AWS Messaging & Targeting Blog

Forward Incoming Email to an External Destination

by Brent Meyer | on 08 OCT 2019 | in Amazon Simple Email Service (SES), Intermediate (200), Messaging | Permalink | Comments | Share

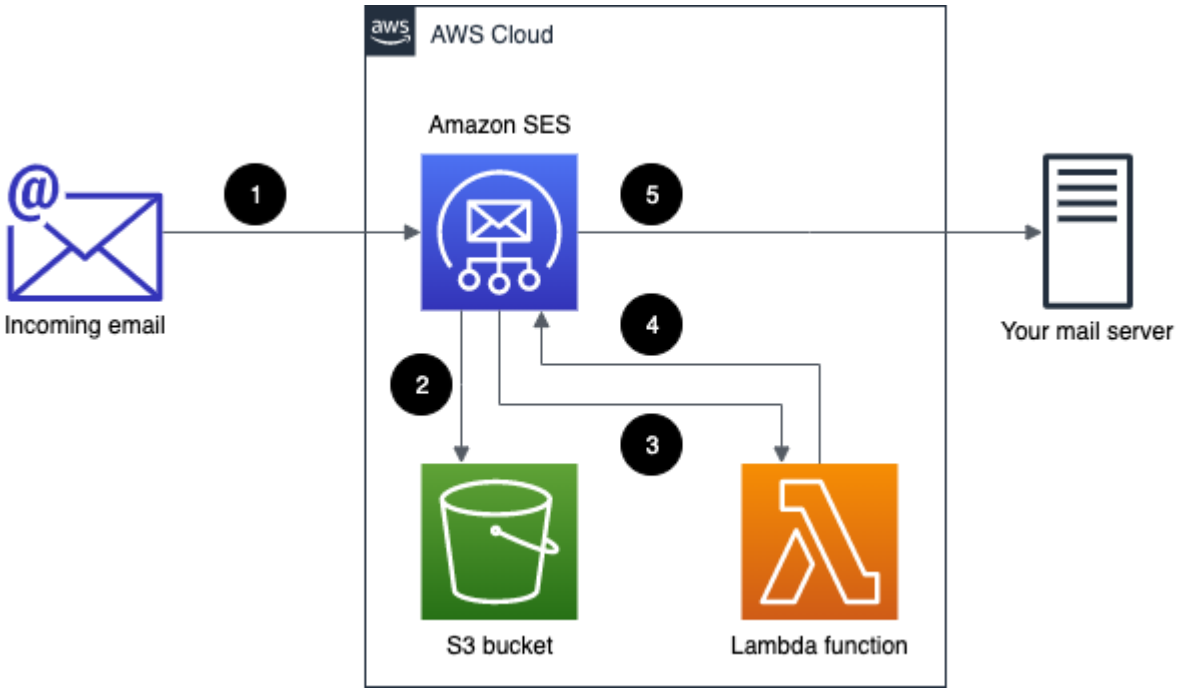
https://aws.amazon.com/blogs/messaging-and-targeting/for

Note: This post was written by Vesselin Tzvetkov, an AWS Senior Security Architect, and by Rostislav Markov, an AWS Senior Engagement Manager.

Amazon SES has included support for incoming email for several years now. However, some customers have told us that they need a solution for forwarding inbound emails to domains that aren't managed by Amazon SES. In this post, you'll learn how to use Amazon SES, Amazon S3, and AWS Lambda to create a solution that forwards incoming email to an email address that's managed outside of Amazon SES.

Architecture

This solution uses several AWS services to forward incoming emails to a single, external email address. The following diagram shows the flow of information in this solution.



The following actions occur in this solution:

1. A new email is sent from an external sender to your domain. Amazon SES handles the incoming email for your domain.
2. An Amazon SES receipt rule saves the incoming message in an S3 bucket.
3. An Amazon SES receipt rule triggers the execution of a Lambda function.

4. The Lambda function retrieves the message content from S3, and then creates a new message and sends it to Amazon SES.
5. Amazon SES sends the message to the destination server.

Limitations

This solution works in all AWS Regions where Amazon SES is currently available. For a complete list of supported Regions, see AWS Service Endpoints in the *AWS General Reference*.

Prerequisites

In order to complete this procedure, you need to have a domain that receives incoming email. If you don't already have a domain, you can purchase one through Amazon Route 53. For more information, see Registering a New Domain in the *Amazon Route 53 Developer Guide*. You can also purchase a domain through one of several third-party vendors.

Procedures

Step 1: Set up Your Domain

1. In Amazon SES, verify the domain that you want to use to receive incoming email. For more information, see Verifying Domains in the *Amazon SES Developer Guide*.
2. Add the following MX record to the DNS configuration for your domain:

```
10 inbound-smtp.<regionInboundUrl>.amazonaws.com
```

   Replace `<regionInboundUrl>` with the URL of the email receiving endpoint for the AWS Region that you use Amazon SES in. For a complete list of URLs, see AWS Service Endpoints – Amazon SES in the *AWS General Reference*.

3. If your account is still in the Amazon SES sandbox, submit a request to have it removed. For more information, see Moving Out of the Sandbox in the *Amazon SES Developer Guide*.

Step 2: Configure Your S3 Bucket

1. In Amazon S3, create a new bucket. For more information, see Create a Bucket in the *Amazon S3 Getting Started Guide*.
2. Apply the following policy to the bucket:

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Sid": "AllowSESPuts",
            "Effect": "Allow",
            "Principal": {
                "Service": "ses.amazonaws.com"
            },
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::<bucketName>/*",
            "Condition": {
                "StringEquals": {
                    "aws:Referer": "<awsAccountId>"
                }
            }
        }
    ]
}
```

3. In the policy, make the following changes:
   - Replace `<bucketName>` with the name of your S3 bucket.
   - Replace `<awsAccountId>` with your AWS account ID.

   For more information, see Using Bucket Policies and User Policies in the *Amazon S3 Developer Guide*.

Step 3: Create an IAM Policy and Role

1. Create a new IAM Policy with the following permissions:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:CreateLogGroup",
                "logs:PutLogEvents"
            ],
            "Resource": "*"
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
```

```
                "s3:GetObject",
                "ses:SendRawEmail"
            ],
            "Resource": [
                "arn:aws:s3:::<bucketName>/*",
                "arn:aws:ses:<region>:<awsAccountId>:identity/*"
            ]
        }
    ]
}
```

In the preceding policy, make the following changes:

- Replace `<bucketName>` with the name of the S3 bucket that you created earlier.
- Replace `<region>` with the name of the AWS Region that you created the bucket in.
- Replace `<awsAccountId>` with your AWS account ID. For more information, see Create a Customer Managed Policy in the *IAM User Guide*.

2. Create a new IAM role. Attach the policy that you just created to the new role. For more information, see Creating Roles in the *IAM User Guide*.

Step 4: Create the Lambda Function

1. In the Lambda console, create a new Python 3.7 function from scratch. For the execution role, choose the IAM role that you created earlier.
2. In the code editor, paste the following code.

Python

```
# Copyright 2010-2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#
# This file is licensed under the Apache License, Version 2.0 (the "License").
# You may not use this file except in compliance with the License. A copy of the
# License is located at
#
# http://aws.amazon.com/apache2.0/
#
# This file is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS
# OF ANY KIND, either express or implied. See the License for the specific
# language governing permissions and limitations under the License.

import os
import boto3
import email
import re
from botocore.exceptions import ClientError
from email.mime.multipart import MIMEMultipart
```

```
from email.mime.text import MIMEText
from email.mime.application import MIMEApplication

region = os.environ['Region']

def get_message_from_s3(message_id):

    incoming_email_bucket = os.environ['MailS3Bucket']
    incoming_email_prefix = os.environ['MailS3Prefix']

    if incoming_email_prefix:
        object_path = (incoming_email_prefix + "/" + message_id)
    else:
        object_path = message_id

    object_http_path = (f"http://s3.console.aws.amazon.com/s3/object/{incoming_em

    # Create a new S3 client.
    client_s3 = boto3.client("s3")

    # Get the email object from the S3 bucket.
    object_s3 = client_s3.get_object(Bucket=incoming_email_bucket,
        Key=object_path)
    # Read the content of the message.
    file = object_s3['Body'].read()

    file_dict = {
        "file": file,
        "path": object_http_path
    }

    return file_dict

def create_message(file_dict):

    sender = os.environ['MailSender']
    recipient = os.environ['MailRecipient']

    separator = ";"

    # Parse the email body.
    mailobject = email.message_from_string(file_dict['file'].decode('utf-8'))

    # Create a new subject line.
    subject_original = mailobject['Subject']
    subject = "FW: " + subject_original
```

```
    # The body text of the email.
    body_text = ("The attached message was received from "
                 + separator.join(mailobject.get_all('From'))
                 + ". This message is archived at " + file_dict['path'])

    # The file name to use for the attached message. Uses regex to remove all
    # non-alphanumeric characters, and appends a file extension.
    filename = re.sub('[^0-9a-zA-Z]+', '_', subject_original) + ".eml"

    # Create a MIME container.
    msg = MIMEMultipart()
    # Create a MIME text part.
    text_part = MIMEText(body_text, _subtype="html")
    # Attach the text part to the MIME message.
    msg.attach(text_part)

    # Add subject, from and to lines.
    msg['Subject'] = subject
    msg['From'] = sender
    msg['To'] = recipient

    # Create a new MIME object.
    att = MIMEApplication(file_dict["file"], filename)
    att.add_header("Content-Disposition", 'attachment', filename=filename)

    # Attach the file object to the message.
    msg.attach(att)

    message = {
        "Source": sender,
        "Destinations": recipient,
        "Data": msg.as_string()
    }

    return message

def send_email(message):
    aws_region = os.environ['Region']

# Create a new SES client.
    client_ses = boto3.client('ses', region)

    # Send the email.
    try:
        #Provide the contents of the email.
```

```
        response = client_ses.send_raw_email(
            Source=message['Source'],
            Destinations=[
                message['Destinations']
            ],
            RawMessage={
                'Data':message['Data']
            }
        )

    # Display an error if something goes wrong.
    except ClientError as e:
        output = e.response['Error']['Message']
    else:
        output = "Email sent! Message ID: " + response['MessageId']

    return output

def lambda_handler(event, context):
    # Get the unique ID of the message. This corresponds to the name of the file
    # in S3.
    message_id = event['Records'][0]['ses']['mail']['messageId']
    print(f"Received message ID {message_id}")

    # Retrieve the file from the S3 bucket.
    file_dict = get_message_from_s3(message_id)

    # Create the message.
    message = create_message(file_dict)

    # Send the email and print the result.
    result = send_email(message)
    print(result)
```

3. Create the following environment variables for the Lambda function:

Key	Value
`MailS3Bucket`	The name of the S3 bucket that you created earlier.
`MailS3Prefix`	The path of the folder in the S3 bucket where you will store incoming email.
`MailSender`	The email address that the forwarded message will be sent from. This address has to be verified.
`MailRecipient`	The address that you want to forward the message to.

`Region`	The name of the AWS Region that you want to use to send the email.

4. Under **Basic settings**, set the **Timeout** value to 30 seconds.

(Optional) Step 5: Create an Amazon SNS Topic

You can optionally create an Amazon SNS topic. This step is helpful for troubleshooting purposes, or if you just want to receive additional notifications when you receive a message.

1. Create a new Amazon SNS topic. For more information, see Creating a Topic in the *Amazon SNS Developer Guide*.
2. Subscribe an endpoint, such as an email address, to the topic. For more information, see Subscribing an Endpoint to a Topic in the *Amazon SNS Developer Guide*.

Step 6: Create a Receipt Rule Set

1. In the Amazon SES console, create a new Receipt Rule Set. For more information, see Creating a Receipt Rule Set in the *Amazon SES Developer Guide*.
2. In the Receipt Rule Set that you just created, add a Receipt Rule. In the Receipt Rule, add an S3 Action. Set up the S3 Action to send your email to the S3 bucket that you created earlier.
3. Add a Lambda action to the Receipt Rule. Configure the Receipt Rule to invoke the Lambda function that you created earlier.

For more information, see Setting Up a Receipt Rule in the *Amazon SES Developer Guide*.

Step 7: Test the Function

- Send an email to an address that corresponds with an address in the Receipt Rule you created earlier. Make sure that the email arrives in the correct S3 bucket. In a minute or two, the email arrives in the inbox that you specified in the `MailRecipient` variable of the Lambda function.

Troubleshooting

If you send a test message, but it is never forwarded to your destination email address, do the following:

- Make sure that the Amazon SES Receipt Rule is active.
- Make sure that the email address that you specified in the MailRecipient variable of the Lambda function is correct.
- Subscribe an email address or phone number to the SNS topic. Send another test email to your domain. Make sure that SNS sends a Received notification to your subscribed email address or phone number.
- Check the CloudWatch Log for your Lambda function to see if any errors occurred.

If you send a test email to your receiving domain, but you receive a bounce notification, do the following:

- Make sure that the verification process for your domain completed successfully.
- Make sure that the MX record for your domain specifies the correct Amazon SES receiving endpoint.

- Make sure that you're sending to an address that is handled by the receipt rule.

Costs of using this solution

The cost of implementing this solution is minimal. If you receive 10,000 emails per month, and each email is 2KB in size, you pay $1.00 for your use of Amazon SES. For more information, see Amazon SES Pricing.

You also pay a small charge to store incoming emails in Amazon S3. The charge for storing 1,000 emails that are each 2KB in size is less than one cent. Your use of Amazon S3 might qualify for the AWS Free Usage Tier. For more information, see Amazon S3 Pricing.

Finally, you pay for your use of AWS Lambda. With Lambda, you pay for the number of requests you make, for the amount of compute time that you use, and for the amount of memory that you use. If you use Lambda to forward 1,000 emails that are each 2KB in size, you pay no more than a few cents. Your use of AWS Lambda might qualify for the AWS Free Usage Tier. For more information, see AWS Lambda Pricing.

> **Note**: These cost estimates don't include the costs associated with purchasing a domain, since many users already have their own domains. The cost of obtaining a domain is the most expensive part of implementing this solution.

Conclusion

This solution makes it possible to forward incoming email from one of your Amazon SES verified domains to an email address that isn't necessarily verified. It's also useful if you have multiple AWS accounts, and you want incoming messages to be sent from each of those accounts to a single destination. We hope you've found this tutorial to be helpful!

TAGS: forward