Using AWS Lambda with Amazon CloudWatch and SNS to Implement a Slack Chat Bot

**SPL-115 Version 2.1.16**

© 2023 Amazon Web Services, Inc. or its affiliates. All rights reserved. This work may not be reproduced or redistributed, in whole or in part, without prior written permission from Amazon Web Services, Inc. Commercial copying, lending, or selling is prohibited. All trademarks are the property of their owners.

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

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

**Overview**

In this lab you will build a chat bot for Slack, using a Lambda blueprint. Chat bots have the ability to interact with teams and users, respond to commands, and post notifications, giving all conversation participants visibility into team activities. You will build a bot that posts to your Slack channel when it receives CloudWatch alarms.

TOPICS COVERED

By the end of this lab, you will be able to:

* Create a Slack chat bot using a Lambda blueprint
* Configure the bot with a Slack webhook to post messages to a Slack channel

TECHNICAL KNOWLEDGE PREREQUISITES

To successfully complete this lab, you should be familiar with AWS Lambda and Amazon CloudWatch.Familiarity with Slack is helpful, though not required.

**Start lab**

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

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

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

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

**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 Your Slack Account**

First, you will need to create a Slack account so your bot can interact with it. You can do this prior to starting the lab to save time. If you already have a Slack account that you plan to use for this lab you can skip to the next [Section](https://labs.skillbuilder.aws/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A470679935125%3Ablueprintversion%2Fspl-115%3A2.1.16-8835d69a/en-US#section).

1. In a new browser tab, go to <https://slack.com/get-started#/createnew>.
2. Enter your email address, then choose **Continue**.
3. Check your email, and use the confirmation code to confirm your account.
4. Choose **Create a Workspace**.
5. For *What’s the name of your company or team?*, enter your company name, then choose **Next**.
6. If asked, **Let’s find the people you work with most**, choose **Skip this step**.
7. For *What’s a project your team is working on?*, enter a project name, then choose **Next**.
8. For *Who do you email most about …*, choose **skip this step**.
9. Choose **Skip Step**.
10. Choose your channel on the left.

**Task 2: Configure Incoming WebHooks For Slack**

1. Sign into your  [Slack](https://slack.com/) account if you haven’t already.
2. In the navigation menu on the left, choose **Browse Slack**.
3. Choose **Apps**.
4. In the Search  box, type

Incoming WebHooks

1. Under **Incoming WebHooks**, choose **Add**.
2. Choose **Add to Slack**.
3. Choose the **Post to channel** drop-down  and select **#general**.
4. Choose **Add Incoming WebHooks integration**.
5. Copy the **Webhook URL** to a text editor.

You will need this later.

1. Scroll to the bottom, then choose **Save Settings**.
2. Leave this browser tab/window open, you will return to it later.

**Task 3: Create and Subscribe to an SNS Topic**

In this task, you will create an SNS topic and subscribe to the topic using your email address.

Amazon Simple Notification Service (SNS) is a flexible, fully managed pub/sub messaging and mobile notifications service for coordinating the delivery of messages to subscribing endpoints and clients. With SNS you can fan-out messages to a large number of subscribers, including distributed systems and services, and mobile devices. It is easy to set up, operate, and reliably send notifications to all your endpoints – at any scale. You can get started using SNS in a matter of minutes using the AWS Management Console, AWS Command Line Interface, or using the AWS SDK with just three simple APIs. SNS eliminates the complexity and overhead associated with managing and operating dedicated messaging software and infrastructure.

1. At the top of the page, in the unified search bar, search for and choose **Simple Notification Service**.
2. In the left navigation pane, choose the three line bars.
3. Choose **Topics**.
4. Choose **Create topic**
5. In the **Create topic** window, configure:

* **Type:**  *Standard*
* **Name:**

slacknews

* **Display name**

slacknews

* Choose **Create topic**

1. Choose **Create subscription** then configure:

* **Protocol:** *Email*
* **Endpoint:** *Enter your email address*
* **Create subscription**

1. Check your email for the **slacknews** confirmation email.

It might take a few minutes for you to receive an email.

1. Choose the **Confirm subscription** link in the **slacknews** email to confirm the subscription.

 Congratulations! Your SNS topic has been created and you are subscribed to it.

**Task 4: Create a Lambda Function**

AWS Lambda lets you run code without provisioning or managing servers. You pay only for the compute time you consume - there is no charge when your code is not running. With Lambda, you can run code for virtually any type of application or backend service - all with zero administration. Just upload your code and Lambda takes care of everything required to run and scale your code with high availability. You can set up your code to automatically trigger from other AWS services or call it directly from any web or mobile app.

In this task, you will create a Lambda function from a blueprint that posts notifications to Slack, based on your SNS topic.

1. At the top of the page, in the unified search bar, search for and choose **Lambda**.
2. Choose **Create function**
3. Choose **Use a blueprint**.

 Blueprints are sample configurations of event sources and Lambda functions that do minimal processing for you, then customize it as needed. When you create a new AWS Lambda function, you can use a blueprint that best aligns with your scenario. There are several Slack bot blueprints available, including functions that handle a Slack slash command and echo details back to the user, and an Amazon SNS trigger that sends CloudWatch alarm notifications to Slack.

1. In the **Select blueprint** section, enter and choose

Send CloudWatch alarm notifications via SNS

Then configure:

* **Function name**:

Slackfunction

* **Execution role**: *Use an existing role*
* **Existing role**: *lambda-role*

This role was pre-created for you as part of the lab setup.

1. Scroll down to the **SNS trigger** section, then configure:

* **SNS topic:** *slacknews*

1. In the **Environment variables** section (at the bottom), enter these values:

* **slackChannel:** Enter the channel name you defined earlier, which was:

#general

* **kmsEncryptedHookUrl:** Enter the Integration Webhook you copied to your text editor earlier in the lab

 You can normally encrypt the URL for added security, but it is not necessary for this lab.

1. Choose **Create function**.
2. In the **Function code** section below, select **lambda\_function.py** and delete all of the code in the code window.
3. Copy the Python code below and paste it into the Lambda code window.

import boto3

import json

import logging

import os

from base64 import b64decode

from urllib.request import Request, urlopen

from urllib.error import URLError, HTTPError

HOOK\_URL = os.environ['kmsEncryptedHookUrl']

SLACK\_CHANNEL = os.environ['slackChannel']

logger = logging.getLogger()

logger.setLevel(logging.INFO)

def lambda\_handler(event, context):

logger.info("Event: " + str(event))

message = event['Records'][0]['Sns']['Message']

logger.info("Message: " + str(message))

alarm\_name = message['AlarmName']

new\_state = message['NewStateValue']

reason = message['NewStateReason']

slack\_message = {

'channel': SLACK\_CHANNEL,

'text': "%s state is now %s: %s" % (alarm\_name, new\_state, reason)

}

req = Request(HOOK\_URL, json.dumps(slack\_message).encode('utf-8'))

try:

response = urlopen(req)

response.read()

logger.info("Message posted to %s", slack\_message['channel'])

except HTTPError as e:

logger.error("Request failed: %d %s", e.code, e.reason)

except URLError as e:

logger.error("Server connection failed: %s", e.reason)

Examine the code. It is doing the following:

* Retrieving information about the CloudWatch Alarm
* Sending a message to the Slack channel with the state of the alarm

1. Choose **Deploy** at the top of the page.

**Sample output:**

 Successfully updated the function Slackfunction.

1. In the **General configuration** section, choose **Edit**. Then configure:

* **Timeout:** *10 sec*

1. Choose **Save**

**Sample output:**

 Successfully updated the function Slackfunction.

 Congratulations! In this task you created and configured your Lambda function.

**Task 5: Test your Lambda function**

In this task, you test the Lambda function.

1. Choose the **Test** tab.
2. n the Test event section, configure:

* **Event name:**

SlackEvent

* **Template:** Search for and select:

SNS Topic Notification

1. Delete all of the code in the window.
2. Copy the JSON code snipit below into the code window.

{

"Records": [

{

"EventVersion": "1.0",

"EventSubscriptionArn": "arn:aws:sns:EXAMPLE",

"EventSource": "aws:sns",

"Sns": {

"SignatureVersion": "1",

"Timestamp": "1970-01-01T00:00:00.000Z",

"Signature": "EXAMPLE",

"SigningCertUrl": "EXAMPLE",

"MessageId": "95df01b4-ee98-5cb9-9903-4c221d41eb5e",

"Message": {

"AlarmName": "SlackAlarm",

"NewStateValue": "OK",

"NewStateReason":"Threshold Crossed: 1 datapoint (0.0) was not greater than or equal to the threshold (1.0)."

},

"MessageAttributes": {

"Test": {

"Type": "String",

"Value": "TestString"

},

"TestBinary": {

"Type": "Binary",

"Value": "TestBinary"

}

},

"Type": "Notification",

"UnsubscribeUrl": "EXAMPLE",

"TopicArn": "arn:aws:sns:EXAMPLE",

"Subject": "TestInvoke"

}

}

]

}

1. Look at the code that you just pasted into the window.

You can see that “AlarmName” in the JSON code is “SlackAlarm”. You will create this alarm in the next part of the lab.

1. Choose **Save**

**Sample output:**

 The test event SlackEvent was successfully saved.

1. Choose **Test**

You should see  **Execution results: succeeded (log)**

 Congratulations! You have successfully tested the Lambda function.

**Task 6: Create a CloudWatch Alarm**

In this task you will create a CloudWatch alarm to notify your SNS topic when the alarm is triggered.

Amazon CloudWatch is a monitoring service for AWS cloud resources and the applications you run on AWS. You can use Amazon CloudWatch to collect and track metrics, collect and monitor log files, set alarms, and automatically react to changes in your AWS resources. Amazon CloudWatch can monitor AWS resources such as Amazon EC2 instances, Amazon DynamoDB tables, and Amazon RDS DB instances, as well as custom metrics generated by your applications and services, and any log files your applications generate. You can use Amazon CloudWatch to gain system-wide visibility into resource utilization, application performance, and operational health. You can use these insights to react and keep your application running smoothly.

1. At the top of the page, in the unified search bar, search for and choose **CloudWatch**.
2. In left navigation pane, choose **Alarms** and then **All Alarms**.
3. Select **Create alarm**.
4. Choose **Select metric**
5. Under **Metrics**, select **Lambda**.
6. Select **Across All Functions**.
7. Select  Errors.
8. Choose **Select metric**
9. Under **Specify metric and conditions**, configure:

* **Statistic:** *Sum*
* **Period:** *1 Minute*

1. Scroll down to the **Conditions** section, configure:

* **Whenever Error is…** select  **Greater/Equal**
* **than:** enter 1

1. Choose **Next**
2. For **Notification**, configure:

* **Alarm state trigger:** choose  **OK**
* Choose  Select an existing SNS topic
* **Send a notification to**: select *slacknews*

1. Choose **Next**
2. **Alarm name** enter

SlackAlarm

1. **Alarm Description** enter

SlackAlarm

1. Choose **Next**
2. Choose **Create alarm**

**Sample output:**

 Successfully created alarm SlackAlarm.

 Congratulations! You have created a CloudWatch alarm to notify your SNS topic when the alarm is triggered.

**Task 7: Test your chat bot**

In this task, you test the Lambda funtion and validate a message is sent through the chat bot to your slack account.

1. At the top of the page, in the unified search bar, search for and choose **Lambda**.
2. Choose **Slackfunction**.
3. Choose **Test**
4. View the log messages.

* An SNS notification has been sent to your email address and your chat bot has posted the notification in your Slack Channel.

1. Return to the web browser tab/window with Slack. Load

https://<your-team-domain>.slack.com/messages

 then select your channel to see the notifications.

CONGRATULATIONS!

You have created your very own Slack bot using AWS Lambda. You can check your Slack feed on your mobile device to see the messages there as well.

**If you have more time…**

* Return to the Lambda Function Test window
* Customize the Test SNS message (JSON format) and view your results in your feed and logs

**Conclusion**

Congratulations! You now have successfully:

* Created a Slack chat bot using a Lambda blueprint
* Configured the bot with a Slack webhook and successfully posted messages to a Slack channel
* Tested the chat bot and verified that it posts CloudWatch alarms to your Slack channel

**End lab**

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

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

**Additional Resources**

For more information about AWS Lambda and Amazon CloudWatch, see:

* [AWS Lambda](https://aws.amazon.com/lambda/)
* [Amazon CloudWatch](https://aws.amazon.com/cloudwatch/)

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

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