

STACS SOLUTIONS ENGINEERING

Email Automation Service with
AWS SES and SNS



PREFACE

The STACS Solutions Engineering team was looking to build a fully automated service that processes emails to enhance our existing AWS hosted Application service. This automation would allow a user to send emails to our application service where it would then process the data in the incoming email payload before sending the processed data to digest back to the user.

A few considerations came to mind when designing a robust solution that could scale, namely:

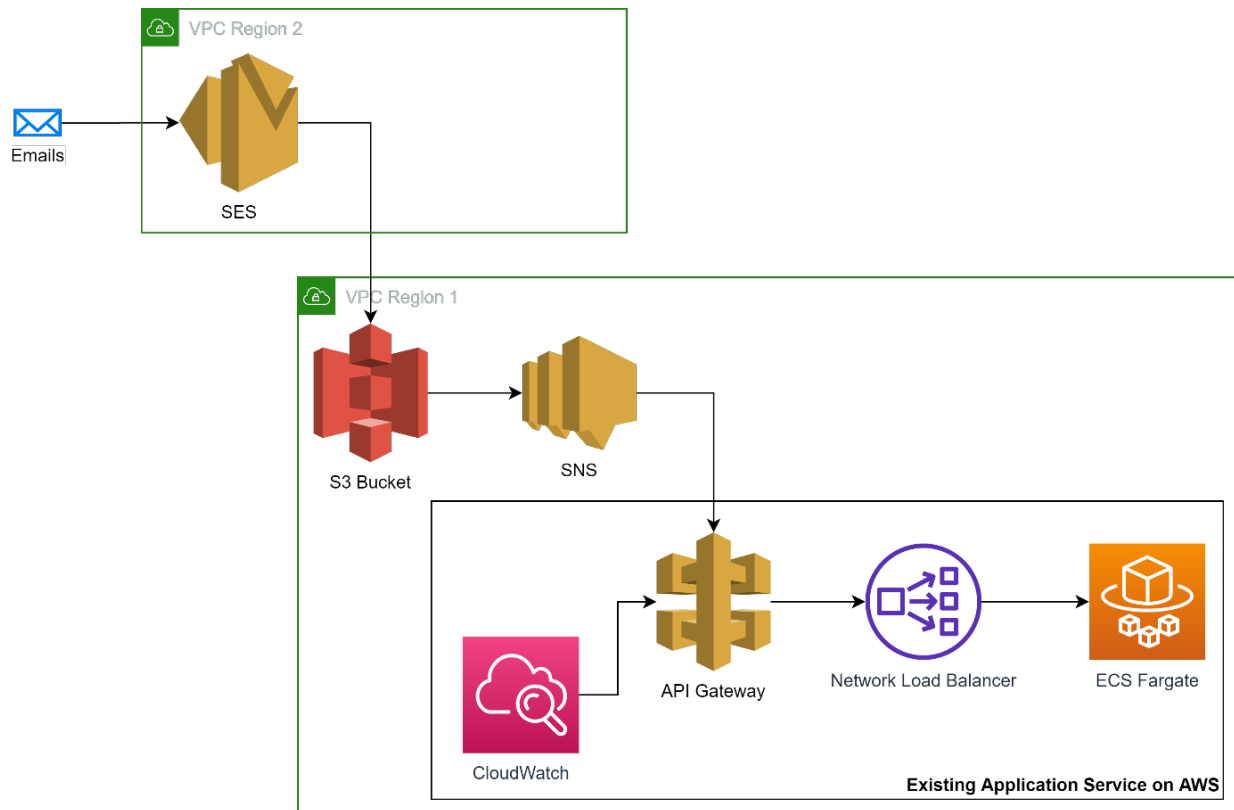
- We needed to ensure the application service could retry processing of an incoming email should the first attempt failed.
- The response back to the user is time critical since the sending of the processed payload to the user must be ASAP
- The solution must be able to handle many email messages for the processing and should scale easily
- Works with our existing application service that is deployed with API Gateway

Hence, we decided to design the solution on the AWS Cloud to enable integration with our current application services and to take advantage of the managed services by AWS to reduce the need for maintenance and dedicated manpower.

CONTENTS

1	Technical Architecture	3
2	Technical Solution Breakdown	4
2.1	SES Domain Setup and Rule Sets Configuration	4
2.1.1	Verifying a New Domain	4
2.1.2	Setting up Rules for incoming emails	4
2.2	SNS Topic Setup and Subscription	5
2.3	S3 Bucket	5
2.3.1	Setting up bucket permission for SES	5
2.3.2	Connecting S3 Events to SNS	6
3	Conclusion	7

1 Technical Architecture



At a high level, we used 3 main AWS Managed services to take advantage of the automated scalability of said services that requires minimal resources from our engineering team to integrate to our existing application service.

The Simple Email Service (SES) enabled us to accept incoming emails without requiring us to spin up a dedicated SMTP email server to service incoming emails. Since we deployed our application service in the Singapore region and there are no SES services for this region, we had to use another region’s SES service while we kept all other services in the same Singapore region.

To enable our application service deployed as containers on ECS Fargate to rerun email workloads in the event of an initial processing failure, we decided to go ahead with S3 for object storage.

Simple Notification Service (SNS) allowed us to handle incoming emails in real-time since it pushes objects in S3 to the application service and the wide variety of available subscription endpoints to SNS allowed us to enable internal monitoring with minimal configuration efforts.

Although SNS functions as a glue between services, the key feature in our solution was a S3 feature, S3 Events, which worked nicely with SNS to provide a real-time push notification service to support our requirements.

2 Technical Solution Breakdown

Provide more detailed diagrams and breakdown the steps of the solution

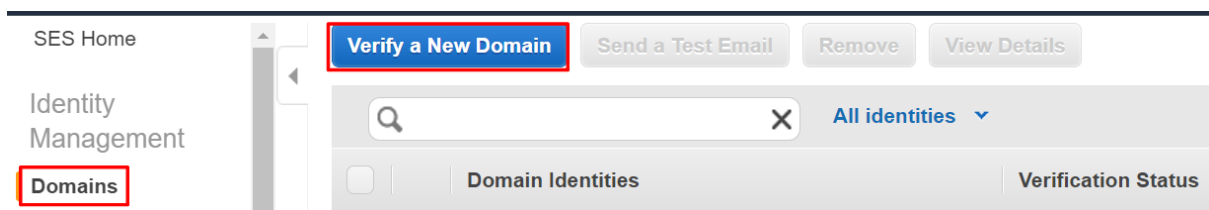
2.1 SES Domain Setup and Rule Sets Configuration

To start off, we needed to setup SES to accept incoming emails on the AWS Cloud.

Since SES is limited to specific AWS Regions and not available in our current Region where our application service is deployed, we simply chose SES based on geographical proximity as well as available SES features.

2.1.1 Verifying a New Domain

The first step was to setup an email Domain in SES by clicking on Verify a New Domain.



Once a domain has been setup, the verification will need to be done on the DNS provider's end and we followed the [detailed instructions](#) provided by AWS which was to add a TXT record.

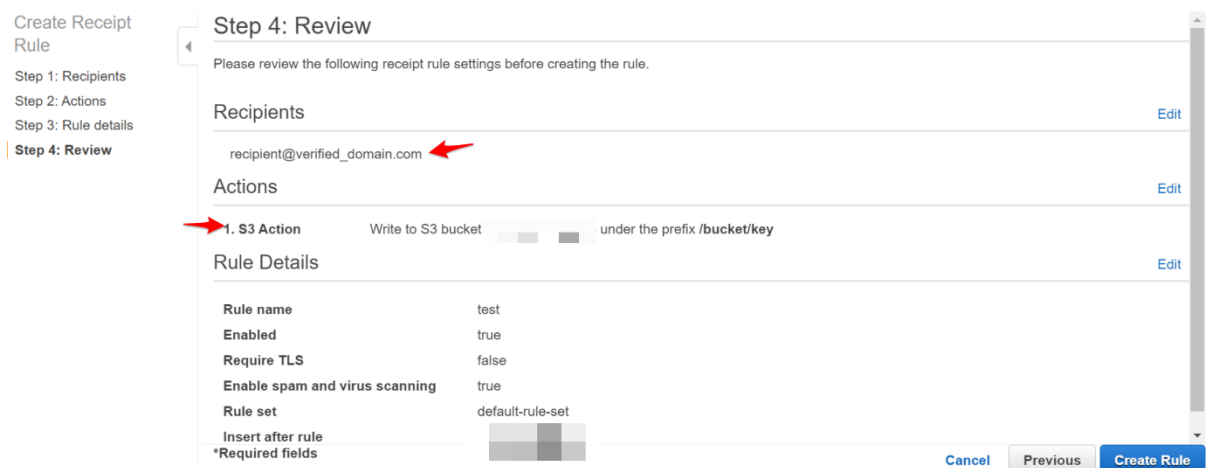
2.1.2 Setting up Rules for incoming emails

With the domain verified, we can start defining the flow of emails using Rule Sets.

The recipient of the Rule Set was an email address related to the domain we had just verified earlier. In this case, it may look like [recipient@verified_domain.com](#) as shown in our screenshot below where we expect incoming emails to be sent to this email address.

The next step is to add Actions which in our case is to write to an S3 bucket.

Note that in this step under Actions there is an option to add an SNS topic. We decided to add the SNS topic at the S3 bucket instead since we were using the same S3 bucket for other object interactions and it would be easier to manage all cascading events from the S3 bucket.



2.2 SNS Topic Setup and Subscription

SNS is used to inform the application service when a new email has arrived via SES and into the S3 bucket, so the first setup is to create a new SNS Topic.

To ensure that the S3 Bucket has permission to publish to the SNS topic, the new SNS topic needs to grant the permission to the S3 Bucket. The following is an example of the SNS topic's access policy:

```

1. "Condition": {
2.     "StringEquals": {
3.         "aws:SourceAccount": "<account id>"
4.     },
5.     "ArnLike": {
6.         "aws:SourceArn": "arn:aws:s3:::<bucket name>"
7.     }
8. }
```

Once the topic has been created, then we can proceed to create subscriptions to the topic. In our case, as our application service will be notified via HTTPS endpoint, we use the AWS Management Console to register the service's HTTPS Endpoint by creating a new subscription with the HTTPS protocol and adding our endpoint:

The screenshot shows the AWS Management Console interface for configuring an SNS subscription. It features two main sections: 'Protocol' and 'Endpoint'. The 'Protocol' section has a dropdown menu currently showing 'HTTPS'. The 'Endpoint' section has a text input field containing the URL 'https://www.example.com'. Below the input field, there is a small blue icon representing a globe.

When the subscription is first created, the confirmation link is sent to the endpoint and the URL for confirmation is visible in CloudWatch if logging is enabled at the endpoint. This enables us to confirm the registration of the endpoint to SNS and to begin receiving notifications when the topic is triggered.

Additional monitoring can be done at this configuration step such as creating subscriptions for email addresses that will be notified when a new email arrives which enables us to debug in greater detail.

2.3 S3 Bucket

With both SES and SNS setup, we can proceed to setup the receiving S3 Bucket.

2.3.1 Setting up bucket permission for SES

Additional configuration is required on the S3 Bucket to enable SES to put objects. The following is an example of how to grant permission to SES in the Bucket Policy:

```

1. {
2.     "Version": "2012-10-17",
3.     "Statement": [
4.         {
5.             "Sid": "AllowSESPuts-<Sid number>",
6.             "Effect": "Allow",
7.             "Principal": {
```

```

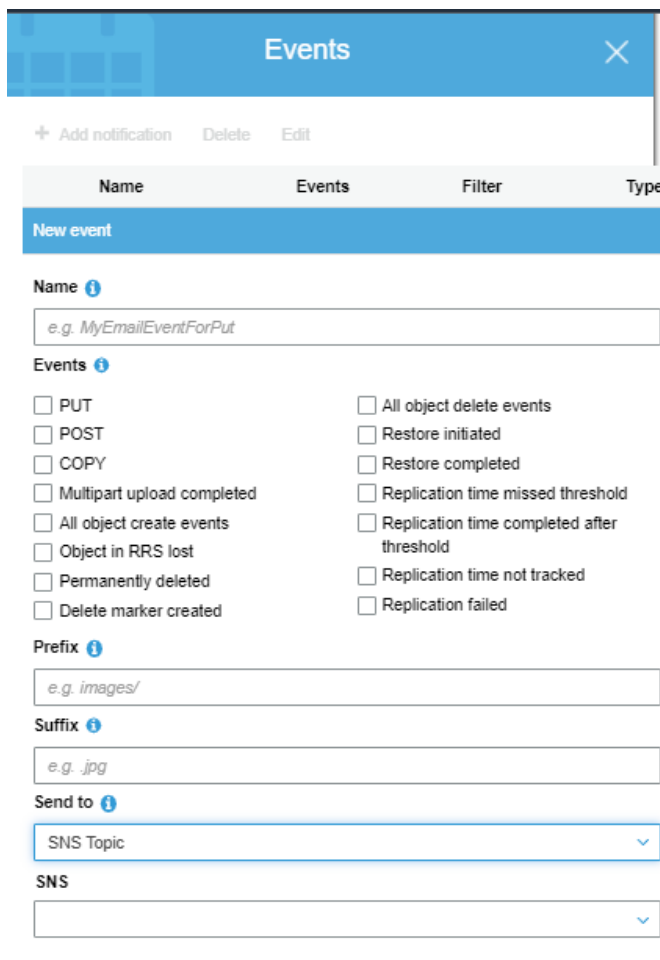
8.         "Service": "ses.amazonaws.com"
9.     },
10.    "Action": "s3:PutObject",
11.    "Resource": "arn:aws:s3:::<bucket name>/*",
12.    "Condition": {
13.        "StringEquals": {
14.            "aws:Referer": "<account-id>"
15.        }
16.    }
17. }
18. ]
19. }

```

2.3.2 Connecting S3 Events to SNS

The final piece of the puzzle is to connect the bucket to SNS.

The AWS Management console provides an easy way to do this under the S3 Bucket's Properties tab. Navigating further down to the Events section, we can easily setup the various Events scenarios (e.g. PUT) and define the exact sub location within the bucket (under Prefix) while selecting the previously created SNS topic with a dropdown list. The screenshot below shows the ease of use when it comes to setting up the final connection which is between S3 and SNS.



The screenshot displays the 'Events' configuration interface in the AWS Management console. At the top, there are buttons for '+ Add notification', 'Delete', and 'Edit'. Below this is a table with columns: Name, Events, Filter, and Type. A 'New event' button is visible. The configuration fields include:

- Name:** A text input field with the example 'e.g. MyEmailEventForPut'.
- Events:** A list of checkboxes for various event types:
 - PUT
 - POST
 - COPY
 - Multipart upload completed
 - All object create events
 - Object in RRS lost
 - Permanently deleted
 - Delete marker created
 - All object delete events
 - Restore initiated
 - Restore completed
 - Replication time missed threshold
 - Replication time completed after threshold
 - Replication time not tracked
 - Replication failed
- Prefix:** A text input field with the example 'e.g. images/'.
- Suffix:** A text input field with the example 'e.g. .jpg'.
- Send to:** A dropdown menu currently showing 'SNS Topic'.
- SNS:** A dropdown menu.

3 Conclusion

The solution presented does meet all requirements. The use of S3 allows the service to retry processing an incoming email after the first try. Push notification of SNS ensures that the processing is done as soon as the emails are received. The solution also works well with our existing application setup on API Gateway since we could simply connect this solution to our existing application service.

Since the above are managed services by AWS, we offload scaling to AWS and save us the effort behind spinning up an SMTP email server (SES) and a messaging server (SNS) and then dedicating manpower to maintain both services.

Some improvements that we are looking into would be to add Simple Queue Service (SQS) to our SNS pipeline to enhance the robustness of the processing process. SQS FIFO queues would enable the application service to acknowledge and delete processed items in the queue, further automating retries at the application service layer.

Our design also enabled reuse of the same S3 bucket where we can apply granular logic via S3 Events to dictate custom processing of different objects in the same bucket. This allows us to manage future feature requests for custom processing within the same dashboard and bucket.

The final solution is an end-to-end automated, robust workflow that helps us handle email requests at a scalable level. The email automation service had minimal impact to our current application service since we simply added on more AWS managed services like LEGO blocks.