decodedSecret)
    };
    return response;
};

async function decodeKMSSecret() {
    if (DecodedKMSSecret) {
        return DecodedKMSSecret;
    }
    const params = {
      CiphertextBlob: Buffer.from(kmsSecret, 'base64')
    };
    const data = await kms.decrypt(params).promise();
    DecodedKMSSecret = data.Plaintext.toString('utf-8');
    return DecodedKMSSecret;
}

async function decodeSSMSecret() {
    const params = {
        Name: ssmSecret,
        WithDecryption: true
    };
    const result = await ssm.getParameter(params).promise();
    return result.Parameter.Value
}
```

154. Choose Deploy

You should see a message that says **Successfully updated the function sam-bookmark-app-secrets-function.**

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

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

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

158. In the **Edit environment variables** page, choose Add environment variable and configure the following details:

- **Key:** Enter `SSM_SECRET`
- **Value:** Enter `/db/secret`

159. Choose Save

The Lambda function has been updated to read the Parameter Store and display the password.

Note There is an IAM permission, **ssm:GetParameter**, needed for the Lambda function to read Systems Manager. This permission has been added to the **LambdaDeploymentRole** in the pre-build process of the lab.

160. To test the above changes, go to the text editor with the **Invoke URL** that you saved from task 3.1. In the **Invoke URL**, replace *{id}* with `ssm`

161. Copy and paste the updated **Invoke URL** into a new browser tab, and press **Enter**.

You should see the following text in the browser.

```
"Plain text secret(s): Hello, Parameter Store!"
```

Task 3.3: Storing a secret using Secrets Manager

Secrets Manager helps you meet your security and compliance requirements by enabling you to rotate secrets safely without the need for code deployments. You can store and retrieve secrets using the Secrets Manager console, AWS SDK, AWS CLI, or CloudFormation. To retrieve secrets, you simply replace plaintext secrets in your applications with code to pull in those secrets programmatically using the Secrets Manager APIs.

162. In the AWS Cloud9 terminal, run the following command to create a secret:

```
aws secretsmanager create-secret --name dbUserId --secret-string
"secretsmanagerpassword"
```

This is similar to storing database credentials. Here the UserId is **dbUserId** with string of **secretsmanagerpassword**.

The following output is displayed:

```
{
    "ARN": "arn:aws:secretsmanager:us-west-2:{AWS::AccountId}:secret:dbUserId-xxxxxx",
    "Name": "dbUserId",
    "VersionId": "xxxxx"
}
```