Technical guide

Moderating Image Content in Slack with Amazon Rekognition and Amazon AppFlow

Moderating Image Content in Slack with Amazon Rekognition and Amazon AppFlow: Technical guide

Table of Contents

Moderating Image Content in Slack with Amazon Rekognition and Amazon AppFlow

Initial publication date: **April 14 2021**

About this guide

This technical guide shows you how to use Amazon Rekognition and Amazon AppFlow to build a fully serverless content moderation pipeline for messages posted in a Slack channel. The content moderation strategy in this guide identifies images that violate sample chosen guidelines:

Images that contain themes of tobacco or alcohol.

Images that contain the following disallowed words:

- medical
- private

These guidelines can be configured to fit your requirements.

Overview

In the increasingly virtual workplace, ease of communication is important to ensure effective collaboration between employees. Chat is taking over email as the preferred form of communication in many organizations. Employers rely on team tools such as Slack to get work done. Slack is growing at 67% year-over-year in the large enterprise segment.

Ensuring all aspects of the virtual work environment are inclusive and safe is a priority for many organizations. Sharing images can be a powerful way to effectively convey concepts and thoughts. There are many popular ways to analyze text, but images present a different challenge. Organizations need a way to detect and react to posted images that violate company guidelines.

Amazon Rekognition content moderation is a deep learning-based service that can detect inappropriate, unwanted, or offensive images and videos, making it easier to find and remove such content at scale. Amazon Rekognition provides a detailed taxonomy of moderation categories, such as Explicit Nudity, Suggestive, Violence, and Visually Disturbing. You can now detect six new categories: Drugs, Tobacco, Alcohol, Gambling, Rude Gestures, and Hate Symbols.

Amazon AppFlow is a fully managed integration service that enables you to securely transfer data between Software-as-a-Service (SaaS) applications like Salesforce, Marketo, Slack, and ServiceNow, and AWS services like Amazon Simple Storage Service (Amazon S3) and Amazon Redshift, in just a few clicks. This solution leverages Amazon AppFlow to capture the content posted in Slack channels for analysis using Amazon Rekognition.

Before you begin

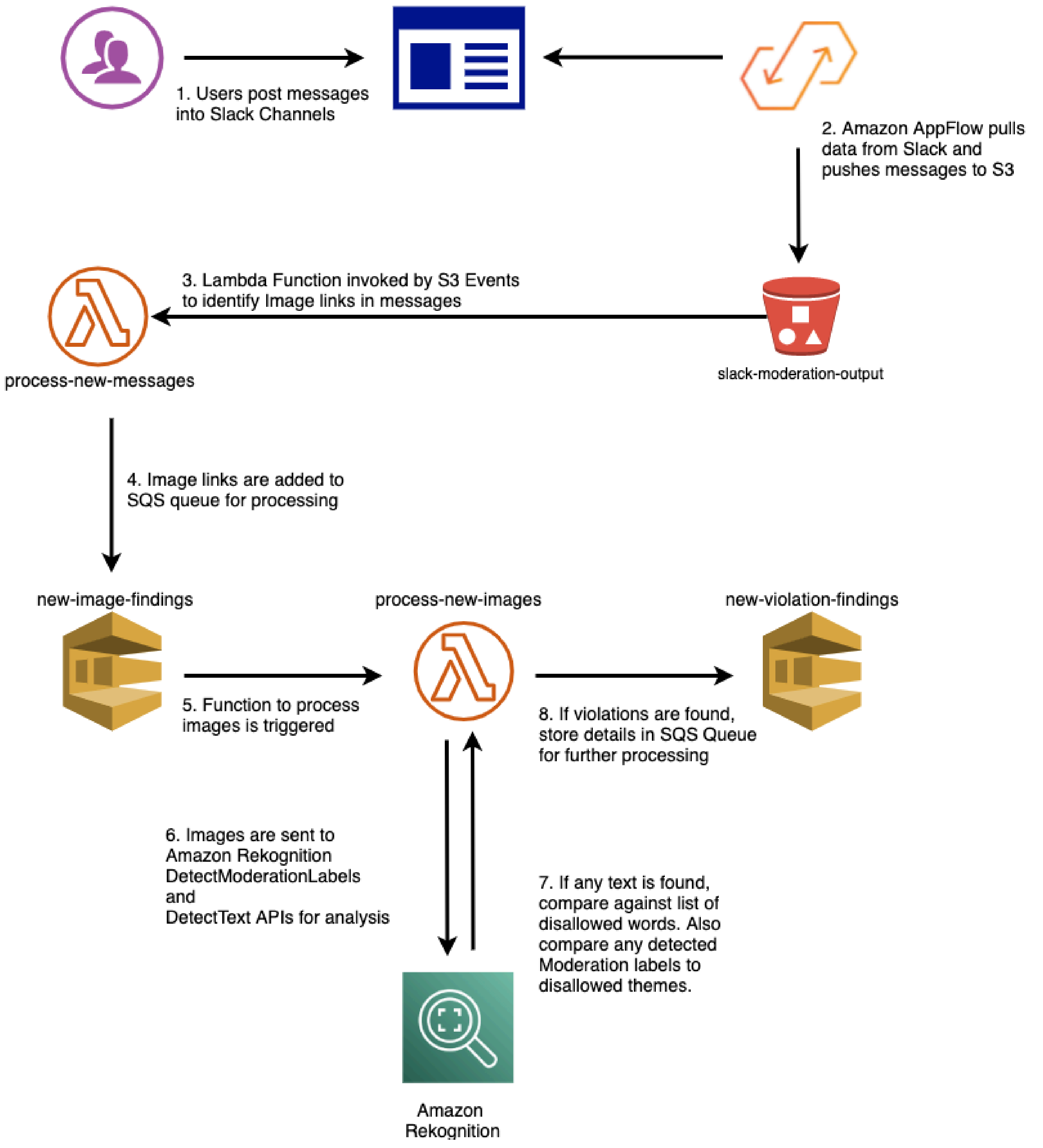
For this solution, you should have the following:

- An AWS account
- A Slack workspace with administrative access and a Slack channel to monitor. If you don't have one, see Create a Slack workspace.
- A S3 bucket. If you don't have one, see Creating a bucket.

This solution does not require any prior machine learning (ML) expertise, or development of your own custom ML models.

Architecture overview

This solution uses serverless technologies and managed services to be scalable and cost-effective. By using an event-driven architecture that incorporates AWS Lambda and Amazon Simple Queue Service (SQS), you can decouple image detection and image processing without provisioning or managing any servers.


1. Users post messages into Slack Channels
2. Amazon AppFlow pulls data from Slack and pushes messages to S3
3. Lambda Function invoked by S3 Events to identify Image links in messages
slack-moderation-output
process-new-messages
4. Image links are added to SQS queue for processing
new-image-findings
process-new-images
new-violation-findings
5. Function to process images is triggered
8. If violations are found, store details in SQS Queue for further processing
6. Images are sent to Amazon Rekognition DetectModerationLabels and DetectText APIs for analysis
7. If any text is found, compare against list of disallowed words. Also compare any detected Moderation labels to disallowed themes.
Amazon Rekognition

Monitoring image content solution architecture diagram

Walkthrough

The sequence of steps is as follows:

1. Users post messages in Slack channels that may contain text or links to images.
2. Amazon AppFlow is used to capture all messages and store them in Amazon S3.
3. As new message content is stored in S3, an object notification invokes an AWS Lambda function (`process-new-messages`) to identify image links in the messages.

> **Note**
>
> Content moderation can be done on the text messages in parallel with the path shown for inspecting images. However, this example focuses on inspecting the images only.

4. If there is a link to an image found in the messages, it is added to an Amazon SQS Queue (`new-image-findings`) for further processing.
5. As items are added to the SQS Queue, another Lambda function (`process-new-images`) is triggered to process these items.
6. This second Lambda function calls Amazon Rekognition twice for analysis. First, Amazon Rekognition determines if there is any textual content found in the image. Second, it detects any inappropriate or offensive themes.
7. If any text is returned by Amazon Rekognition, it is compared against a list of disallowed words. Any detected moderation labels in the image are compared to disallowed themes.
8. If violations are found, the details are stored in a SQS Queue (`new-violation-findings`) for further processing (for example, alerting a moderator).

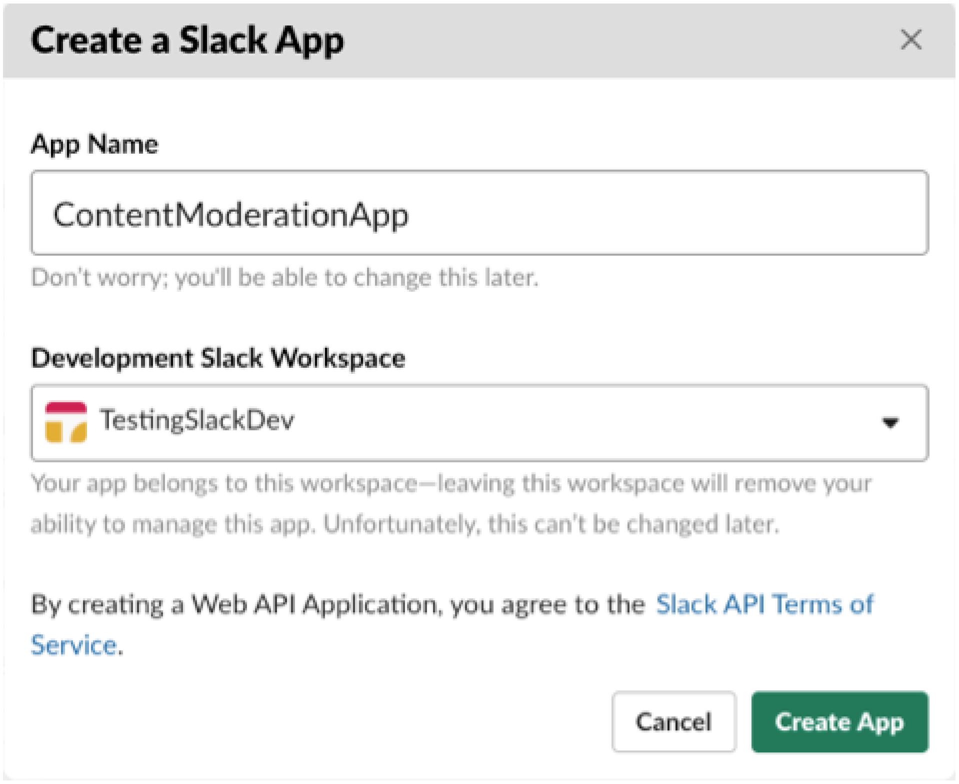
Creating the Slack App in your Slack workspace

Before you can configure the Amazon AppFlow Flow in AWS, you need to create an App in your Slack workspace, and obtain the Client ID and Client Secret credentials that will be required for the AppFlow Flow.

To configure and install the Slack app:

1. Navigate to api.slack.com and log in to your Workspace.

2. Choose **Your Apps** in the navigation menu at the top or the screen.
3. Choose the **Create New App** button.
4. In the **Create a Slack App** dialog, enter an **App Name** and choose your Workspace from the dropdown list.



Create a Slack App

App Name

ContentModerationApp

Don't worry; you'll be able to change this later.

Development Slack Workspace

TestingSlackDev

Your app belongs to this workspace—leaving this workspace will remove your ability to manage this app. Unfortunately, this can't be changed later.

By creating a Web API Application, you agree to the Slack API Terms of Service.

***Create a Slack App** dialog*

5. After the app is created, within the **Basic Information** page for the app, scroll down to the **App Credentials** section and make note of the **Client ID** and **Client Secret**. These credentials will be used configure the Amazon AppFlow Flow configuration in the next step.
6. Navigate to the **Oauth & Permissions** page from the left menu, and in the **Redirect URLs** section, choose the **Add New Redirect URL** button.
7. Paste in the following value: https://console.aws.amazon.com/appflow/oauth.
8. Choose **Add**.
9. Choose the **Add New Redirect URL** button again, and paste in the following value: https://us-east-1.console.aws.amazon.com/appflow/oauth.
10. Choose **Add**.
11. Choose the **Save URLs** button directly below the two URLs you added.
12. Navigate to the **OAuth & Permissions** page, and scroll down to the **User Token Scopes** section.

13.Choose the **Add an OAuth Scope** button, and add in the user token scopes mentioned in the documentation in the following link, one at a time: https://docs.aws.amazon.com/appflow/latest/userguide/slack.html

14.Scroll to the top of the page and choose **Install into Workspace.**

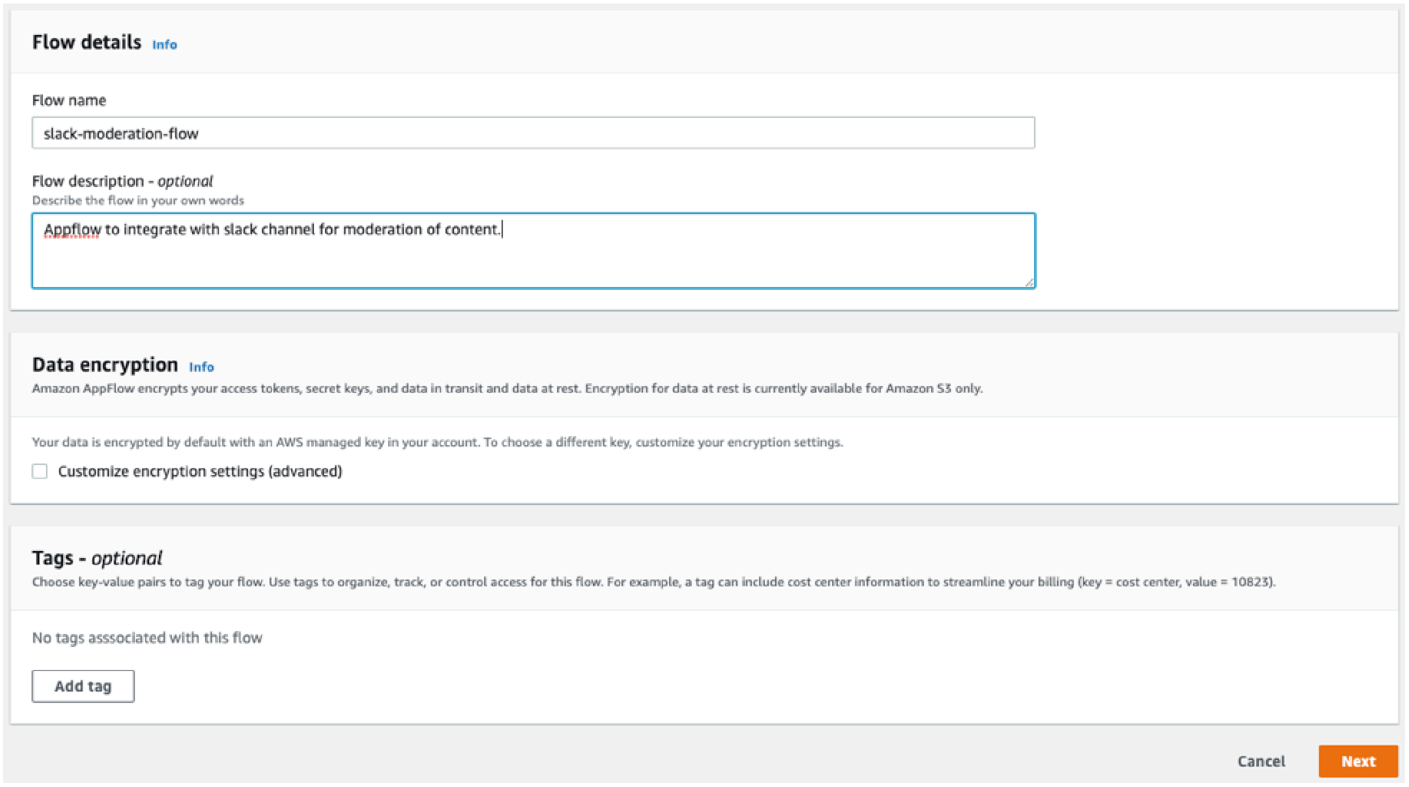
15.A confirmation dialog appears, requesting permissions. Choose **Allow.**

Create the Amazon AppFlow Integration with your Slack workspace

After the Slack App has been created, follow these steps to configure the Amazon AppFlow integration.

To configure the Amazon AppFlow Integration:

1. Navigate to the Amazon AppFlow console and choose **Create flow**.
2. In Step 1 (**Specify flow details**) of the creation process, enter a **Flow name**, and optionally, a description. For the purposes of this demo, leave the **Data encryption** setting as it is. Optionally, enter any tags you'd like for the flow.
3. Choose **Next**.



Flow details Info

Flow name

slack-moderation-flow

Flow description - *optional*
Describe the flow in your own words

Appflow to integrate with slack channel for moderation of content.

Data encryption Info
Amazon AppFlow encrypts your access tokens, secret keys, and data in transit and data at rest. Encryption for data at rest is currently available for Amazon S3 only.

Your data is encrypted by default with an AWS managed key in your account. To choose a different key, customize your encryption settings.

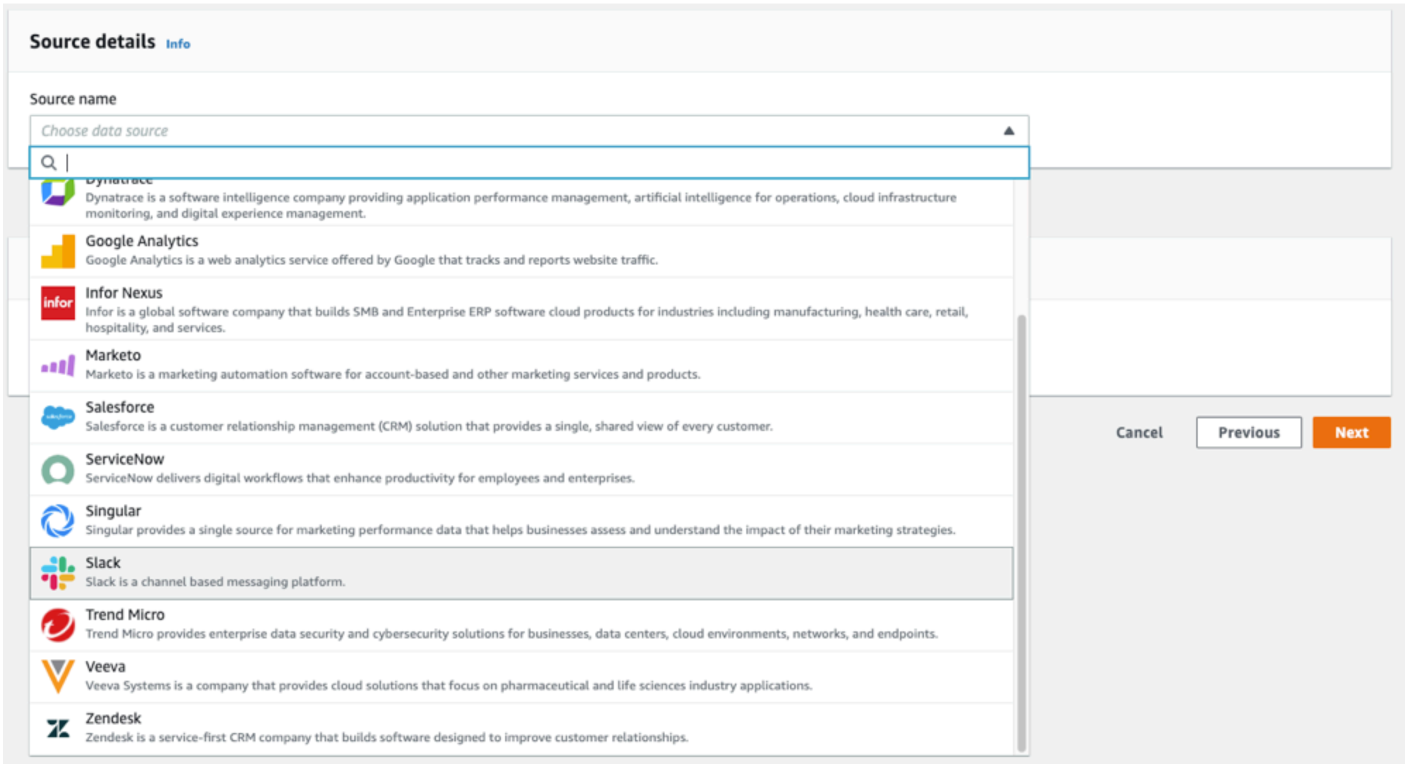
☐ Customize encryption settings (advanced)

Tags - *optional*
Choose key-value pairs to tag your flow. Use tags to organize, track, or control access for this flow. For example, a tag can include cost center information to streamline your billing (key = cost center, value = 10823).

No tags asssociated with this flow

*The **Flow details** page*

4. In Step 2 (**Configure flow**), choose the **Source name** dropdown list and choose **Slack** from the list of options:


Source details Info

Source name

Choose data source

Dynatrace
Dynatrace is a software intelligence company providing application performance management, artificial intelligence for operations, cloud infrastructure monitoring, and digital experience management.

Google Analytics
Google Analytics is a web analytics service offered by Google that tracks and reports website traffic.

Infor Nexus
Infor is a global software company that builds SMB and Enterprise ERP software cloud products for industries including manufacturing, health care, retail, hospitality, and services.

Marketo
Marketo is a marketing automation software for account-based and other marketing services and products.

Salesforce
Salesforce is a customer relationship management (CRM) solution that provides a single, shared view of every customer.

ServiceNow
ServiceNow delivers digital workflows that enhance productivity for employees and enterprises.

Singular
Singular provides a single source for marketing performance data that helps businesses assess and understand the impact of their marketing strategies.

Slack
Slack is a channel based messaging platform.

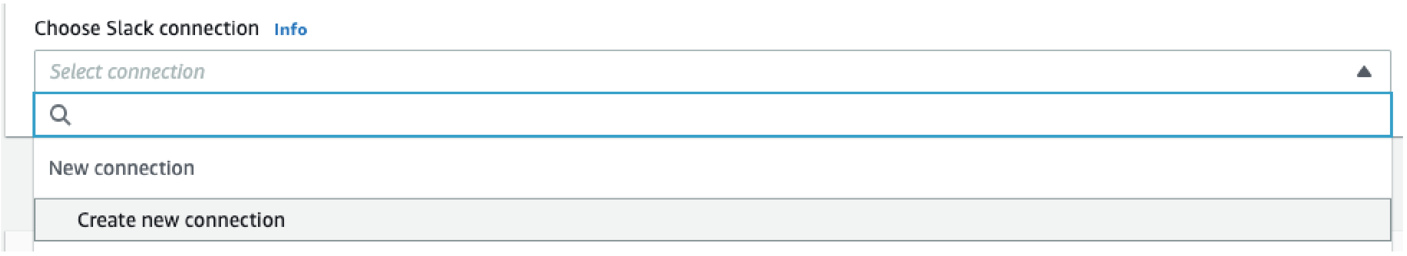
Trend Micro
Trend Micro provides enterprise data security and cybersecurity solutions for businesses, data centers, cloud environments, networks, and endpoints.

Veeva
Veeva Systems is a company that provides cloud solutions that focus on pharmaceutical and life sciences industry applications.

Zendesk
Zendesk is a service-first CRM company that builds software designed to improve customer relationships.

*The **Source name** dropdown list*

5. A **Choose Slack connection** dropdown list appears. From this list, choose **Create new connection**:


Choose Slack connection Info

Select connection

New connection

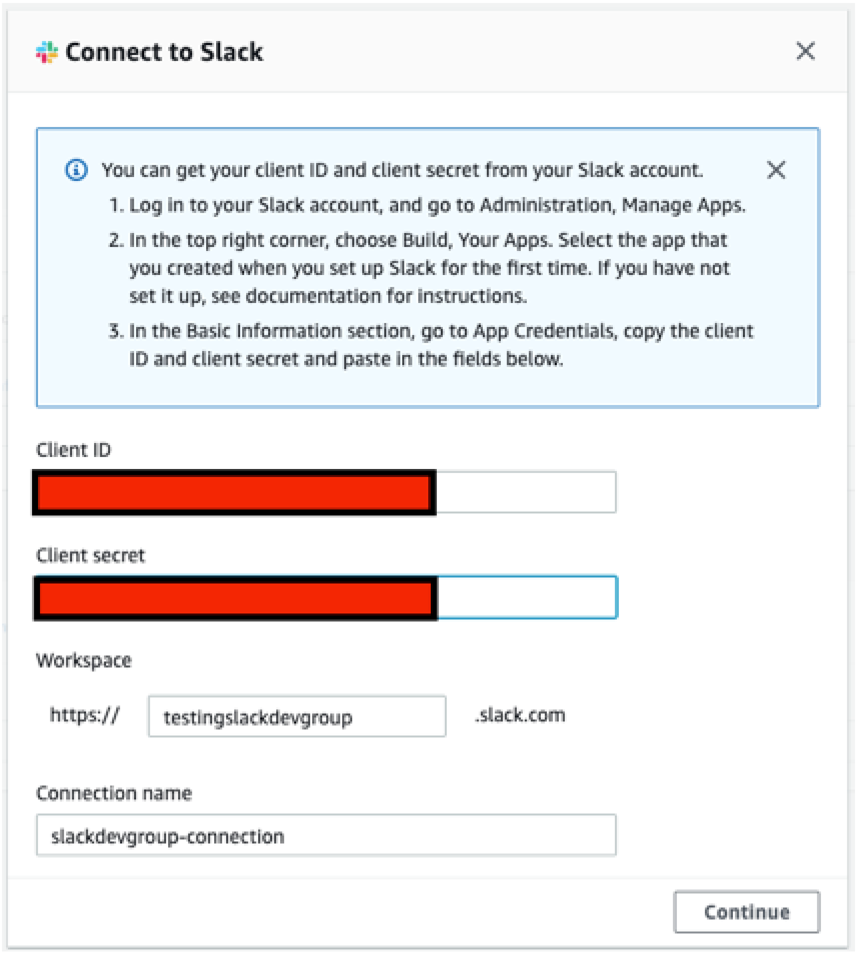
Create new connection

*Choose **Create new connection***

6. Enter your Slack workspace address (for example, `testingslackdevgroup.slack.com`), and the **Client ID** and **Client Secret** generated when you created the Slack App.

7. Give your connection a name on the **Connect to Slack** popup window.

8. Choose **Continue**.



Connect to Slack

You can get your client ID and client secret from your Slack account.

1. Log in to your Slack account, and go to Administration, Manage Apps.
2. In the top right corner, choose Build, Your Apps. Select the app that you created when you set up Slack for the first time. If you have not set it up, see documentation for instructions.
3. In the Basic Information section, go to App Credentials, copy the client ID and client secret and paste in the fields below.

Client ID

[redacted]

Client secret

[redacted]

Workspace

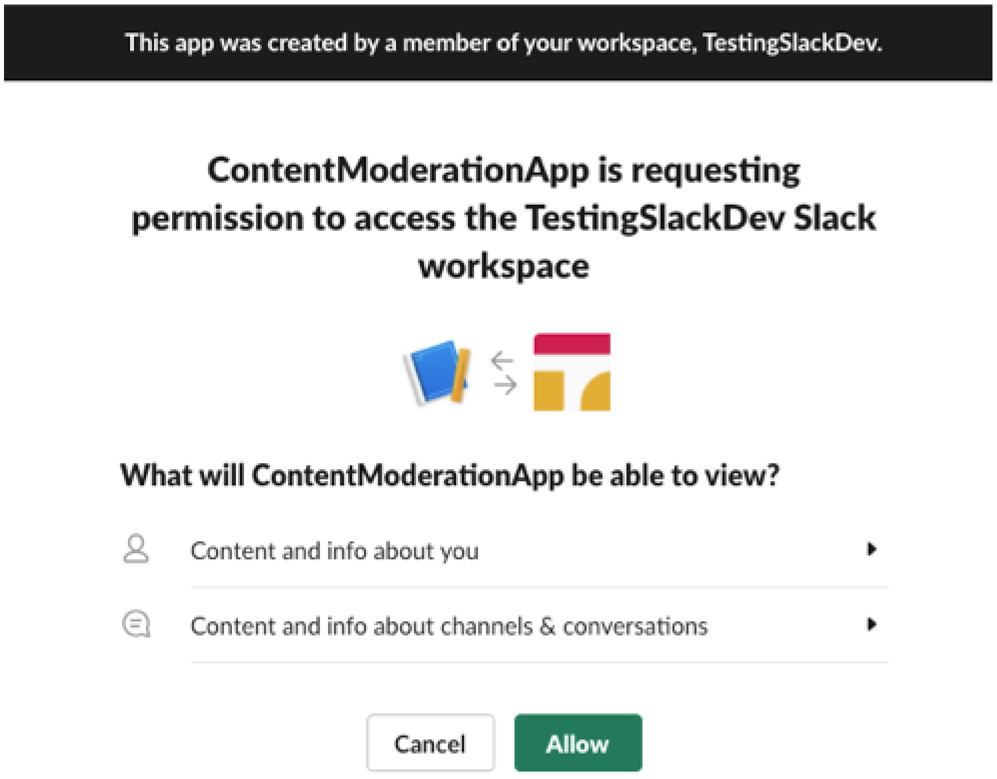
https:// testingslackdevgroup .slack.com

Connection name

slackdevgroup-connection

*The **Connect to Slack** window*

9. A window pops up with a confirmation prompt to allow permissions. Choose **Allow**.



This app was created by a member of your workspace, TestingSlackDev.

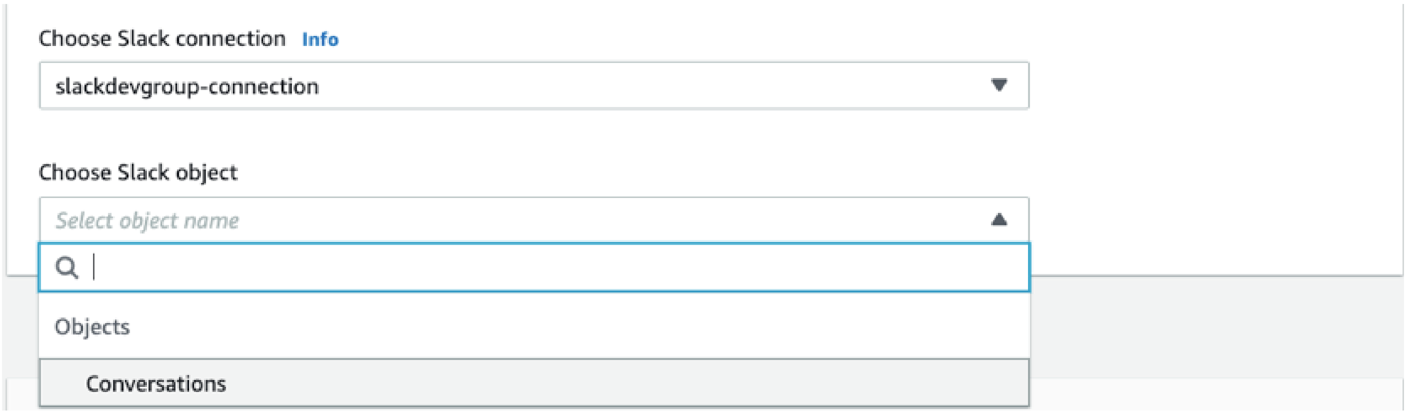
ContentModerationApp is requesting permission to access the TestingSlackDev Slack workspace

What will ContentModerationApp be able to view?

Content and info about you
Content and info about channels & conversations
Cancel
Allow

The confirmation prompt

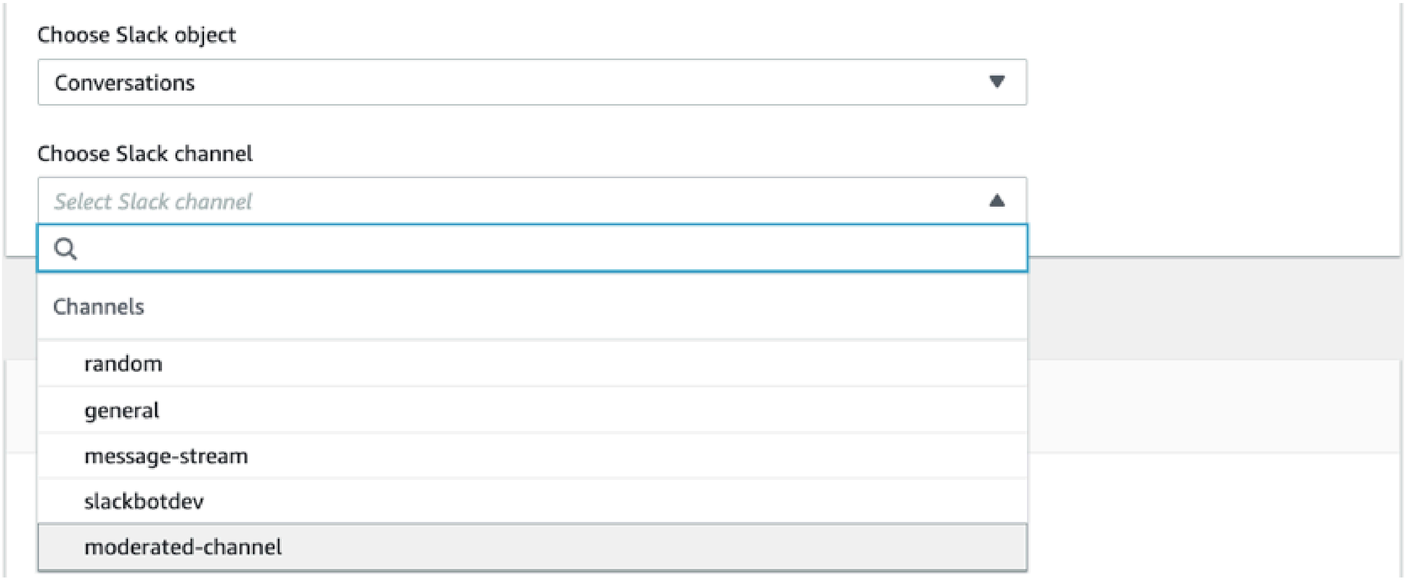
10. Your new connection is configured and displayed in the **Choose Slack connection** dropdown list, and a new **Choose Slack object** dropdown list appears directly below it. Choose **Conversations**.



Choose Slack connection: slackdevgroup-connection

*Select **Conversations** from the dropdown list*

11. A new dropdown appears directly below **Choose Slack channel**. From this list, choose the Slack channel that you would like to perform content moderation on.



Choose Slack object

Conversations

Choose Slack channel

Select Slack channel

Channels

random

general

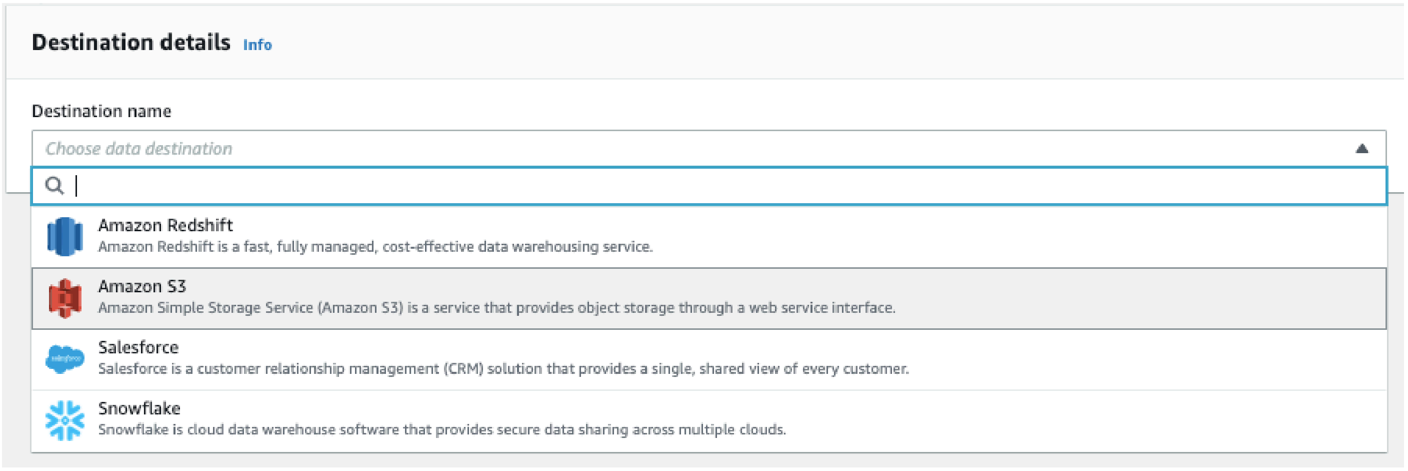
message-stream

slackbotdev

moderated-channel

Choose a Slack channel to moderate

12.With the Slack workspace connected, and the channel for moderation selected, you can move on to configuring the **Destination details**. First, choose **Amazon S3** from the **Destination name** dropdown list.



Destination details Info

Destination name

Choose data destination

Amazon Redshift
Amazon Redshift is a fast, fully managed, cost-effective data warehousing service.

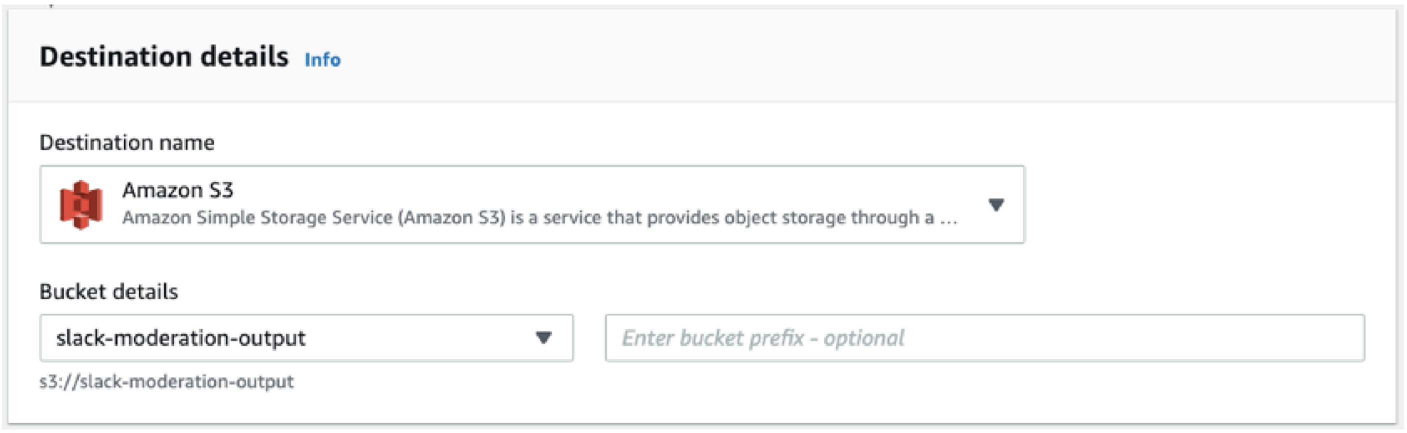
Amazon S3
Amazon Simple Storage Service (Amazon S3) is a service that provides object storage through a web service interface.

Salesforce
Salesforce is a customer relationship management (CRM) solution that provides a single, shared view of every customer.

Snowflake
Snowflake is cloud data warehouse software that provides secure data sharing across multiple clouds.

*Select **Amazon S3** from the **Destination name** dropdown list*

13.A **Bucket details** dropdown list appears. Choose the S3 bucket you would like to use, and leave the prefix empty.



Destination name

Amazon S3
Amazon Simple Storage Service (Amazon S3) is a service that provides object storage through a ...

Bucket details

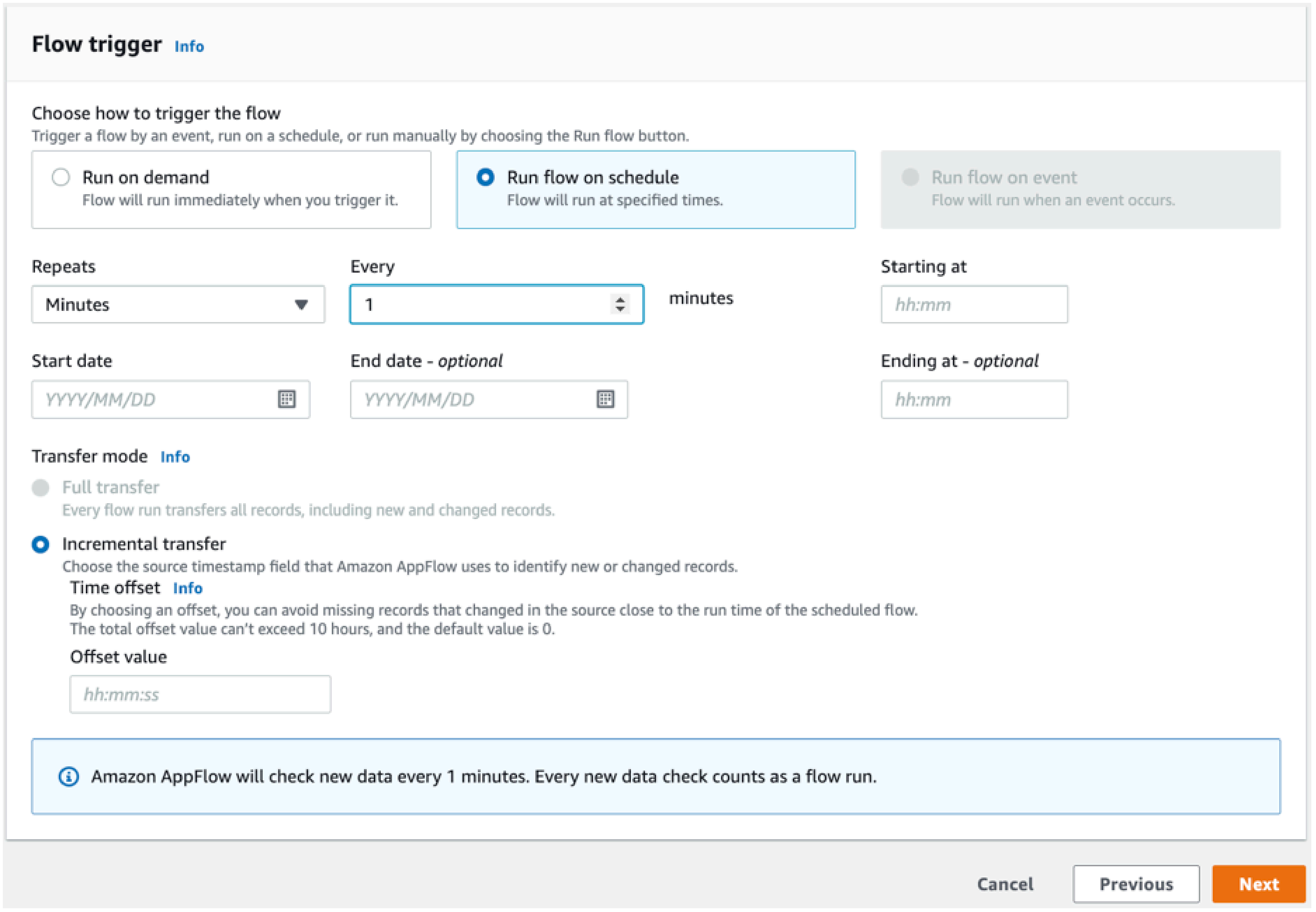
slack-moderation-output

Enter bucket prefix - optional

s3://slack-moderation-output

Choose the S3 bucket you want to use

14. A new section titled **Flow trigger** appears with two options: **Run on demand** or **Run flow on schedule**. Choose the second option, and configure the schedule to run every one (1) minute.
15. When you choose this option, the **Incremental Transfer** option is auto-selected. Enter a value for **Starting at** and **Start date**.
16. Choose **Next**.



Flow trigger Info

Choose how to trigger the flow
Trigger a flow by an event, run on a schedule, or run manually by choosing the Run flow button.

- ○ Run on demand — Flow will run immediately when you trigger it.
- ● Run flow on schedule — Flow will run at specified times.
- ○ Run flow on event — Flow will run when an event occurs.

Repeats	Every		Starting at
Minutes	1	minutes	hh:mm

Start date	End date - *optional*	Ending at - *optional*
YYYY/MM/DD	YYYY/MM/DD	hh:mm

Transfer mode Info

- ○ Full transfer — Every flow run transfers all records, including new and changed records.
- ● Incremental transfer — Choose the source timestamp field that Amazon AppFlow uses to identify new or changed records.

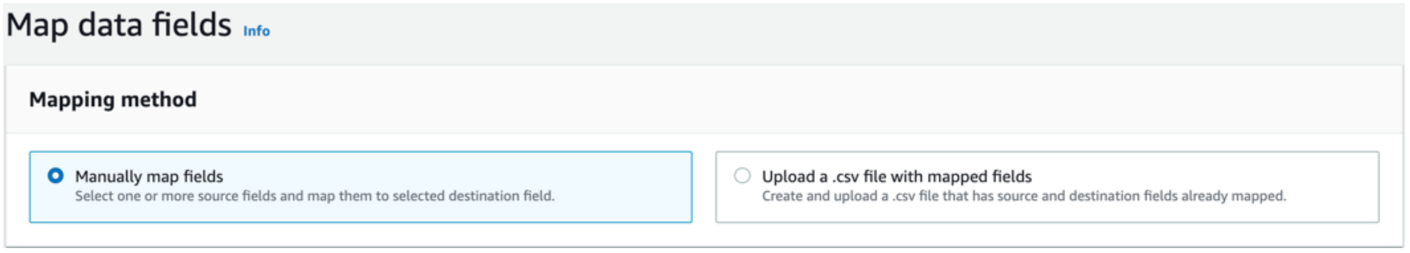
Time offset Info
By choosing an offset, you can avoid missing records that changed in the source close to the run time of the scheduled flow.
The total offset value can't exceed 10 hours, and the default value is 0.

Offset value: hh:mm:ss

Amazon AppFlow will check new data every 1 minutes. Every new data check counts as a flow run.

Flow trigger options

17. In Step 3 (**Map data fields**), you have the option to perform transformations on the data fields. Choose **Manually map fields**.



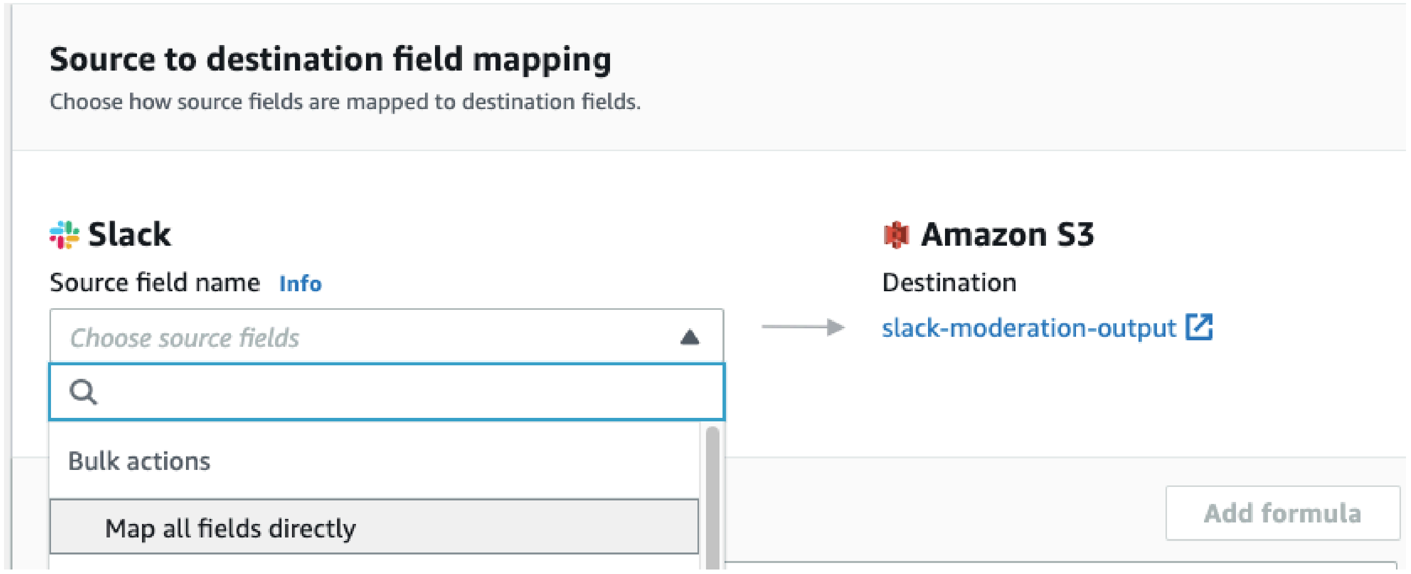
Map data fields Info

Mapping method

- ● Manually map fields — Select one or more source fields and map them to selected destination field.
- ○ Upload a .csv file with mapped fields — Create and upload a .csv file that has source and destination fields already mapped.

*The **Map data fields** options*

18. From the **Source field name** dropdown, select **Map all fields directly**. This creates a mapping of all the fields without any transformations.


Source to destination field mapping
Choose how source fields are mapped to destination fields.

*Select **Map all fields directly***

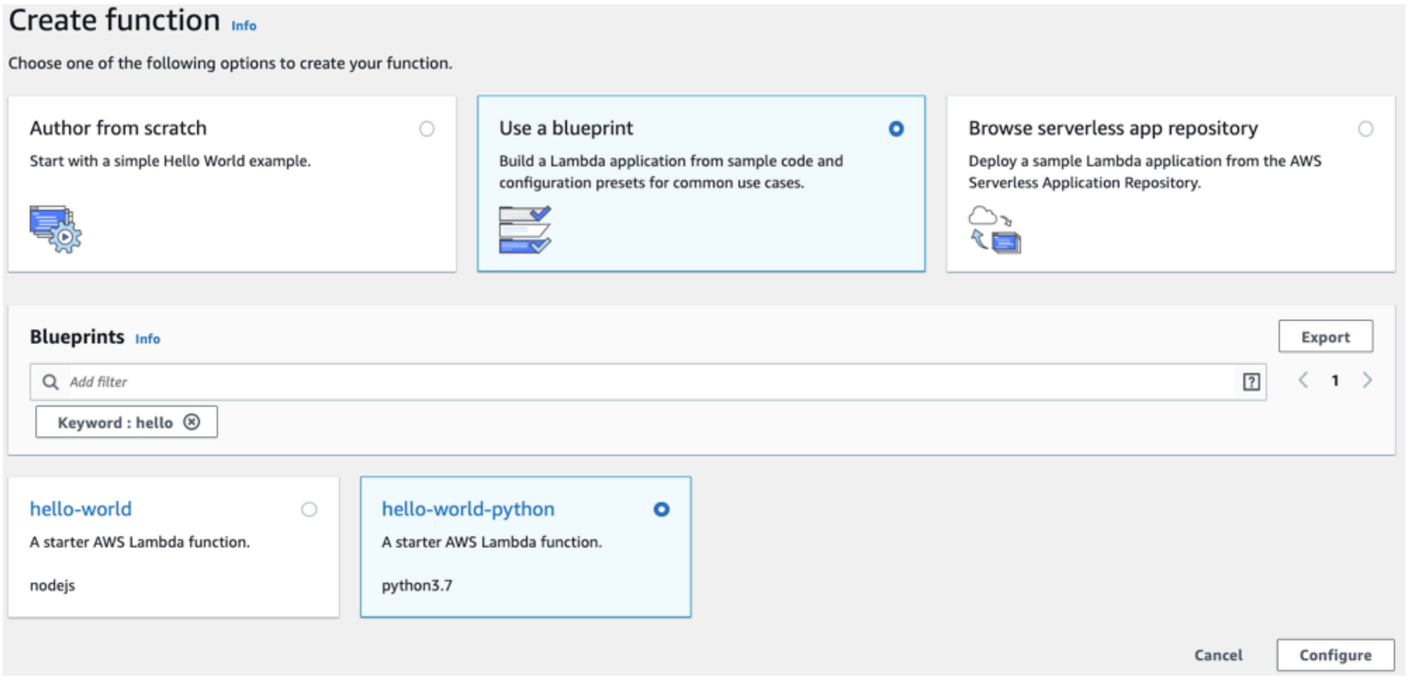
19. Choose **Next**.
20. In Step 4 (**Add filters**), you have the option to perform filtering on the data. Do not add any filters here, simply choose **Next** to continue.
21. On the **Review and Create** screen, a summary of all your selections from previous steps is shown. Review these for accuracy, then scroll to the bottom of the page and choose **Create flow**.
22. After the flow has been created, on the following screen, choose the **Activate flow** button.

Create a Lambda function to process files in the S3 bucket that contain new Slack messages

Now that you have the Amazon AppFlow integration complete, your flow will download the latest messages from the Slack channel and store them in your S3 bucket. The next step is to create the Lambda function that will be invoked to process these new files. The Lambda function finds any messages in the files that have references to an external image URL, and stores these URLs in an SQS queue. While you could directly inspect those images within this same function, the best practice is to decouple these two operations by using a different Lambda function to perform the inspection. This makes your architecture more fault tolerant and resilient.

To create the Lambda function:

1. Because the Lambda function needs to store the image URLs it finds into a new SQS queue, first create that queue by following the steps outlined in Getting started with Amazon SQS. Name this queue `new-image-findings`.
2. Navigate to the Lambda console. Choose **Create Function** and choose the option to **Use a blueprint**, then provide a filter called `hello`. This displays the `hello-world-python` blueprint in the results at the bottom.
3. Choose the **Configure** button.


Create function Info

Choose one of the following options to create your function.

Author from scratch
Start with a simple Hello World example.

Use a blueprint [✓]
Build a Lambda application from sample code and configuration presets for common use cases.

Browse serverless app repository
Deploy a sample Lambda application from the AWS Serverless Application Repository.

Blueprints Info

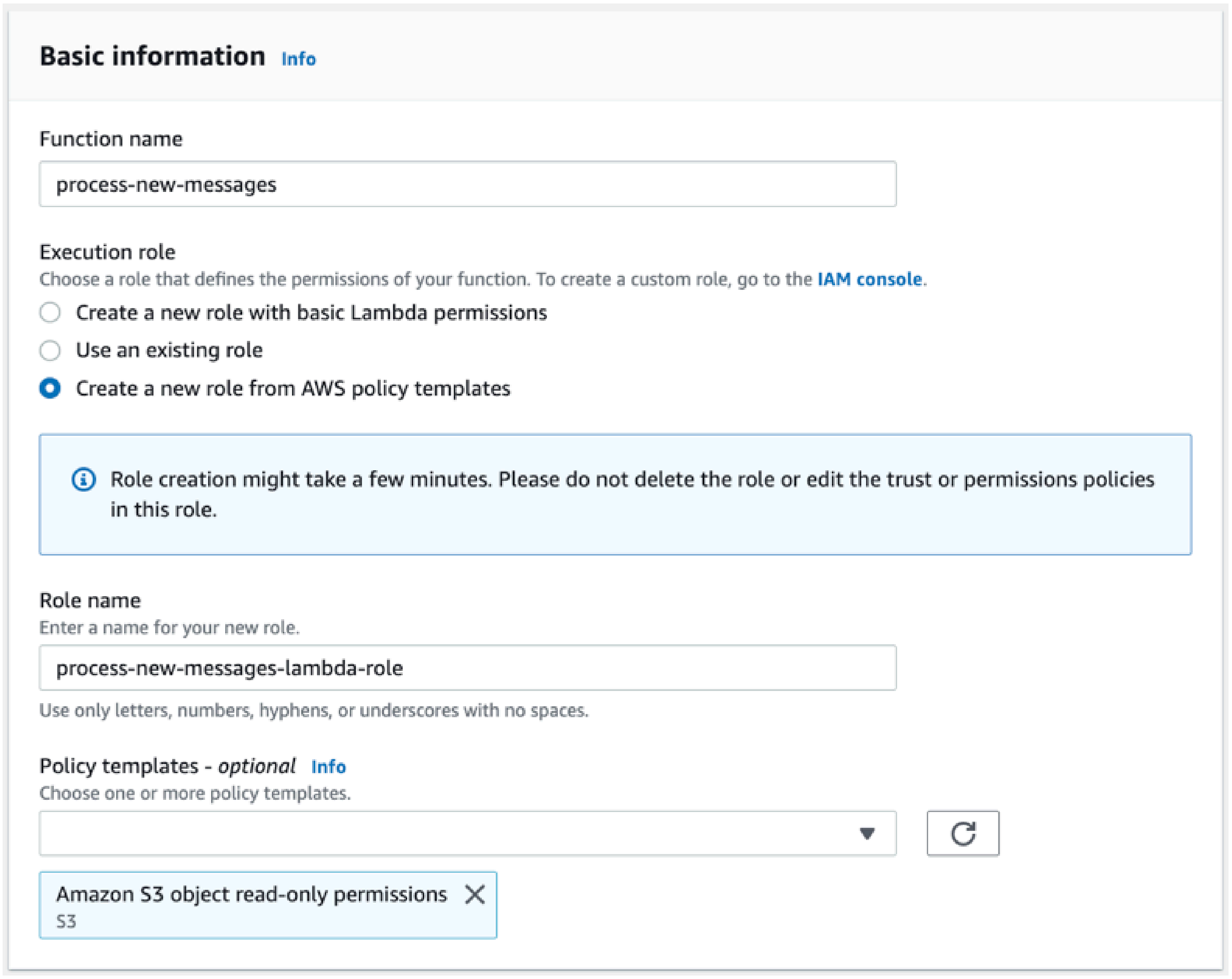
Keyword : hello

hello-world
A starter AWS Lambda function.
nodejs

hello-world-python [✓]
A starter AWS Lambda function.
python3.7

*The **Create function** screen on the Lambda console*

4. On the next screen, provide a name for your new function called `process-new-messages`, and create a new IAM role called `process-new-messages-lambda-role` using the available “Amazon S3 object read-only permissions” template. This role will need to be customized in a later step.



Basic information Info

Function name

process-new-messages

Execution role

Choose a role that defines the permissions of your function. To create a custom role, go to the IAM console.

- ○ Create a new role with basic Lambda permissions
- ○ Use an existing role
- [✓ Create a new role from AWS policy templates]

Role creation might take a few minutes. Please do not delete the role or edit the trust or permissions policies in this role.

Role name

Enter a name for your new role.

process-new-messages-lambda-role

Use only letters, numbers, hyphens, or underscores with no spaces.

Policy templates - *optional* Info

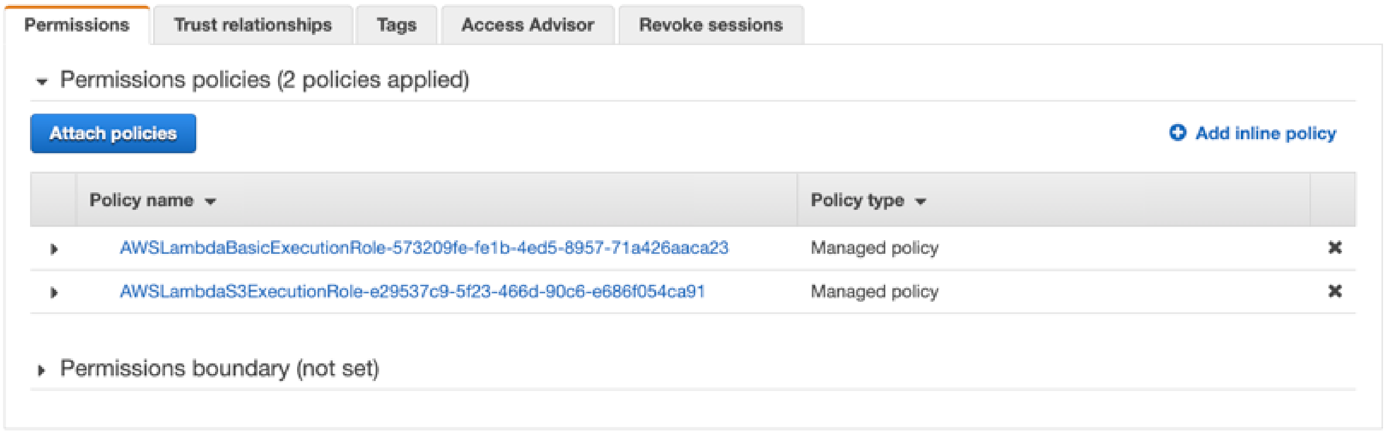
Choose one or more policy templates.

Amazon S3 object read-only permissions
S3

*The **Basic information** screen*

5. After the function has been created, choose the **Permissions** tab. The role you created (`process-new-messages-lambda-role`) is displayed.

6. Choose the role name to open a second window where you can view the two policies applied to this role.



Policy name	Policy type
AWSLambdaBasicExecutionRole-573209fe-fe1b-4ed5-8957-71a426aaca23	Managed policy
AWSLambdaS3ExecutionRole-e29537c9-5f23-466d-90c6-e686f054ca91	Managed policy

Permissions policies

7. Expand each policy to view the permissions details. The policy named `AWSLambdaBasicExecutionRole-*` grants the necessary permissions for the function to log information in CloudWatch. The policy named `AWSLambdaS3ExecutionRole-*` provides S3 permissions and needs to be modified. To modify the policy, choose **Edit Policy** and switch to the JSON view to customize this policy. The final permissions statement should appear as follows:

```
"Statement": [
        {
            "Action": [
                "s3:GetObject*",
                "s3:GetBucket*",
                "s3:List*"
            ],
            "Resource": [
                "arn:aws:s3:::slack-moderation-output",
                "arn:aws:s3:::slack-moderation-output/*"
            ],
            "Effect": "Allow"
        }
    ]
```

The preceding statement follows the principle of least privilege, and limits the permissions of this Lambda function to only the bucket you created for this exercise. Save the change you've made to this policy.

For this function to write messages to the `new-image-findings` SQS queue, an additional minimally scoped IAM policy needs to be added to this role.

To add the IAM policy:

1. Choose **Add inline policy** and switch to the JSON view to create the following permissions. Note that the following Resource element needs to be updated with the correct Amazon Resource Name (ARN) for the `new-image-findings` SQS queue which contains your actual account number.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "sqs:SendMessage",
                "sqs:GetQueueAttributes",
                "sqs:GetQueueUrl"
            ],
            "Resource": "arn:aws:sqs:us-east-1:111111111111:new-image-findings",
            "Effect": "Allow"
        }
    ]
}
```

2. Choose **Review policy**, then enter a name for this policy and choose **Create policy**.
3. With the permissions properly configured, switch back to the **Configuration** tab in the Lambda function window, and paste the following code into the **Function code** section:

```
import boto3
from urllib.parse import unquote_plus
import json

s3_client = boto3.client('s3')
s3 = boto3.resource('s3')
sqs = boto3.client('sqs')

def sendToSqS(attributes, queueurl):

    sqs = boto3.client('sqs')
    sqs.send_message(
        QueueUrl=queueurl,
        MessageBody='Image to Check',
        MessageAttributes={
            "url": {
                "StringValue": attributes["image_url"],
                "DataType": 'String'
            },
            "slack_msg_id": {
                "StringValue": attributes["client_msg_id"],
                "DataType": 'String'
            }
        }
    )

def lambda_handler(event, context):

    image_processing_queueurl = "https://queue.amazonaws.com/111111111111/new-image-
findings”

    for record in event['Records']:
        bucket = record['s3']['bucket']['name']
        key = unquote_plus(record['s3']['object']['key'])

        file_lines = s3.Object(bucket, key).get()
['Body'].read().decode('utf-8').splitlines()

        attachment_list = []

        for line in file_lines:
            if line:  # Check for blank lines
                jsonline = json.loads(line)
                if "attachments" in jsonline.keys():  # Check for lines with
 attachements
                    for attachment in jsonline["attachments"]:
                        if "image_url" in attachment.keys():
                            if "client_msg_id" in jsonline.keys():
                                thisdict = {
                                    "image_url": attachment["image_url"],
                                    "client_msg_id": jsonline["client_msg_id"]
                                }
                                attachment_list.append(thisdict.copy())
                            else:
                                thisdict = {
                                    "image_url": attachment["image_url"],
                                    "client_msg_id": "None Found"
                                }
                                attachment_list.append(thisdict.copy())

            for item in attachment_list:
                sendToSqS(item, image_processing_queueurl)
```

4. After you have pasted the code, update the `image_processing_queueurl` variable in the function handler with the correct ARN for the `new-image-findings` SQS queue which contains your actual account number.
5. Choose **Deploy** to deploy the updated code.

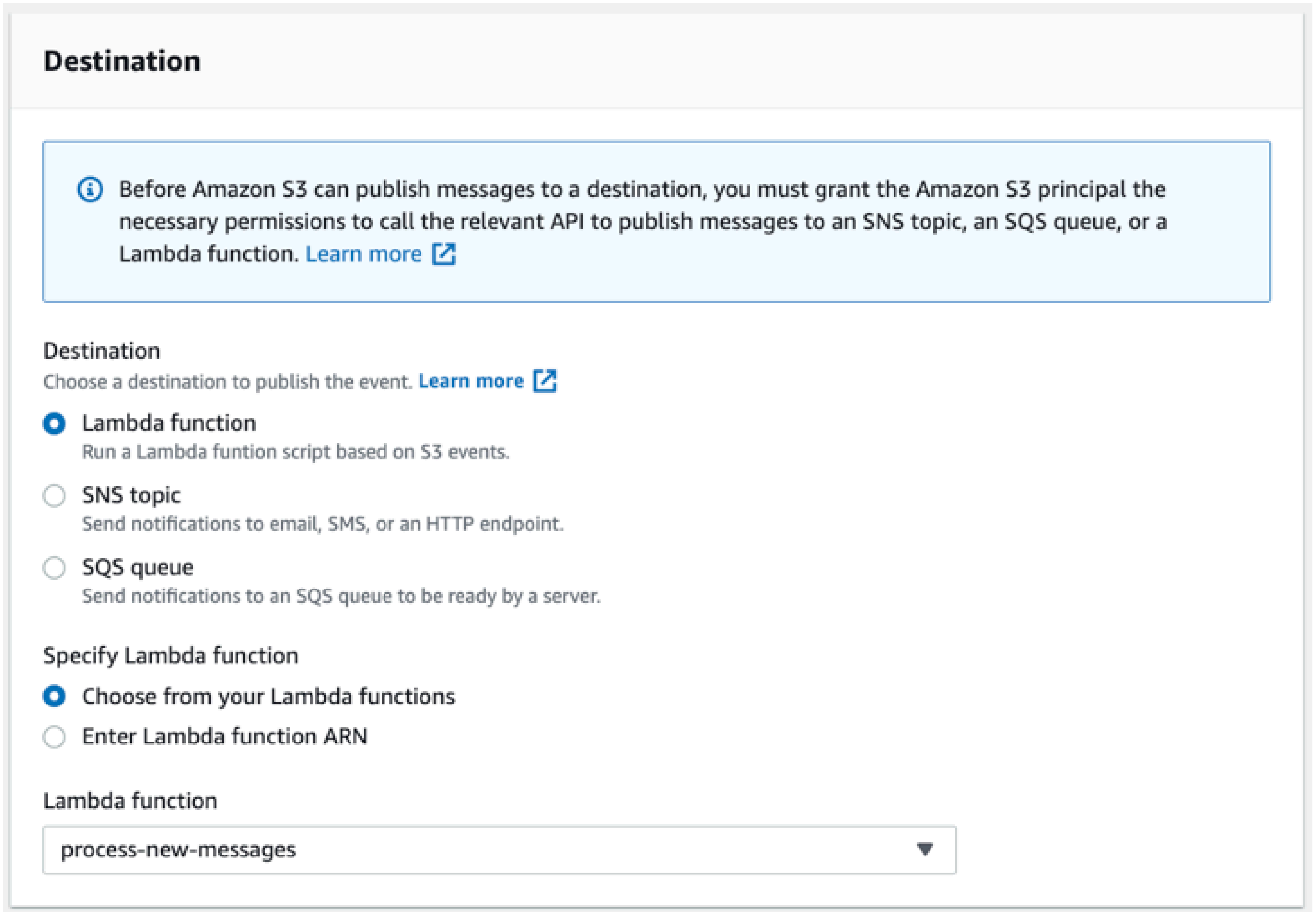
Configure the Lambda function to be invoked when new objects are added to your S3 bucket

With your Lambda function (`process-new-messages`) created, the next step is to configure bucket notifications on your S3 bucket, and subscribe this Lambda function to the notifications.

To create the S3 / Lambda event integration:

Configure event notifications on your S3 bucket by following the steps outlined in this User Guide.

- In Step 5 of the configuration, choose the **All object create events** option.
- In Step 6, choose your Lambda function named `process-new-messages`.



Destination

Before Amazon S3 can publish messages to a destination, you must grant the Amazon S3 principal the necessary permissions to call the relevant API to publish messages to an SNS topic, an SQS queue, or a Lambda function. Learn more

Destination

Choose a destination to publish the event. Learn more

- [✓ Lambda function] Run a Lambda funtion script based on S3 events.
- SNS topic Send notifications to email, SMS, or an HTTP endpoint.
- SQS queue Send notifications to an SQS queue to be ready by a server.

Specify Lambda function

- [✓ Choose from your Lambda functions]
- Enter Lambda function ARN

Lambda function: process-new-messages

*The **Destination** screen*

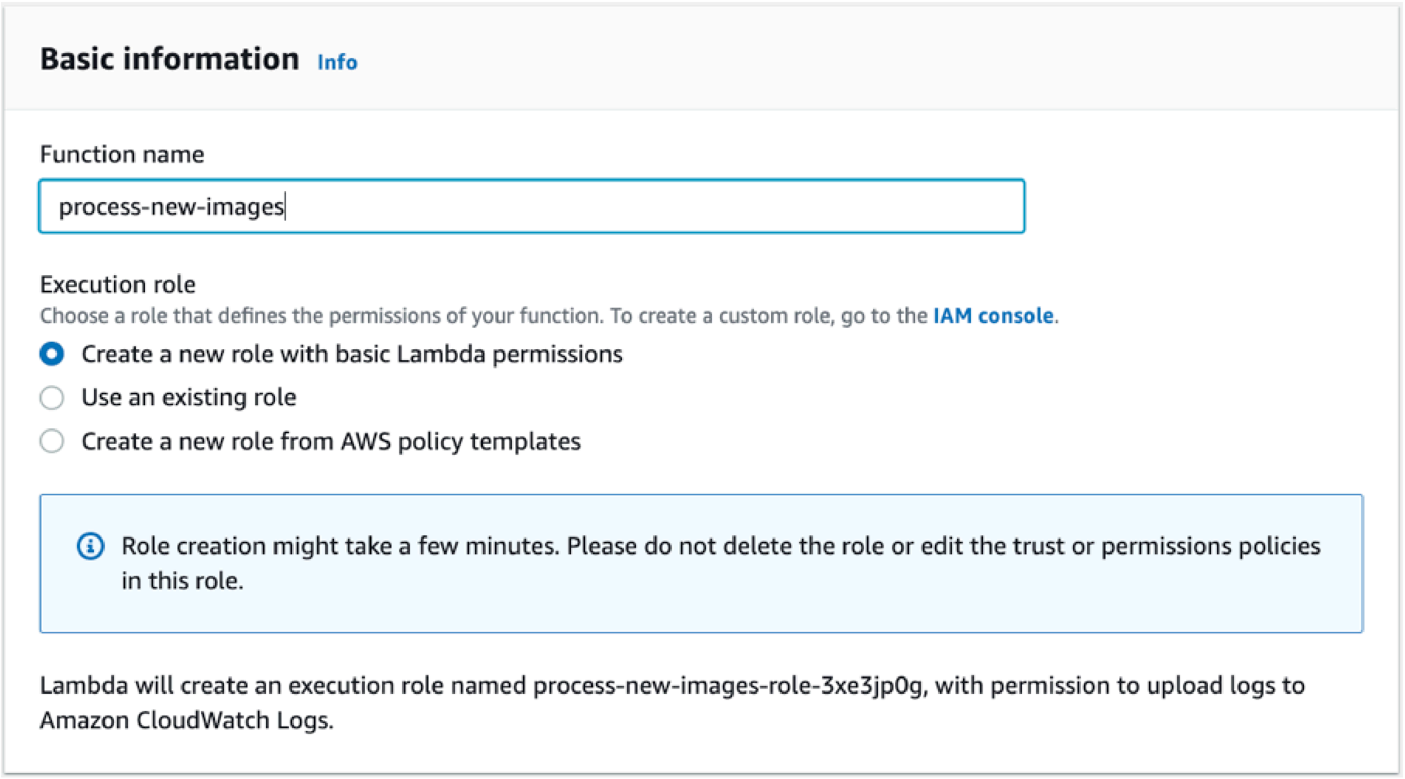
Create a Lambda function to process messages where image references were found (via SQS queue)

Your first Lambda function (`process-new-messages`) is now being invoked, and any image references found in Slack messages have been stored in the `new-image-findings` SQS queue. The next step is to create and invoke another Lambda function (`process-new-images`) that will use Amazon Rekognition to determine if there are any policy violations in the content.

To configure the SQS / Lambda / Amazon Rekognition Integration:

1. Because the Lambda function you are about to create needs to store any content violations found into a new SQS queue, first create that queue by following the steps outlined in [Getting started with Amazon SQS](#).

2. Name this queue `new-violation-findings`.
3. Navigate to the Lambda console and choose **Create Function**.
4. Choose the **Use a blueprint** option and provide a filter called `hello`. This will display the `hello-world-python` blueprint in the results at the bottom.
5. Choose the **Configure** button. Name the new Lambda function `process-new-images`.
6. Create a new execution role with basic Lambda permissions.



Basic information Info

Function name

process-new-images

Execution role

Choose a role that defines the permissions of your function. To create a custom role, go to the IAM console.

[✓ Create a new role with basic Lambda permissions]

Use an existing role

Create a new role from AWS policy templates

Role creation might take a few minutes. Please do not delete the role or edit the trust or permissions policies in this role.

Lambda will create an execution role named process-new-images-role-3xe3jp0g, with permission to upload logs to Amazon CloudWatch Logs.

Create the `process-new-images` function

7. After the function has been created, choose the **Permissions** tab.
8. Choose **IAM Role** to open a second window where you can view the policy attached to this role.
9. Choose **Attach Policies**.
10. Search for `AmazonRekognitionReadOnlyAccess` and choose **Attach Policy** to complete the action. This allows your Lambda function permissions to call Amazon Rekognition.
11. The function also needs permissions to read from the `new-image-findings` queue and write new messages to the `new-violation-findings` queue. Choose **Add inline policy** and switch to the JSON view to create the following permissions.

Note that the following **Resource** elements need to be updated with the correct ARNs for the `new-image-findings` and `new-violation-findings` SQS queues respectively, which contain your actual account number:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "sqs:ReceiveMessage",
                "sqs:ChangeMessageVisibility",
                "sqs:GetQueueUrl",
                "sqs:DeleteMessage",
                "sqs:GetQueueAttributes"
            ],
            "Resource": "arn:aws:sqs:us-east-1:111111111111:new-image-findings",
            "Effect": "Allow"
        },
        {
            "Action": [
                "sqs:SendMessage",
                "sqs:GetQueueAttributes",
                "sqs:GetQueueUrl"
            ],
            "Resource": "arn:aws:sqs:us-east-1:111111111111:new-violation-findings",
            "Effect": "Allow"
        }
    ]
}
```

12. Choose **Review policy**.

13. Enter a name for this policy and choose **Create policy**.

14. With the permissions configured, switch back to the **Configuration** tab in the Lambda function window, and paste the following code into the **Function code** section:

```
import urllib.request
import boto3

sqs = boto3.client('sqs')
rekognition = boto3.client('rekognition')
```

```
def analyze_themes(file, min_confidence=80):
    with open(file, 'rb') as document:
        imageBytes = bytearray(document.read())

    response = rekognition.detect_moderation_labels(Image={'Bytes': imageBytes},
 MinConfidence=min_confidence)

    found_high_confidence_labels = []
    for label in response['ModerationLabels']:
        found_high_confidence_labels.append(str(label['Name']))

    return found_high_confidence_labels


def analyze_text(file):
    with open(file, 'rb') as document:
        imageBytes = bytearray(document.read())

    response = rekognition.detect_text(Image={'Bytes': imageBytes})

    textDetections = response['TextDetections']

    found_text = ""
    for text in textDetections:
        found_text += text['DetectedText']

    return found_text


def sendToSqS(words, attributes, queueurl):

    sqs.send_message(
        QueueUrl=queueurl,
        MessageBody='Image with "' + words + '" found',
        MessageAttributes={
            "url": {
                "StringValue": attributes["image_url"],
                "DataType": 'String'
            },
            "slack_msg_id": {
                "StringValue": attributes["slack_msg_id"],
                "DataType": 'String'
            }
        }
    )


def lambda_handler(event, context):

    violations = "https://queue.amazonaws.com/111111111111/new-violation-findings"

    disallowed_words = ["medical", "private"]

    # Categories listed here - https://docs.aws.amazon.com/rekognition/latest/dg/
moderation.html#moderation-api
    disallowed_themes = ["Tobacco", "Alcohol"]  # Case Sensitive

    file_name = "/tmp/image.jpg"

    for record in event['Records']:
        print(record)
        receiptHandle = record["receiptHandle"]
        image_url = record["messageAttributes"]["url"]["stringValue"]
        slack_msg_id = record["messageAttributes"]["slack_msg_id"]["stringValue"]
        eventSourceARN = record["eventSourceARN"]

        arn_elements = eventSourceARN.split(':')

        img_queue_url = sqs.get_queue_url(
            QueueName=arn_elements[5],
            QueueOwnerAWSAccountId=arn_elements[4]
        )

        sqs.delete_message(
            QueueUrl=img_queue_url["QueueUrl"],
            ReceiptHandle=receiptHandle
        )

        urllib.request.urlretrieve(image_url, file_name)

        detected_text = analyze_text(file_name)

        print("Detected Text: " + detected_text)

        found_words = []
        for disallowed_word in disallowed_words:
            if disallowed_word.lower() in detected_text.lower():
                found_words.append(disallowed_word)
                print("WORD VIOLATION: " + disallowed_word.lower() + " found in " +
detected_text.lower())

        violating_words = ", ".join(found_words)
        if not violating_words == "":
            attributes_json = {}
            attributes_json["slack_msg_id"] = slack_msg_id
            attributes_json["image_url"] = image_url
            sendToSqS(violating_words, attributes_json, violations)

        detected_themes = analyze_themes(file_name)

        print("Detected Themes: " + ", ".join(detected_themes))

        found_themes = []
        for disallowed_theme in disallowed_themes:
            if disallowed_theme in detected_themes:
                found_themes.append(disallowed_theme)
                print("THEME VIOLATION: " + disallowed_theme + " found in image")

        violating_themes = ", ".join(found_themes)
        if not violating_themes == "":
            attributes_json = {}
            attributes_json["slack_msg_id"] = slack_msg_id
            attributes_json["image_url"] = image_url
            sendToSqS(violating_themes, attributes_json, violations)
```

15. After you have pasted the code, update the **violations** variable in the function handler with the correct ARN for the `new-violation-findings` SQS queue which contains your actual account number.
16. Choose **Deploy**.

To ensure that your SQS queues cannot be accessed by resources outside the account, SQS permissions policies can be applied to each of the queues.

To apply permissions policies:

1. Navigate to the SQS console and choose the new-violation-findings queue.
2. Choose the **Access policy** tab.
3. Choose the **Edit** button and paste in the following policy.

Note that the following **Resource** elements need to be updated with the correct ARN for the `new-violation-findings` SQS queues respectively, which contain your actual account number.

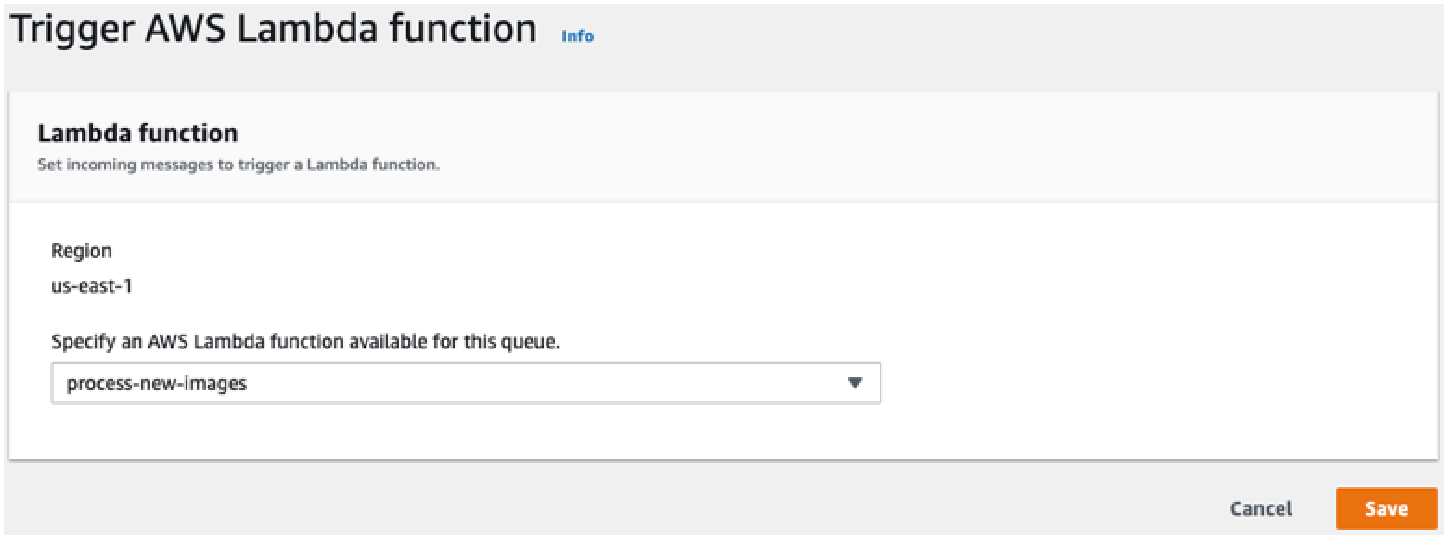
```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "QueueOwnerOnlyAccess",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam:: 111111111111:root"
      },
      "Action": [
        "sqs:DeleteMessage",
        "sqs:ReceiveMessage",
        "sqs:SendMessage",
        "sqs:GetQueueAttributes",
        "sqs:RemovePermission",
        "sqs:AddPermission",
        "sqs:SetQueueAttributes"
      ],
      "Resource": "arn:aws:sqs:us-east-1: 111111111111:new-violation-findings"
    },
    {
      "Sid": "HttpsOnly",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "SQS:*",
      "Resource": "arn:aws:sqs:us-east-1: 111111111111:new-violation-findings",
      "Condition": {
        "Bool": {
          "aws:SecureTransport": "false"
        }
      }
    }
  ]
}
```

4. Repeat the preceding steps for the `new-image-findings` queue. Remember to use the `new-image-findings` ARN in the policy.

You can now configure our SQS queue to trigger your Lambda function.

To configure your SQS queue:

1. In the SQS Console, choose the `new-image-findings` queue from the **Lambda triggers** tab.
2. Choose **Configure Trigger for Lambda Function**.
3. From the dropdown list, choose the function you just created.


Trigger AWS Lambda function Info

Lambda function
Set incoming messages to trigger a Lambda function.

Region
us-east-1

Specify an AWS Lambda function available for this queue.
process-new-images

Trigger the Lambda function you created

Test the solution

You can now post some messages to your moderated Slack channel for testing. You can easily change the content violation policies in the Python code by modifying the `disallowed_words` and `disallowed_themes` variables.

To test the solution:

1. Post sample images that will be used to trigger violations for the current configured policies:
   - Post this image which contains the disallowed word "private": https://i.imgur.com/662ptww.png
   - Post this image which contains a "Tobacco" theme: https://i.imgur.com/XgAtyWU.png
2. After creating those posts, wait 2-3 minutes and then navigate to the SQS Console. View the queues and choose the `new-violation-findings` queue.
3. Choose the **Send and receive messages** button.
4. At the bottom of the screen, choose the **Poll for messages** button.
5. After a few seconds you should see two messages pop up. You can choose each message to interrogate the contents.
6. Choose the **Message ID**. The body of the message contains information about what violation was triggered. The **Attributes** show the image URL and “`slack_msg_id`” for the offending item.

Use the findings

You are now at the stage where images containing content violations (as identified by Amazon Rekognition) have been stored in the SQS queue named `new-violation-findings`. What you do next is up to you. You can take one or more of these (or other) actions:

- Trigger a Lambda Function to notify an Amazon SNS Topic that moderators subscribe to.
- Post a message back into the Slack channel about the violation using an AWS Chatbot.

Cleaning up

To avoid incurring future charges, delete the resources:

1. Delete the two SQS queues: `new-image-findings` and `new-violation-findings`.
2. Delete the two Lambda functions: `process-new-images` and `process-new-messages`.
3. Delete the S3 bucket.
4. Deactivate and delete the AppFlow Flow.

Conclusion

Congratulations — you've built a full Image Moderation pipeline utilizing serverless and managed services. Your solution will automatically scale to handle the volume of messages posted in the moderated Slack channel.

Contributors

Contributors to this document include:

- Haider Abdullah, Senior Partner Solution Architect, Global Systems Integrators
- John Culkin, Partner Solution Architect, Business Consultant and Advisory Partners

Document Revisions

To be notified about updates to this whitepaper, subscribe to the RSS feed.

Change	Description	Date
Initial publication	Initial publication	April 14, 2021

Notices