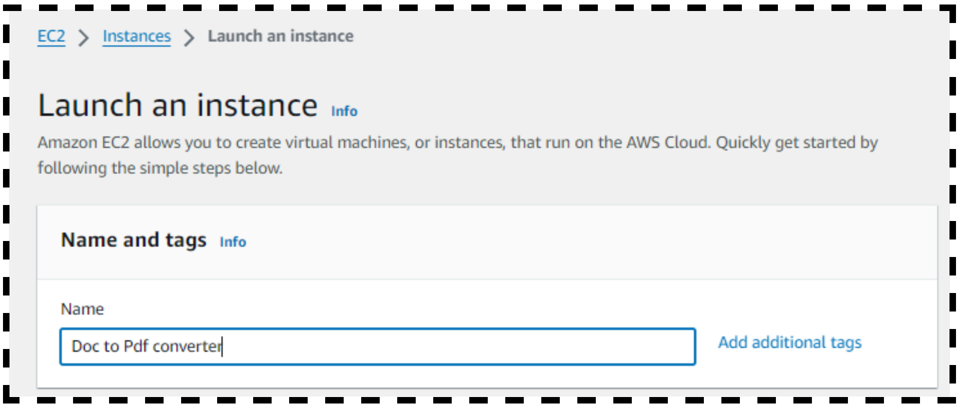
EC2 File Conversion App

- **Step-by-Step Implementation**

1. Setup the EC2 Instance

1. **Launch an EC2 instance** with the latest Amazon Linux 2.



EC2 > Instances > Launch an instance

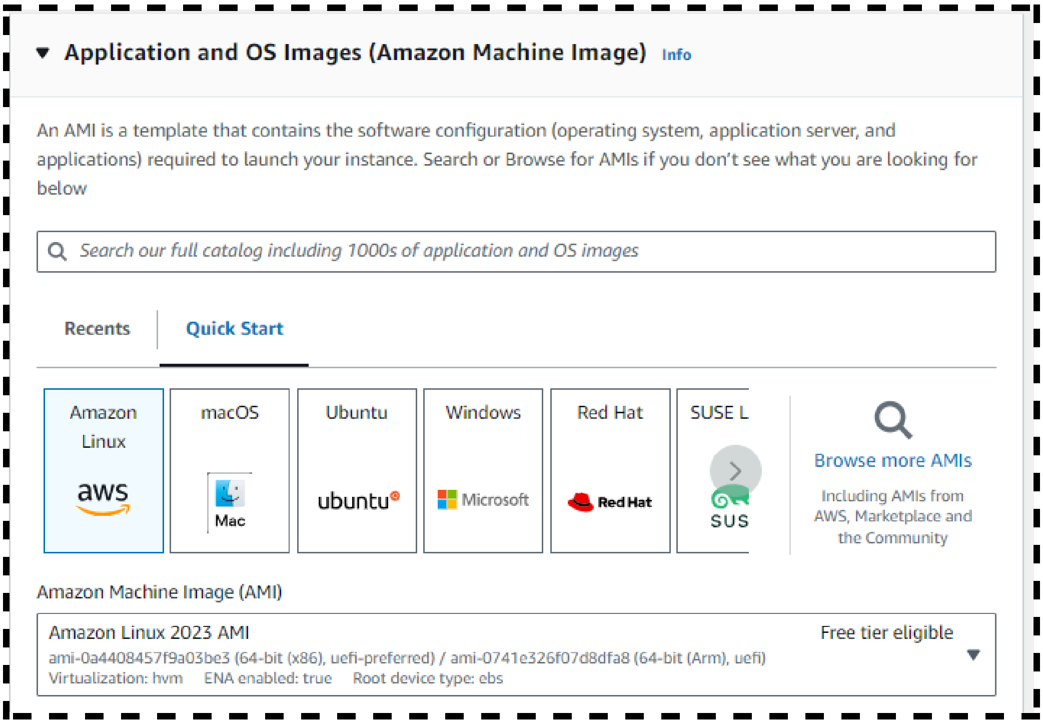
Launch an instance Info

Amazon EC2 allows you to create virtual machines, or instances, that run on the AWS Cloud. Quickly get started by following the simple steps below.

Name and tags Info

Name: Doc to Pdf converter

- Choosing the Amazon Linux2 machine



▼ Application and OS Images (Amazon Machine Image) Info

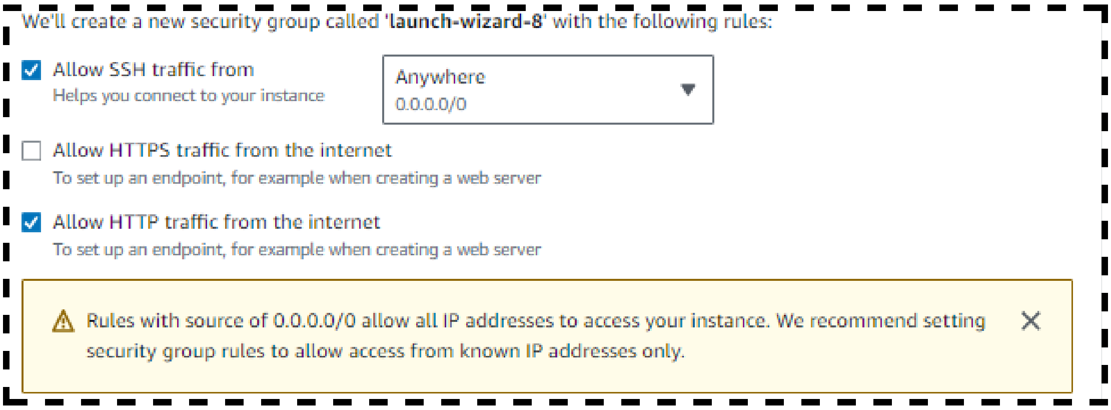
An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below

Search our full catalog including 1000s of application and OS images

Amazon Machine Image (AMI)

Amazon Linux 2023 AMI — Free tier eligible
ami-0a4408457f9a03be3 (64-bit (x86), uefi-preferred) / ami-0741e326f07d8dfa8 (64-bit (Arm), uefi)
Virtualization: hvm ENA enabled: true Root device type: ebs

- In Inbound rule choose allow ssh and the http



We'll create a new security group called '**launch-wizard-8**' with the following rules:

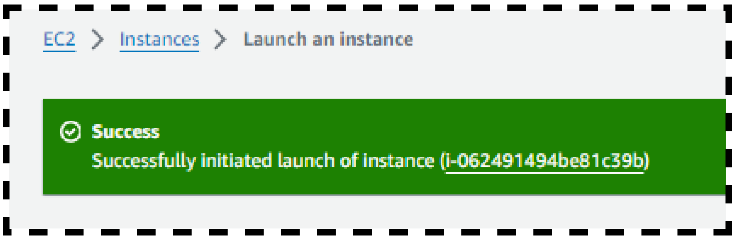
☑ Allow SSH traffic from
Helps you connect to your instance

□ Allow HTTPS traffic from the internet
To set up an endpoint, for example when creating a web server

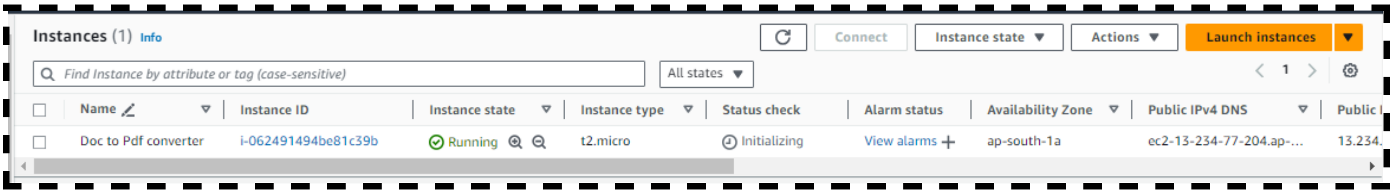
☑ Allow HTTP traffic from the internet
To set up an endpoint, for example when creating a web server

⚠ Rules with source of 0.0.0.0/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.

- Instance created successfully



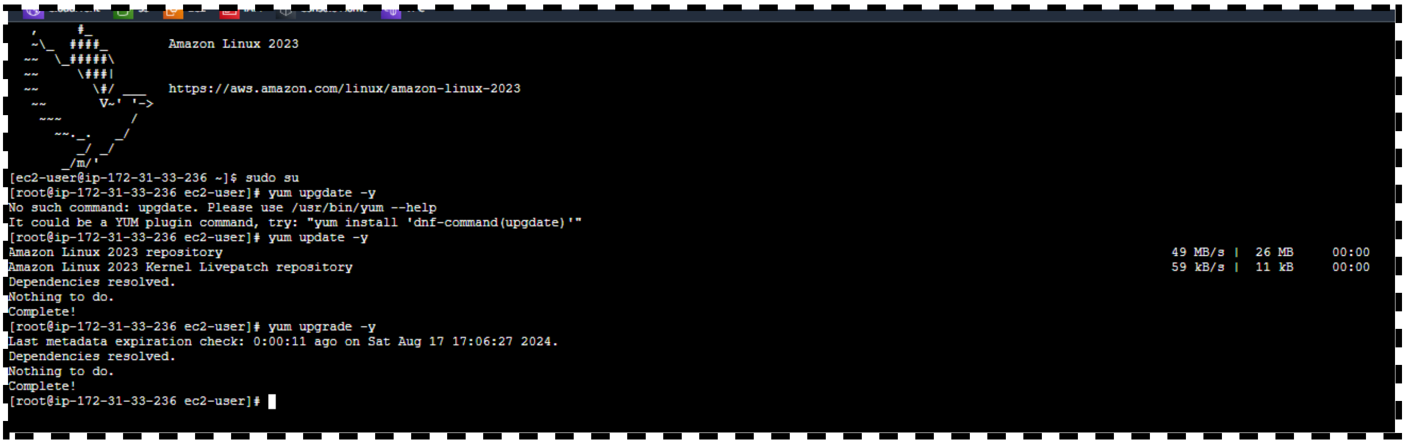
Success
Successfully initiated launch of instance (i-062491494be81c39b)



□	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public I
□	Doc to Pdf converter	i-062491494be81c39b	✓ Running	t2.micro	Initializing	View alarms +	ap-south-1a	ec2-13-234-77-204.ap-...	13.234.

- Connect the Ec2 Instance to run the commands and to perform the tasks
- Run the Following the Commands:

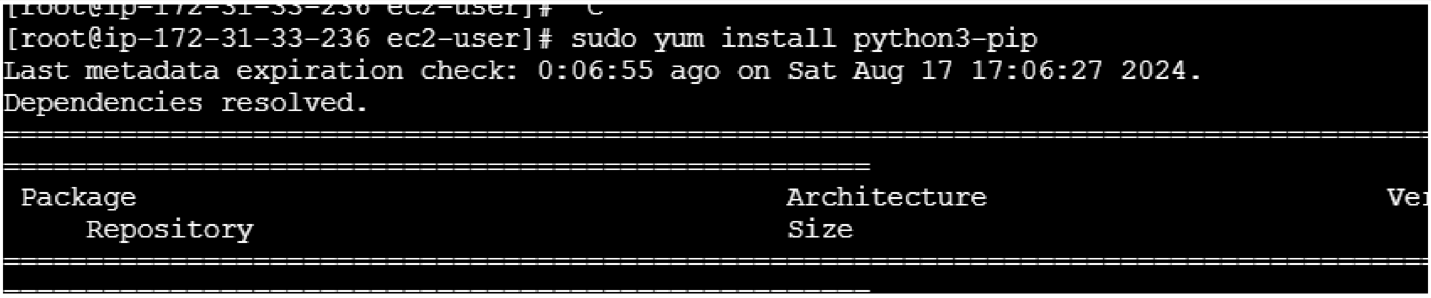
```
sudo yum update -y
sudo yum install python3 python3-pip -y
sudo pip3 install flask boto3
```



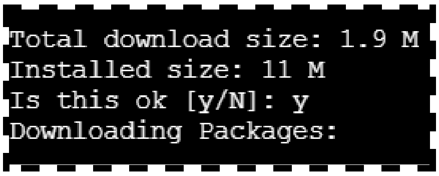
```
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
[ec2-user@ip-172-31-33-236 ~]$ sudo su
[root@ip-172-31-33-236 ec2-user]# yum upgdate -y
No such command: upgdate. Please use /usr/bin/yum --help
It could be a YUM plugin command, try: "yum install 'dnf-command(upgdate)'"
[root@ip-172-31-33-236 ec2-user]# yum update -y
Amazon Linux 2023 repository                                                          49 MB/s |  26 MB     00:00
Amazon Linux 2023 Kernel Livepatch repository                                         59 kB/s |  11 kB     00:00
Dependencies resolved.
Nothing to do.
Complete!
[root@ip-172-31-33-236 ec2-user]# yum upgrade -y
Last metadata expiration check: 0:00:11 ago on Sat Aug 17 17:06:27 2024.
Dependencies resolved.
Nothing to do.
Complete!
[root@ip-172-31-33-236 ec2-user]# █
```

- Python setup

sudo yum install python3-pip

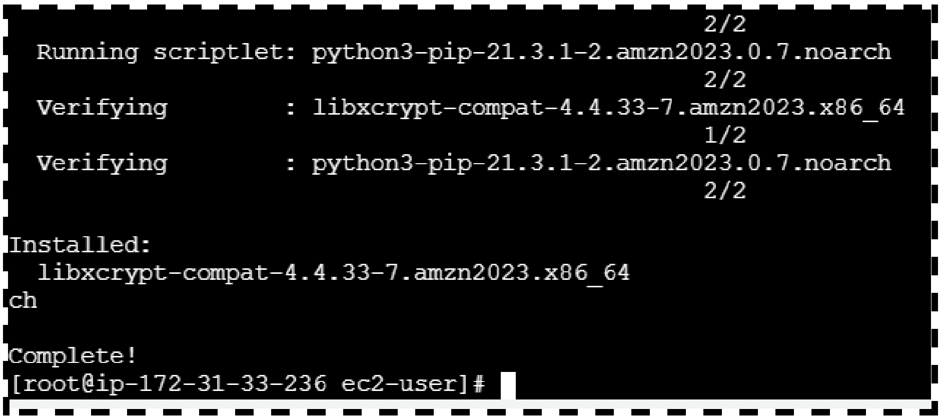


```
[root@ip-172-31-33-236 ec2-user]# ^C
[root@ip-172-31-33-236 ec2-user]# sudo yum install python3-pip
Last metadata expiration check: 0:06:55 ago on Sat Aug 17 17:06:27 2024.
Dependencies resolved.
==========================================================================================
=====================================================
 Package                                             Architecture                        Ve
     Repository                                      Size
==========================================================================================
=====================================================
```



```
Total download size: 1.9 M
Installed size: 11 M
Is this ok [y/N]: y
Downloading Packages:
```

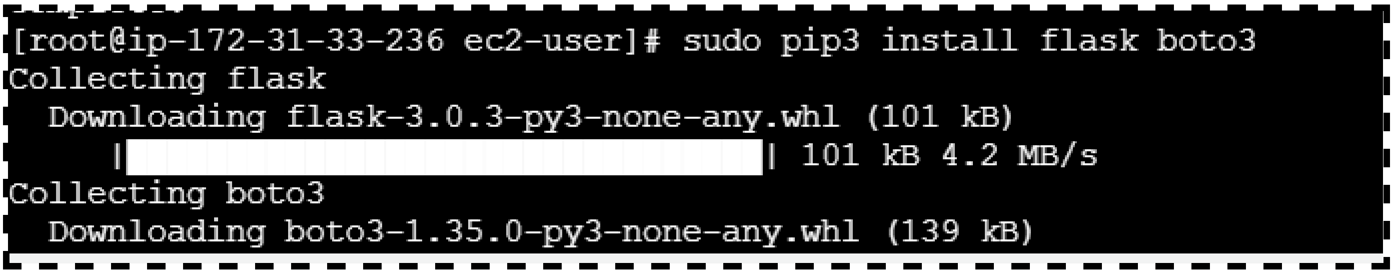
- Pip installed Successfully



```
                                                       2/2
  Running scriptlet: python3-pip-21.3.1-2.amzn2023.0.7.noarch
                                                       2/2
  Verifying        : libxcrypt-compat-4.4.33-7.amzn2023.x86_64
                                                       1/2
  Verifying        : python3-pip-21.3.1-2.amzn2023.0.7.noarch
                                                       2/2

Installed:
  libxcrypt-compat-4.4.33-7.amzn2023.x86_64
ch

Complete!
[root@ip-172-31-33-236 ec2-user]# 
```

- Flask Setup

sudo pip3 install flask boto3



```
[root@ip-172-31-33-236 ec2-user]# sudo pip3 install flask boto3
Collecting flask
  Downloading flask-3.0.3-py3-none-any.whl (101 kB)
     |████████████████████████████████| 101 kB 4.2 MB/s
Collecting boto3
  Downloading boto3-1.35.0-py3-none-any.whl (139 kB)
```

- Follow the following steps to create the web application into the ec2

```
[root@ip-172-31-33-236 ec2-user]# mkdir my-web-app
[root@ip-172-31-33-236 ec2-user]# cd my-web-app
[root@ip-172-31-33-236 my-web-app]# nano app.py
[root@ip-172-31-33-236 my-web-app]# nano app.py
[root@ip-172-31-33-236 my-web-app]# cat nano app.py
```

```
[root@ip-172-31-33-236 ec2-user]# mkdir my-web-app
[root@ip-172-31-33-236 ec2-user]# cd my-web-app
[root@ip-172-31-33-236 my-web-app]# nano app.py
[root@ip-172-31-33-236 my-web-app]# nano app.py
[root@ip-172-31-33-236 my-web-app]# cat nano app.py
```

- **Create a Simple Web Application**:
    - Creating a simple web application to take input doc file from the user

→

Code:

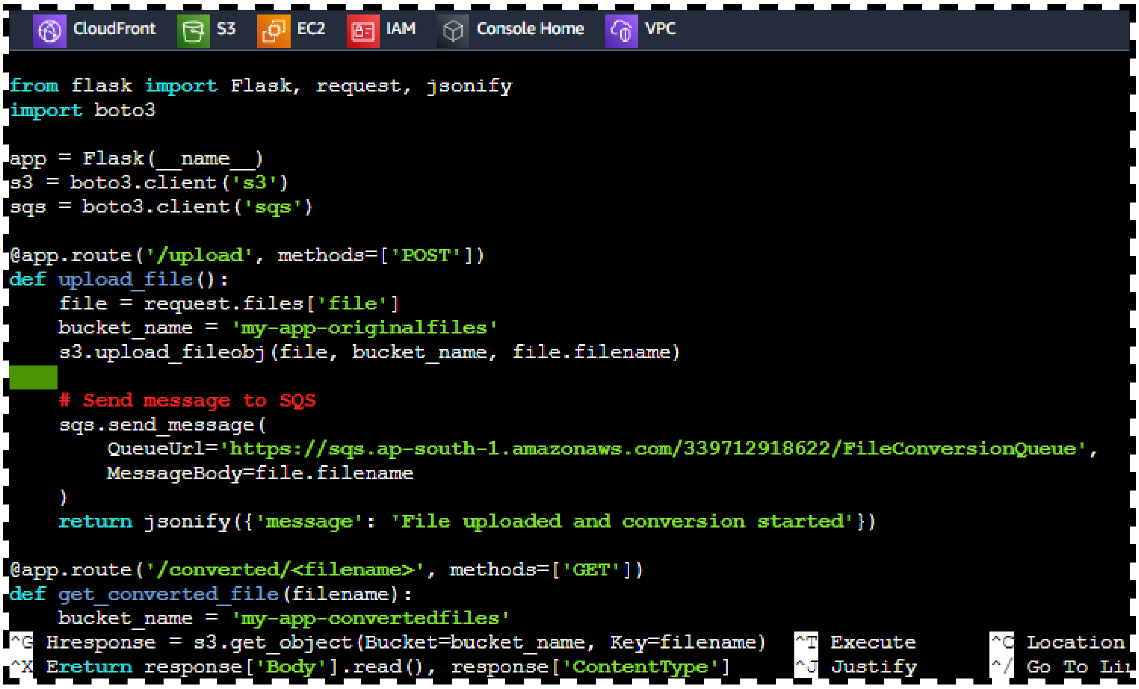
```
from flask import Flask, request, jsonify
import boto3

app = Flask(__name__)
s3 = boto3.client('s3')
sqs = boto3.client('sqs')

@app.route('/upload', methods=['POST'])
def upload_file():
    file = request.files['file']
    bucket_name = 'my-app-originalfiles'
    s3.upload_fileobj(file, bucket_name, file.filename)

    # Send message to SQS
    sqs.send_message(
        QueueUrl='https://sqs.ap-south-1.amazonaws.com/339712918622/FileConversionQueue',
        MessageBody=file.filename
    )
    return jsonify({'message': 'File uploaded and conversion started'})

@app.route('/converted/<filename>', methods=['GET'])
def get_converted_file(filename):
    bucket_name = 'my-app-convertedfiles'
    response = s3.get_object(Bucket=bucket_name, Key=filename)
    return response['Body'].read(), response['ContentType']
```

```
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=80)
```

- Running the web application

python3 your_flask_app.py

- Entering the web application code into the c2 machine



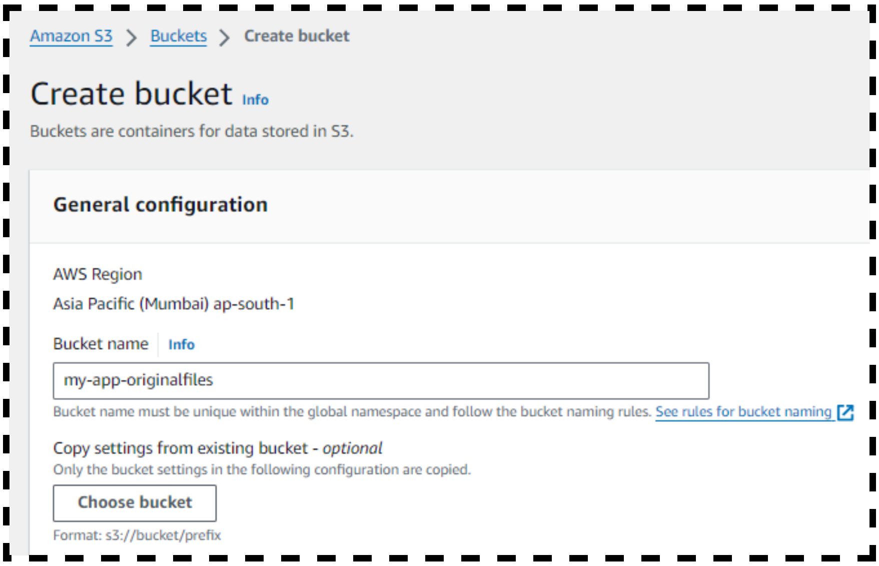
```
from flask import Flask, request, jsonify
import boto3

app = Flask(__name__)
s3 = boto3.client('s3')
sqs = boto3.client('sqs')

@app.route('/upload', methods=['POST'])
def upload_file():
    file = request.files['file']
    bucket_name = 'my-app-originalfiles'
    s3.upload_fileobj(file, bucket_name, file.filename)

    # Send message to SQS
    sqs.send_message(
        QueueUrl='https://sqs.ap-south-1.amazonaws.com/339712918622/FileConversionQueue',
        MessageBody=file.filename
    )
    return jsonify({'message': 'File uploaded and conversion started'})

@app.route('/converted/<filename>', methods=['GET'])
def get_converted_file(filename):
    bucket_name = 'my-app-convertedfiles'
^G Hresponse = s3.get_object(Bucket=bucket_name, Key=filename)  ^T Execute      ^C Location
^X Ereturn response['Body'].read(), response['ContentType']     ^J Justify      ^/ Go To Li
```

- S3 Bucket Configuration

- Create two S3 buckets:

- My-app-originalfiles : (for storing original files)

→



Amazon S3 > Buckets > Create bucket

Create bucket Info

Buckets are containers for data stored in S3.

General configuration

AWS Region

Asia Pacific (Mumbai) ap-south-1

Bucket name Info

my-app-originalfiles

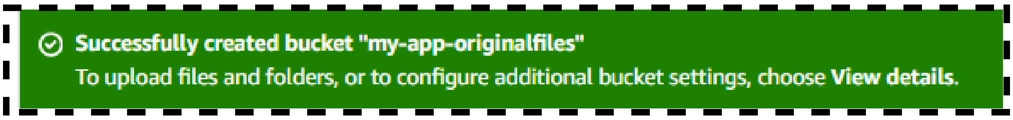
Bucket name must be unique within the global namespace and follow the bucket naming rules. See rules for bucket naming

Copy settings from existing bucket - *optional*

Only the bucket settings in the following configuration are copied.

Choose bucket

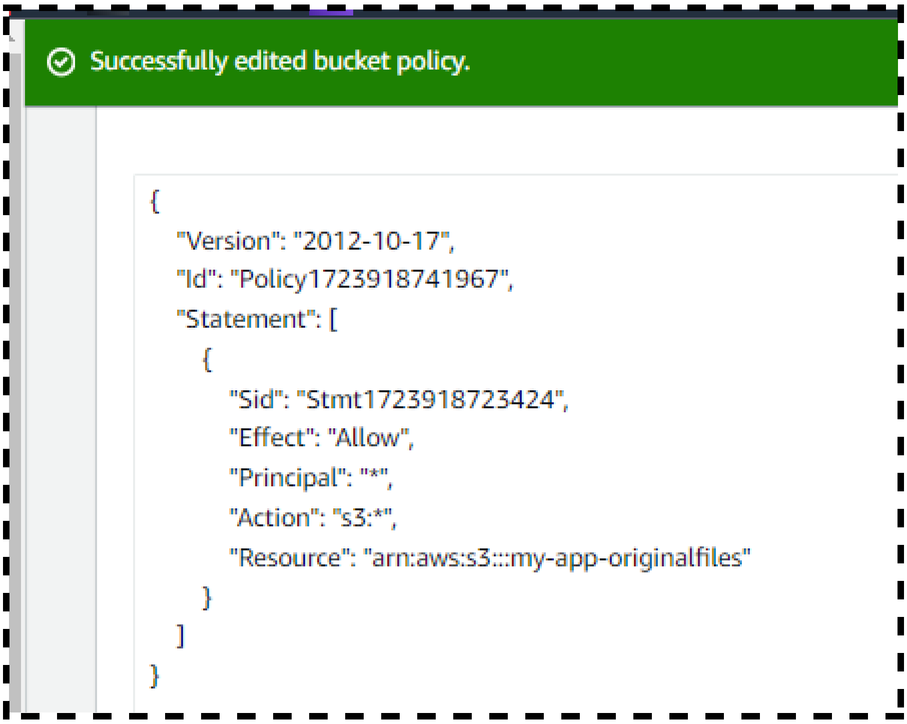
Format: s3://bucket/prefix



Successfully created bucket "my-app-originalfiles"

To upload files and folders, or to configure additional bucket settings, choose **View details**.

- Add Policy

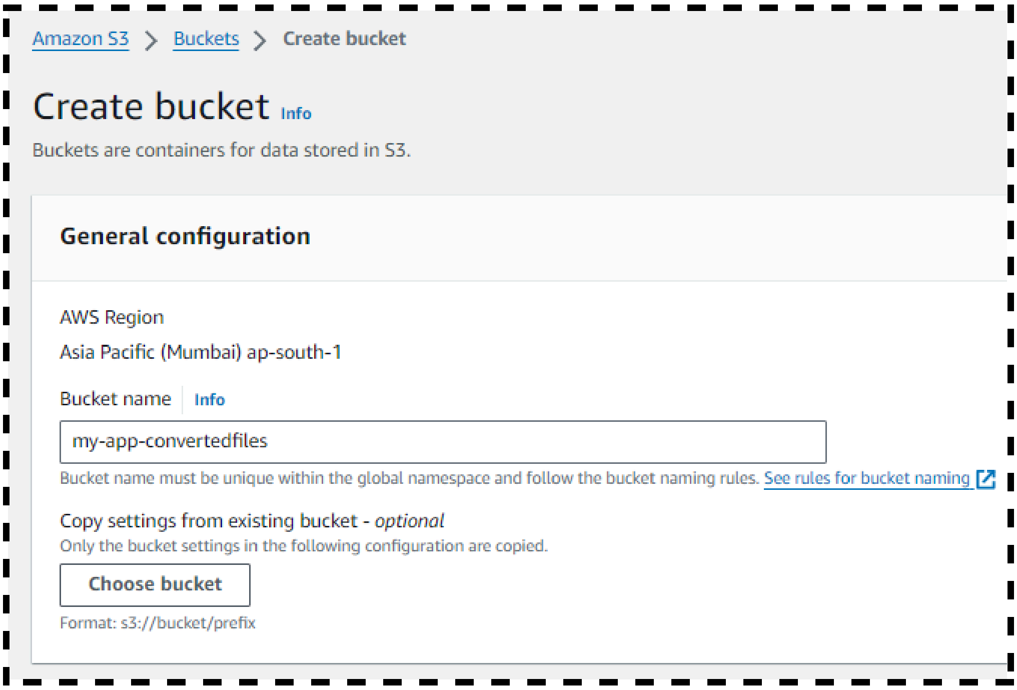


Successfully edited bucket policy.

```
{
  "Version": "2012-10-17",
  "Id": "Policy1723918741967",
  "Statement": [
    {
      "Sid": "Stmt1723918723424",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:*",
      "Resource": "arn:aws:s3:::my-app-originalfiles"
    }
  ]
}
```

- My-app-convertedfiles :(for storing converted files)

→



Create bucket Info

Buckets are containers for data stored in S3.

General configuration

AWS Region

Asia Pacific (Mumbai) ap-south-1

Bucket name | Info

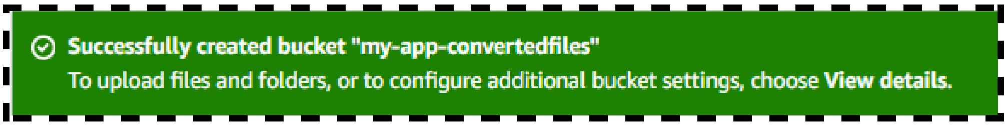
my-app-convertedfiles

Bucket name must be unique within the global namespace and follow the bucket naming rules. See rules for bucket naming

Copy settings from existing bucket - *optional*

Only the bucket settings in the following configuration are copied.

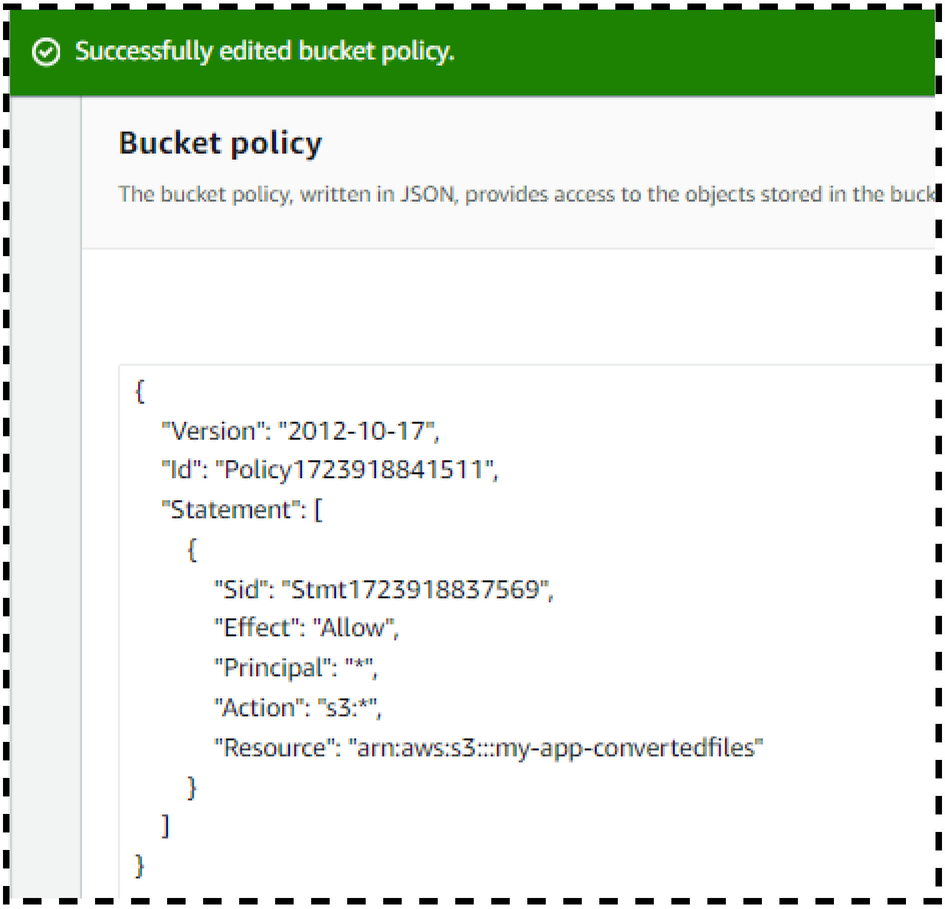
Format: s3://bucket/prefix



Successfully created bucket "my-app-convertedfiles"

To upload files and folders, or to configure additional bucket settings, choose **View details.**

- Add Policy:



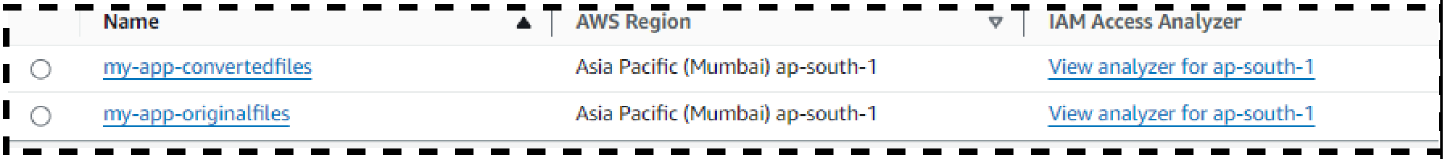
Successfully edited bucket policy.

Bucket policy

The bucket policy, written in JSON, provides access to the objects stored in the buck

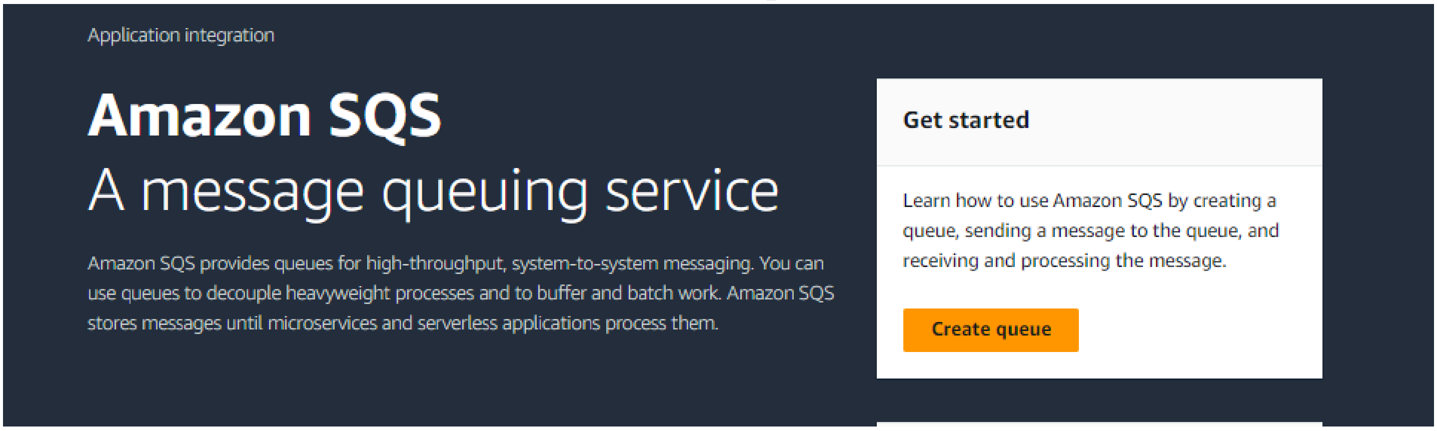
```
{
  "Version": "2012-10-17",
  "Id": "Policy1723918841511",
  "Statement": [
    {
      "Sid": "Stmt1723918837569",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:*",
      "Resource": "arn:aws:s3:::my-app-convertedfiles"
    }
  ]
}
```

- Two Buckets Are created successfully



	Name ▲	AWS Region ▽	IAM Access Analyzer
○	my-app-convertedfiles	Asia Pacific (Mumbai) ap-south-1	View analyzer for ap-south-1
○	my-app-originalfiles	Asia Pacific (Mumbai) ap-south-1	View analyzer for ap-south-1

- Sqs



Application integration

Amazon SQS

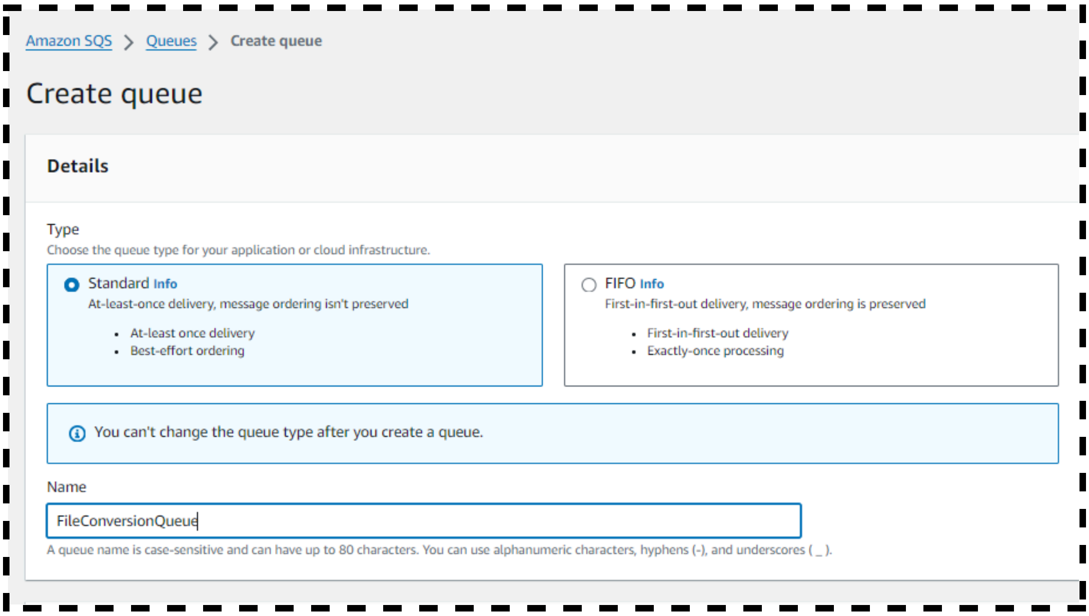
A message queuing service

Amazon SQS provides queues for high-throughput, system-to-system messaging. You can use queues to decouple heavyweight processes and to buffer and batch work. Amazon SQS stores messages until microservices and serverless applications process them.

Get started

Learn how to use Amazon SQS by creating a queue, sending a message to the queue, and receiving and processing the message.

- Creating the sqs service



Amazon SQS > Queues > Create queue

Create queue

Details

Type

Choose the queue type for your application or cloud infrastructure.

● Standard Info
At-least-once delivery, message ordering isn't preserved
- At-least once delivery
- Best-effort ordering

○ FIFO Info
First-in-first-out delivery, message ordering is preserved
- First-in-first-out delivery
- Exactly-once processing

ⓘ You can't change the queue type after you create a queue.

Name

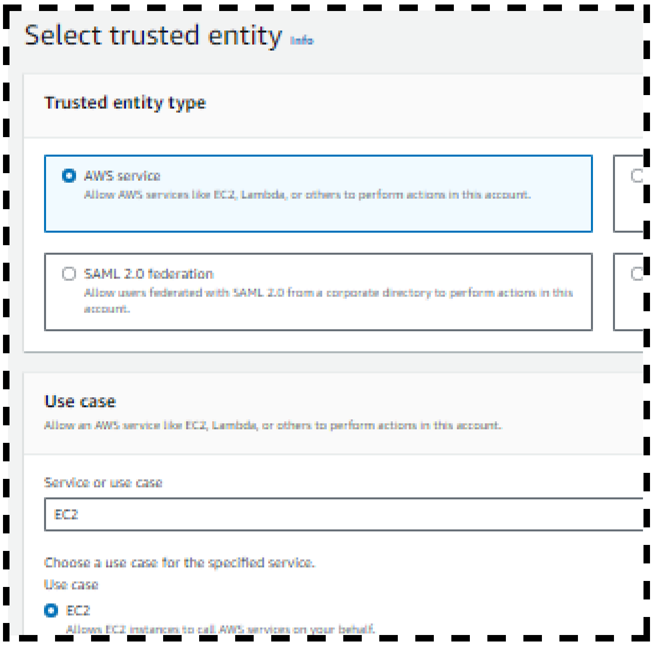
FileConversionQueue

A queue name is case-sensitive and can have up to 80 characters. You can use alphanumeric characters, hyphens (-), and underscores (_).

- Link: https://sqs.ap-south-1.amazonaws.com/339712918622/FileConversionQueue

- I am Roles

- Creating the iam role for ec2 to provide full access of sqs , s3 bucket , lambda function to ec2.



Select trusted entity Info

Trusted entity type

● AWS service
Allow AWS services like EC2, Lambda, or others to perform actions in this account.

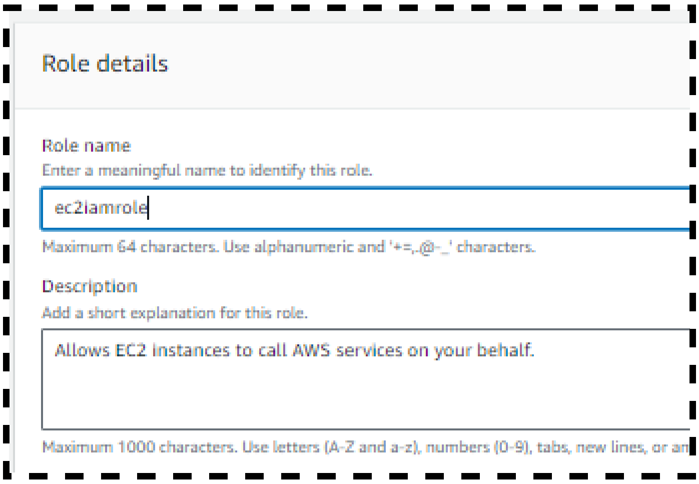
○ SAML 2.0 federation
Allow users federated with SAML 2.0 from a corporate directory to perform actions in this account.

Use case
Allow an AWS service like EC2, Lambda, or others to perform actions in this account.

Service or use case
EC2

Choose a use case for the specified service.
Use case
● EC2
Allows EC2 instances to call AWS services on your behalf.

- Set the role name



Role details

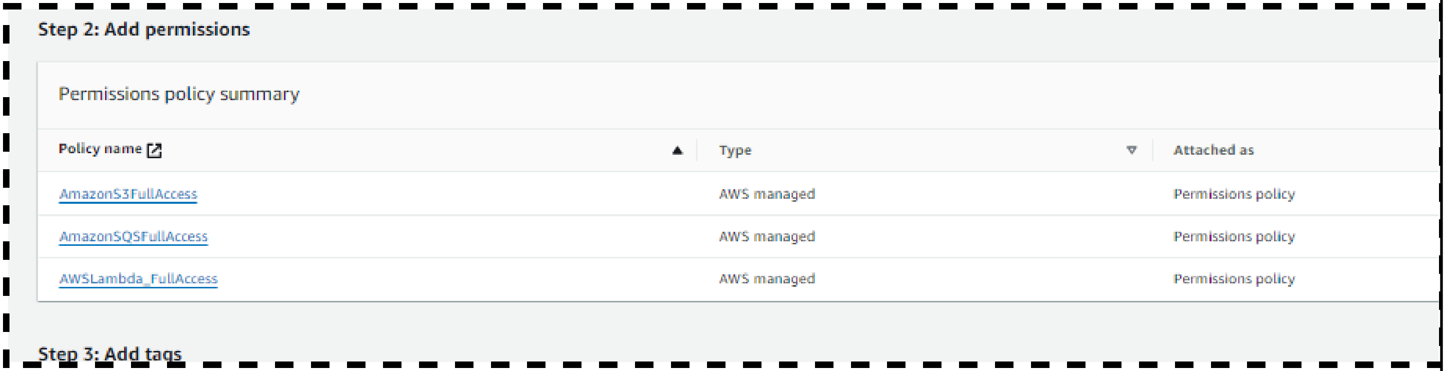
Role name
Enter a meaningful name to identify this role.
ec2iamrole

Maximum 64 characters. Use alphanumeric and '+=,.@-_' characters.

Description
Add a short explanation for this role.
Allows EC2 instances to call AWS services on your behalf.

Maximum 1000 characters. Use letters (A-Z and a-z), numbers (0-9), tabs, new lines, or an

- The access given



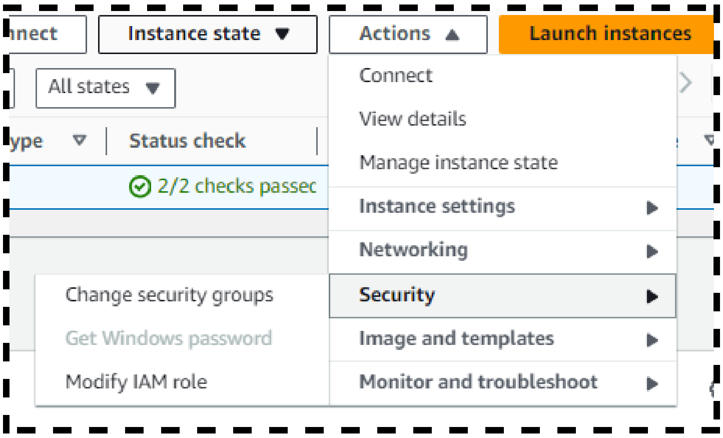
Step 2: Add permissions

Permissions policy summary

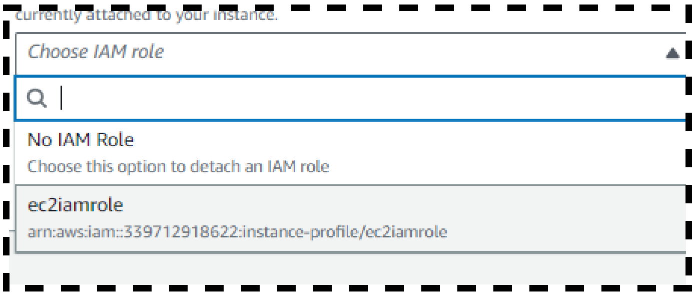
Policy name ▲	Type ▽	Attached as
AmazonS3FullAccess	AWS managed	Permissions policy
AmazonSQSFullAccess	AWS managed	Permissions policy
AWSLambda_FullAccess	AWS managed	Permissions policy

Step 3: Add tags

- Attaching the i am role to the ec2



- Choose the i am role that we made earlier

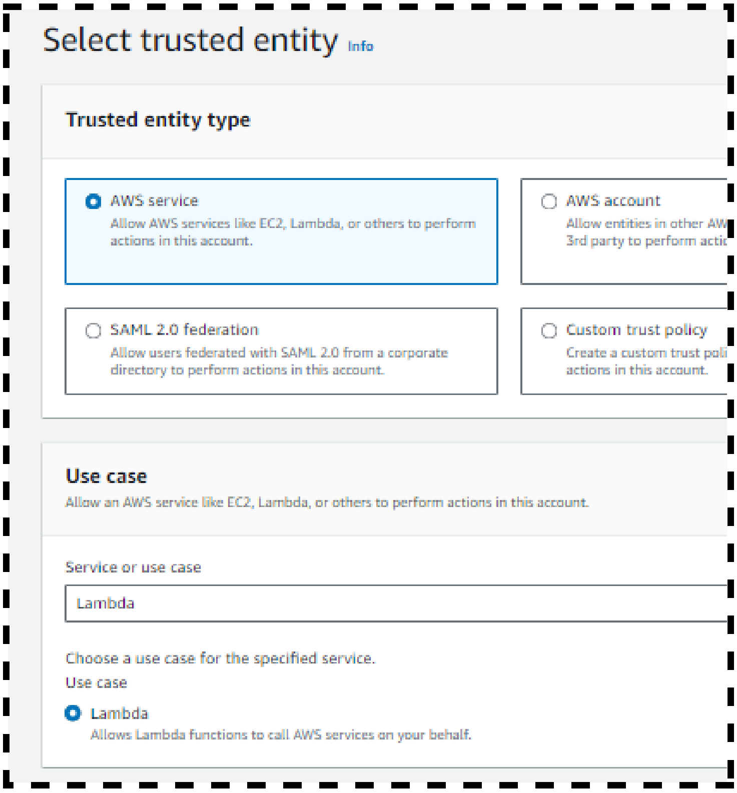


No IAM Role
Choose this option to detach an IAM role

ec2iamrole
arn:aws:iam::339712918622:instance-profile/ec2iamrole

- Lambda

- Creating the role for lambda function



Select trusted entity Info

Trusted entity type

● AWS service
Allow AWS services like EC2, Lambda, or others to perform actions in this account.

○ AWS account
Allow entities in other AW 3rd party to perform actic

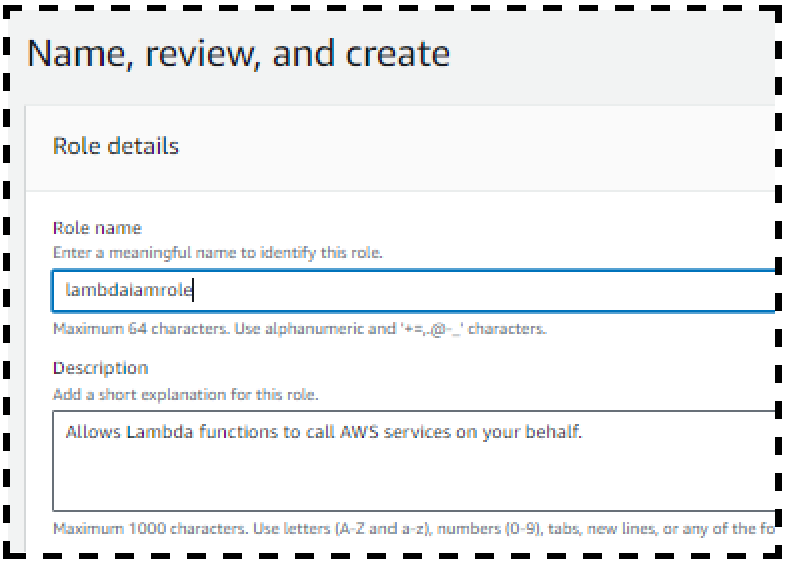
○ SAML 2.0 federation
Allow users federated with SAML 2.0 from a corporate directory to perform actions in this account.

○ Custom trust policy
Create a custom trust poli actions in this account.

Use case
Allow an AWS service like EC2, Lambda, or others to perform actions in this account.

Service or use case
Lambda

Choose a use case for the specified service.
Use case
● Lambda
Allows Lambda functions to call AWS services on your behalf.



Name, review, and create

Role details

Role name

Enter a meaningful name to identify this role.

lambdaiamrole

Maximum 64 characters. Use alphanumeric and '+=,.@-_' characters.

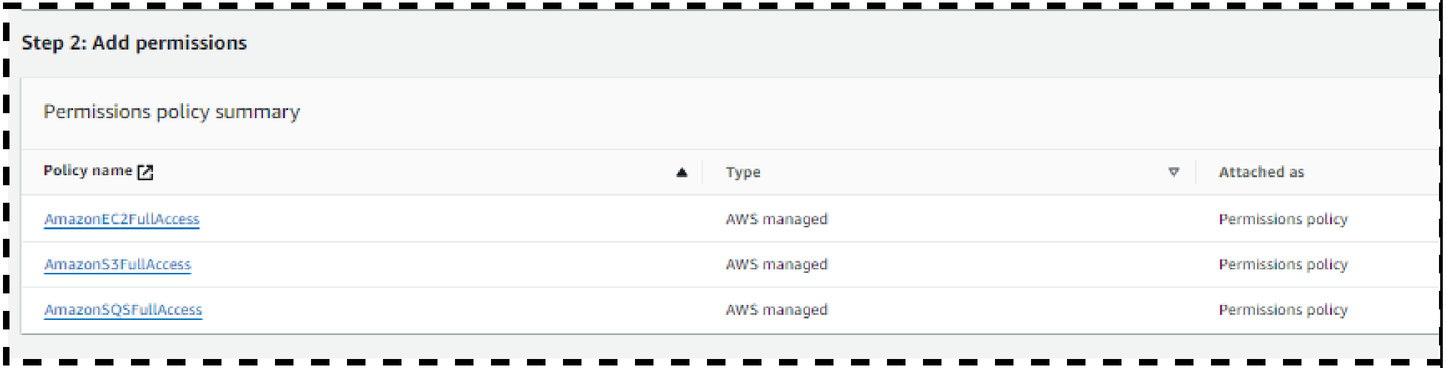
Description

Add a short explanation for this role.

Allows Lambda functions to call AWS services on your behalf.

Maximum 1000 characters. Use letters (A-Z and a-z), numbers (0-9), tabs, new lines, or any of the fo

- Provide following access to the lambda function

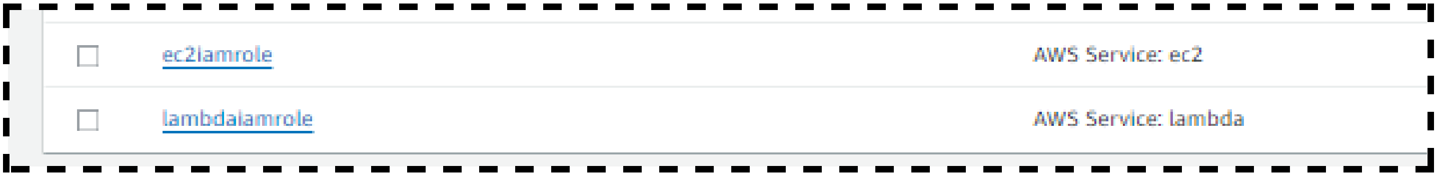


Step 2: Add permissions

Permissions policy summary

Policy name	Type	Attached as
AmazonEC2FullAccess	AWS managed	Permissions policy
AmazonS3FullAccess	AWS managed	Permissions policy
AmazonSQSFullAccess	AWS managed	Permissions policy

- Both the roles are created successfully



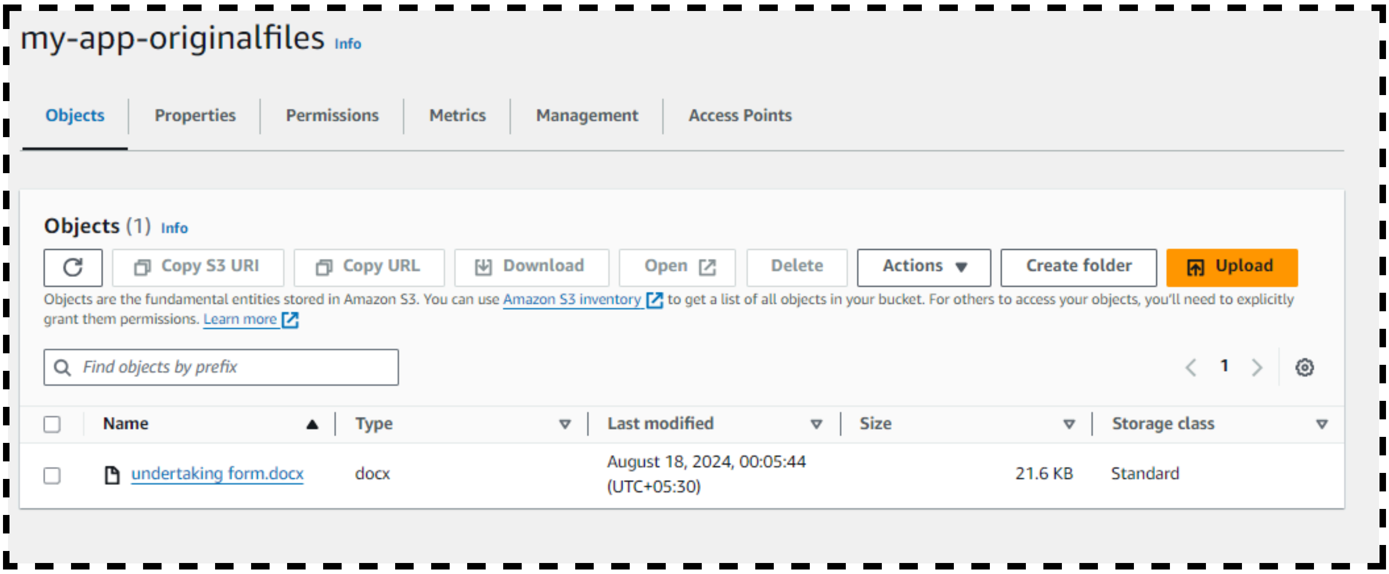
□	ec2iamrole	AWS Service: ec2
□	lambdaiamrole	AWS Service: lambda

- Updated working code

```
import boto3
import os
from flask import Flask, request, jsonify, send_from_directory
import logging

app = Flask(__name__)
logging.basicConfig(level=logging.DEBUG)

AWS_REGION = os.getenv('AWS_REGION', 'ap-south-1')

s3 = boto3.client('s3', region_name=AWS_REGION)
sqs = boto3.client('sqs', region_name=AWS_REGION)

ORIGINAL_BUCKET = 'my-app-originalfiles'
CONVERTED_BUCKET = 'my-app-convertedfiles'
SQS_QUEUE_URL =
'https://sqs.ap-south-1.amazonaws.com/339712918622/FileConversionQueue'

@app.route('/')
def index():
    return '''
    <h1>Upload File</h1>
    <form action="/upload" method="post" enctype="multipart/form-data">
        <input type="file" name="file">
        <input type="submit" value="Upload">
    </form>
    '''

@app.route('/upload', methods=['POST'])
```

```
def upload_file():
    try:
        if 'file' not in request.files:
            return jsonify({'error': 'No file part'}), 400
        file = request.files['file']
        if file.filename == '':
            return jsonify({'error': 'No selected file'}), 400
        if file:
            s3.upload_fileobj(file, ORIGINAL_BUCKET, file.filename)
            sqs.send_message(
                QueueUrl=SQS_QUEUE_URL,
                MessageBody=file.filename
            )
            return jsonify({'message': 'File uploaded and conversion started'})
        return jsonify({'error': 'File upload failed'}), 500
    except Exception as e:
        logging.error(f'Error in upload_file: {e}')
        return jsonify({'error': str(e)}), 500

@app.route('/converted/<filename>', methods=['GET'])
def get_converted_file(filename):
    try:
        s3.download_file(CONVERTED_BUCKET, filename, '/tmp/' + filename)
        return send_from_directory('/tmp', filename)
    except Exception as e:
        logging.error(f'Error in get_converted_file: {e}')
        return jsonify({'error': str(e)}), 500

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=80, debug=True)
```

- Uploading process
- Uploaded docx file is saved into the Original bucket



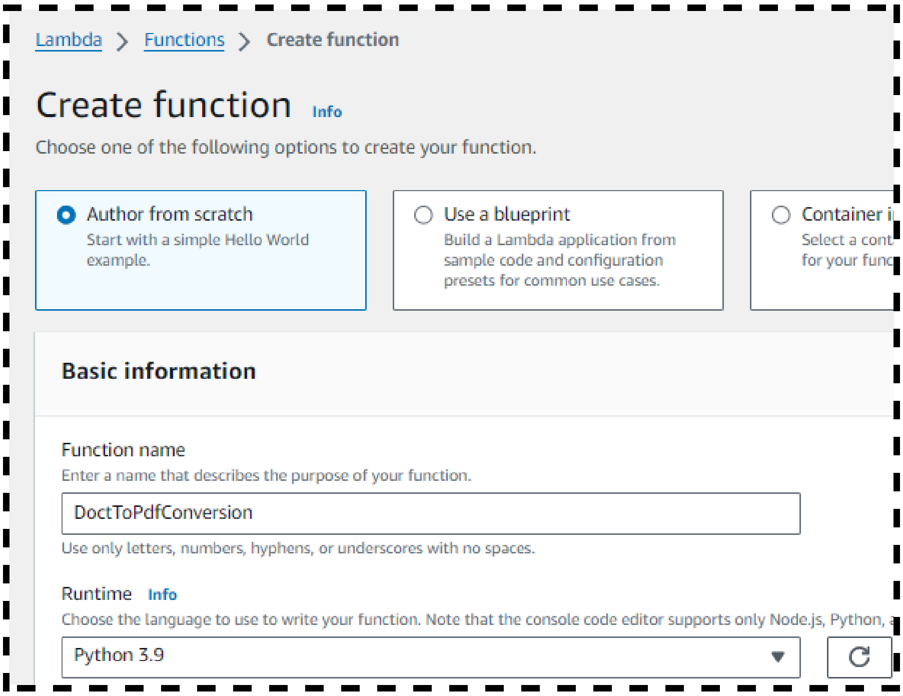
my-app-originalfiles Info

Objects (1) Info

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

□	Name	Type	Last modified	Size	Storage class
□	undertaking form.docx	docx	August 18, 2024, 00:05:44 (UTC+05:30)	21.6 KB	Standard

- Lambda Function



Lambda > Functions > Create function

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.

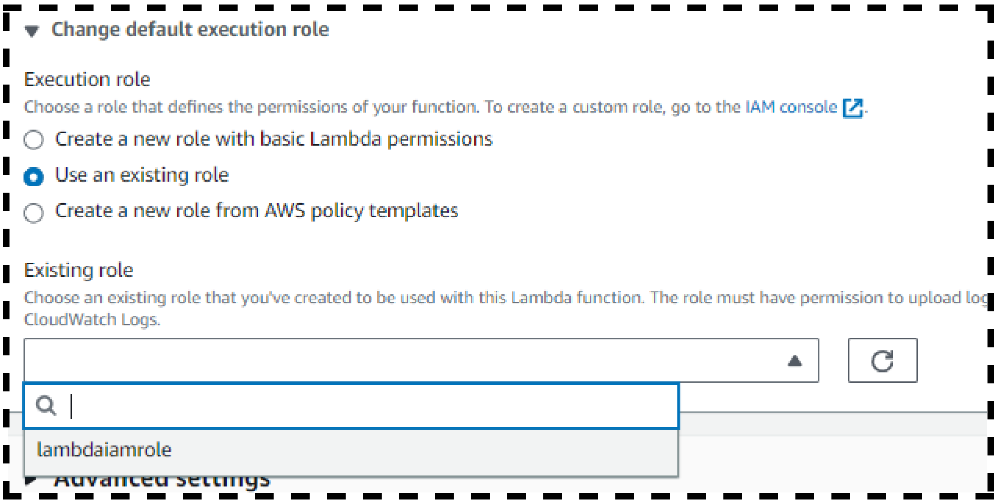
○ Container i
Select a cont
for your func

Basic information

Function name
Enter a name that describes the purpose of your function.
DoctToPdfConversion
Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime Info
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, a
Python 3.9

- Choose the Existing role that we created earlier



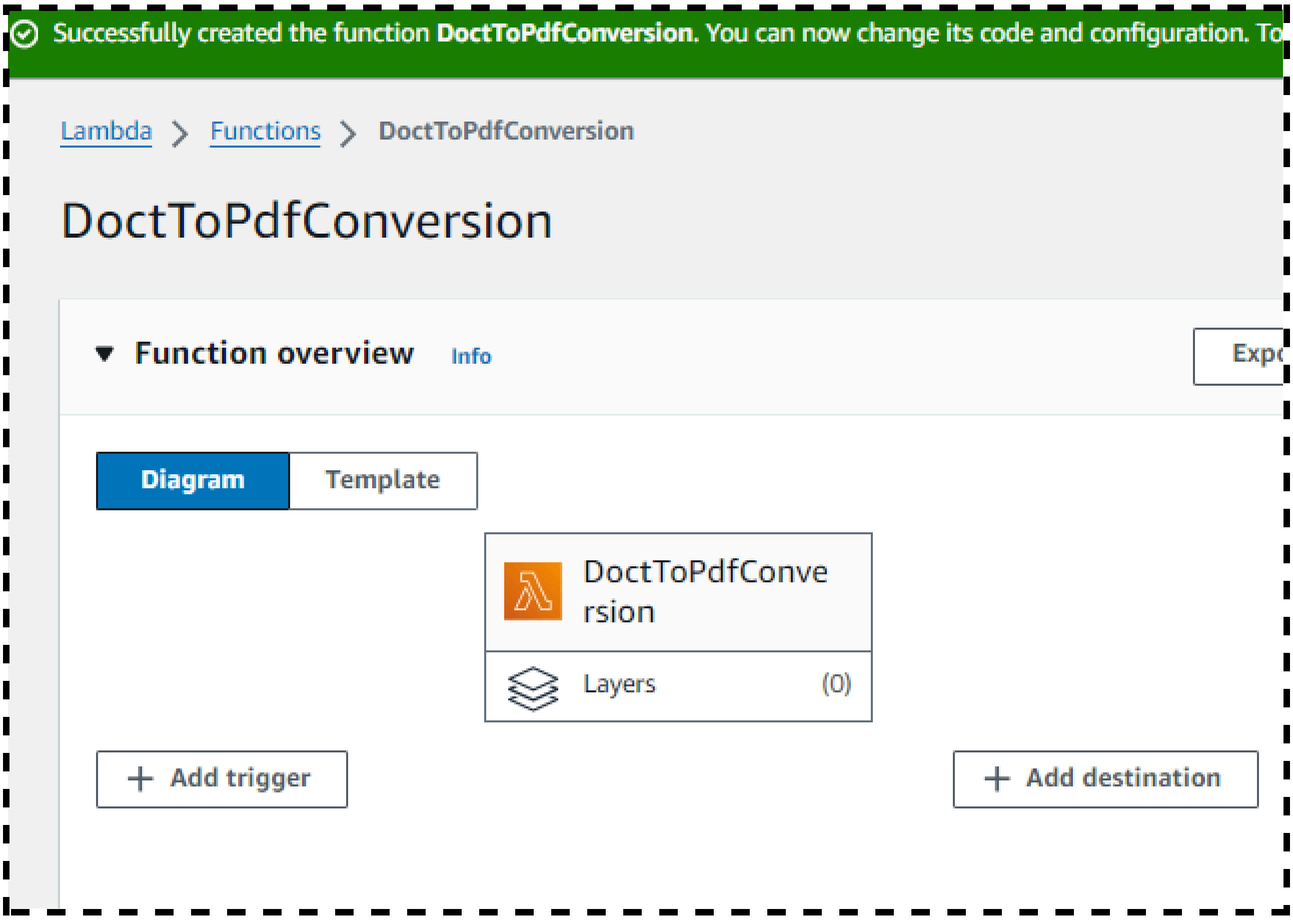
Change default execution role

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

Existing role
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload log
CloudWatch Logs.

lambdaiamrole

Advanced settings



Successfully created the function **DoctToPdfConversion**. You can now change its code and configuration. To

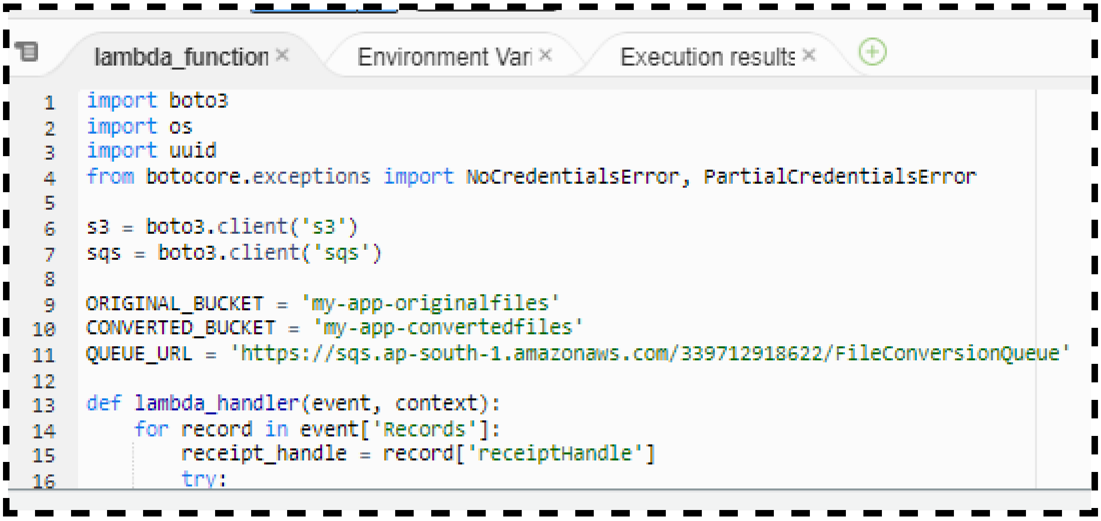
DoctToPdfConversion

- Adding the code and testing it

- code

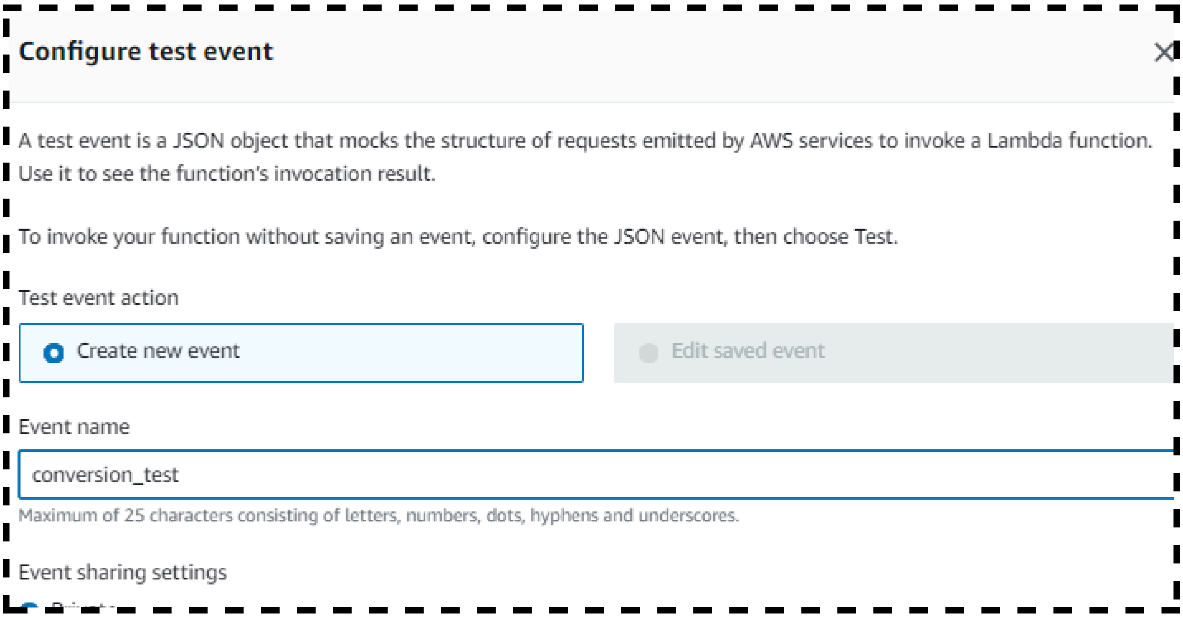
```
import boto3
import os
import uuid
from botocore.exceptions import NoCredentialsError, PartialCredentialsError

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

ORIGINAL_BUCKET = 'my-app-originalfiles'
CONVERTED_BUCKET = 'my-app-convertedfiles'
QUEUE_URL = 'https://sqs.ap-south-1.amazonaws.com/339712918622/FileConversionQueue'

def lambda_handler(event, context):
    for record in event['Records']:
        receipt_handle = record['receiptHandle']
        try:
            # Get the object from the S3 bucket
            file_key = record['body']
            download_path = f'/tmp/{uuid.uuid4()}_{file_key}'
            s3.download_file(ORIGINAL_BUCKET, file_key, download_path)
            # Perform the document conversion (example: converting .docx to .pdf)
            converted_path = convert_document(download_path)

            # Upload the converted file back to S3
            converted_key = f'converted/{os.path.basename(converted_path)}'
            s3.upload_file(converted_path, CONVERTED_BUCKET, converted_key)

            # Delete the message from the queue
            sqs.delete_message(QueueUrl=QUEUE_URL, ReceiptHandle=receipt_handle)

        except NoCredentialsError:
            print("Error: Credentials not available")

        except PartialCredentialsError:
            print("Error: Incomplete credentials")

        except Exception as e:
            print(f"Error processing {file_key}: {str(e)}")


def convert_document(input_path):
    # Example conversion logic
    output_path = input_path.replace('.docx', '.pdf')
    # Use a library like python-docx or other to perform actual conversion
    # Here we simply rename the file for demonstration
    os.rename(input_path, output_path)
    return output_path
```

- Adding the code:



```
import boto3
import os
import uuid
from botocore.exceptions import NoCredentialsError, PartialCredentialsError

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

ORIGINAL_BUCKET = 'my-app-originalfiles'
CONVERTED_BUCKET = 'my-app-convertedfiles'
QUEUE_URL = 'https://sqs.ap-south-1.amazonaws.com/339712918622/FileConversionQueue'

def lambda_handler(event, context):
    for record in event['Records']:
        receipt_handle = record['receiptHandle']
        try:
```

- Testing the code



Configure test event

A test event is a JSON object that mocks the structure of requests emitted by AWS services to invoke a Lambda function. Use it to see the function's invocation result.

To invoke your function without saving an event, configure the JSON event, then choose Test.

Test event action

● Create new event   ○ Edit saved event

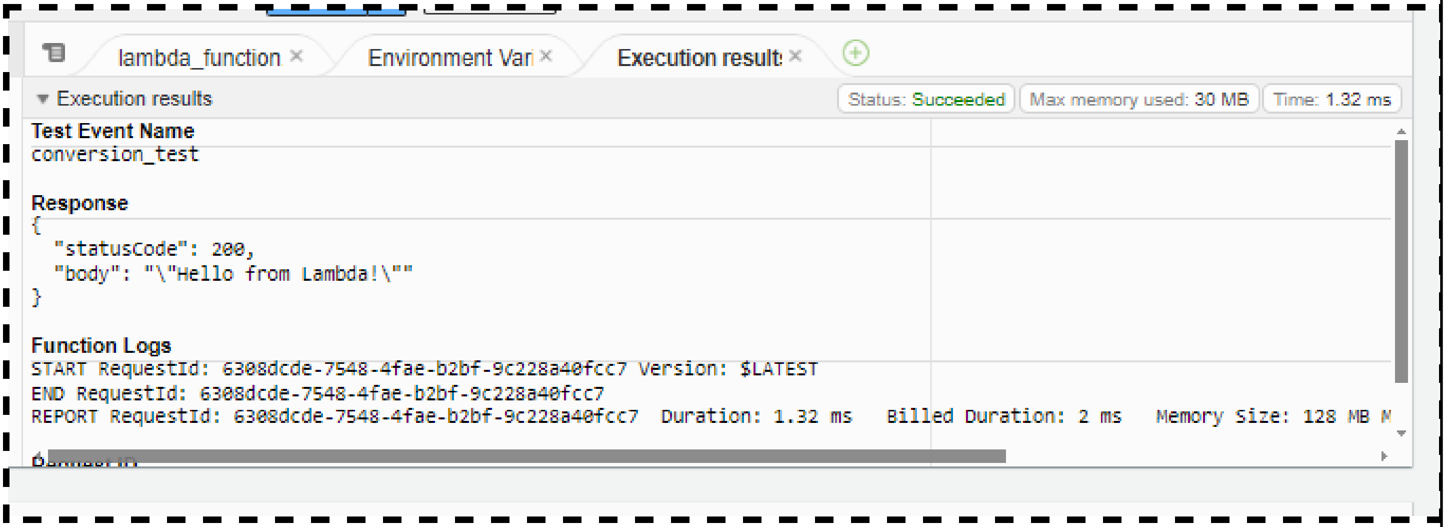
Event name

conversion_test

Maximum of 25 characters consisting of letters, numbers, dots, hyphens and underscores.

Event sharing settings

- Test is succeeded



Execution results   Status: Succeeded   Max memory used: 30 MB   Time: 1.32 ms

Test Event Name

```
conversion_test
```

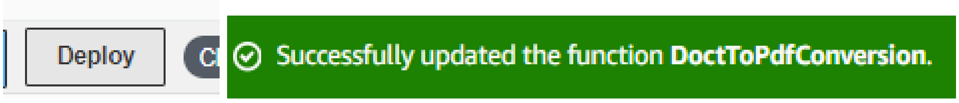
Response

```
{
  "statusCode": 200,
  "body": "\"Hello from Lambda!\""
}
```

Function Logs

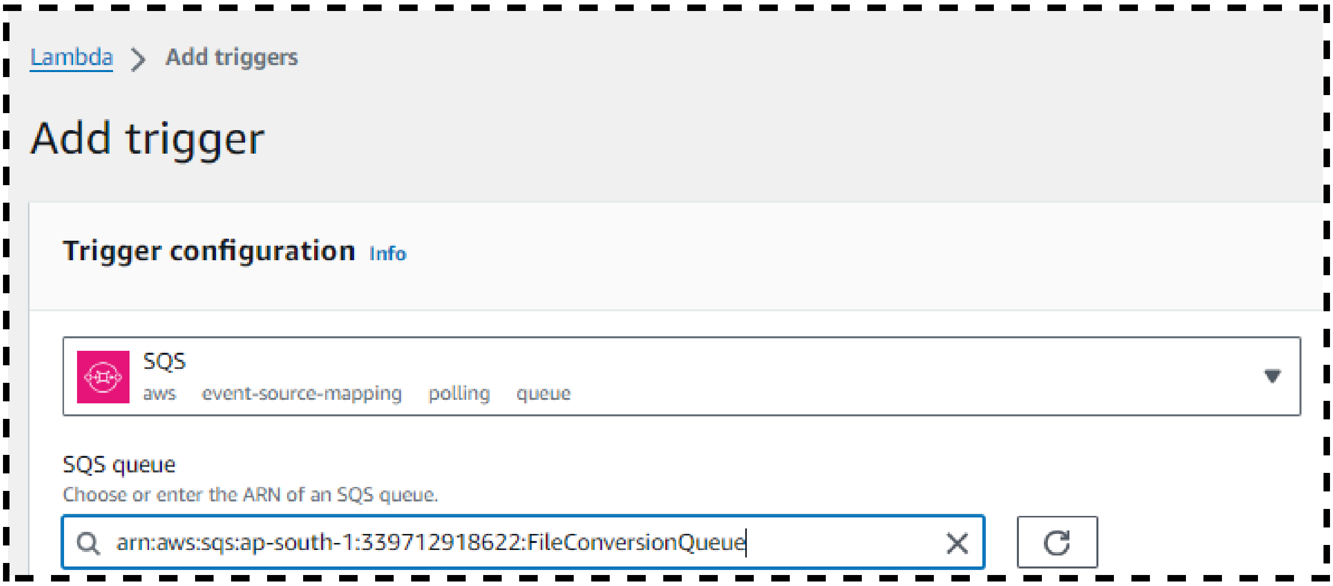
```
START RequestId: 6308dcde-7548-4fae-b2bf-9c228a40fcc7 Version: $LATEST
END RequestId: 6308dcde-7548-4fae-b2bf-9c228a40fcc7
REPORT RequestId: 6308dcde-7548-4fae-b2bf-9c228a40fcc7  Duration: 1.32 ms   Billed Duration: 2 ms   Memory Size: 128 MB M
```

- Deploying the code0→Click on deploy



Successfully updated the function DoctToPdfConversion.

- After the successful deployment of the code
- Add the trigger that is sqs trigger
- And select the existing created sqs



Lambda > Add triggers

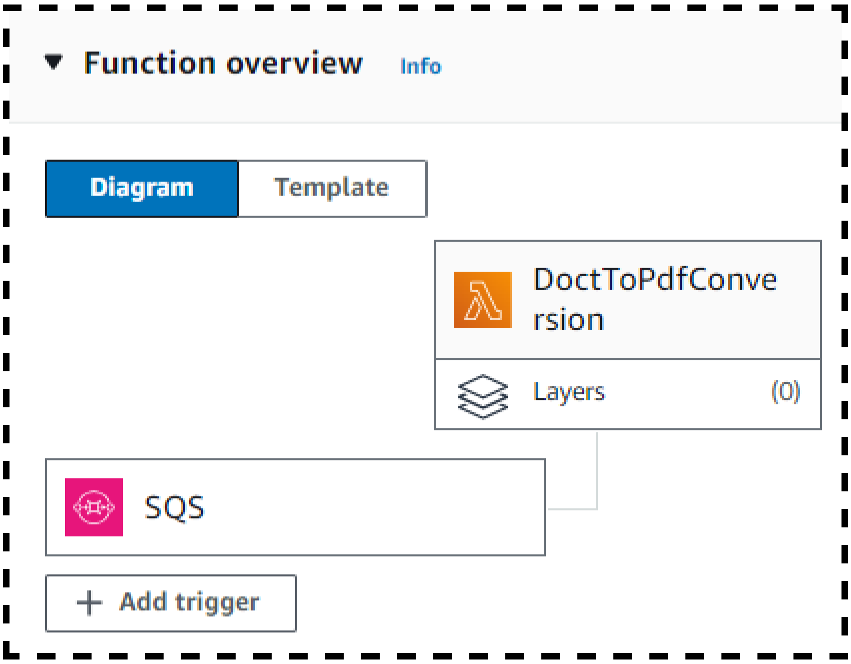
Add trigger

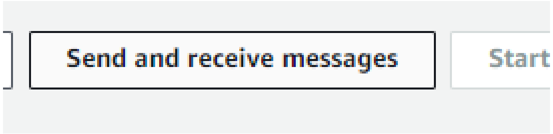
Trigger configuration Info

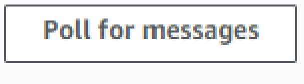
SQS
aws event-source-mapping polling queue

SQS queue
Choose or enter the ARN of an SQS queue.
arn:aws:sqs:ap-south-1:339712918622:FileConversionQueue

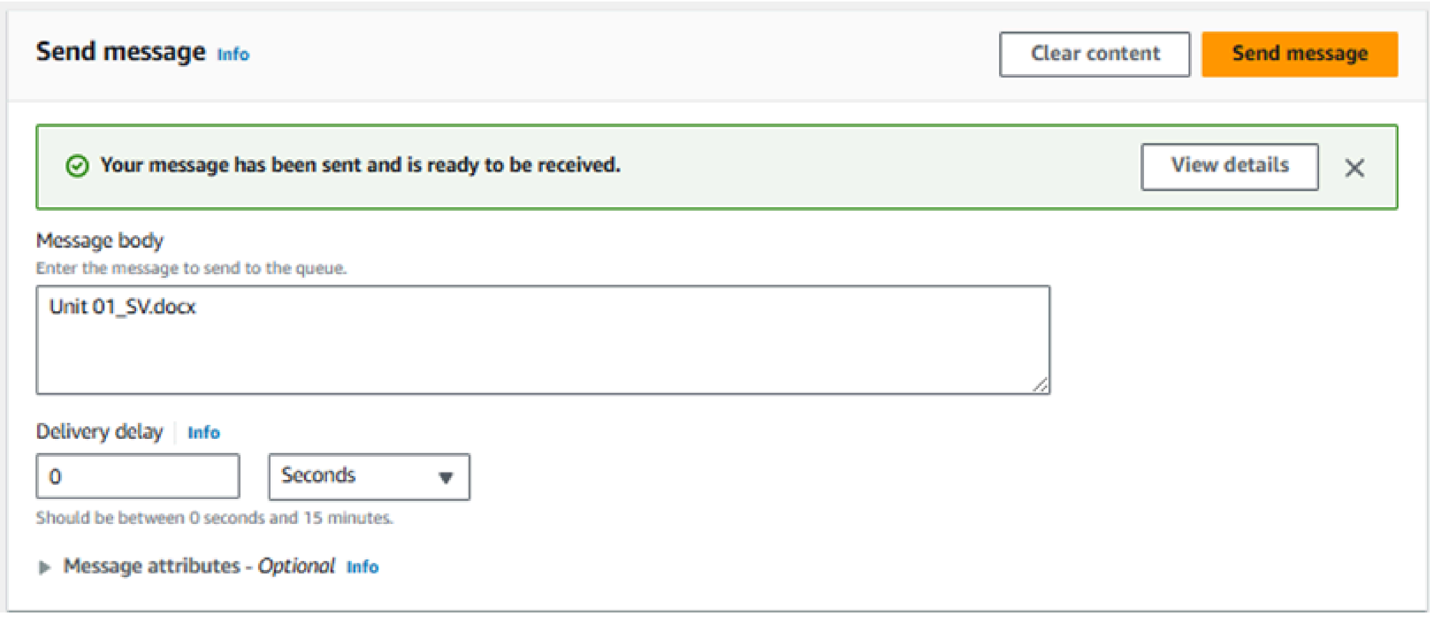
- Trigger added successfully



- To fetch the document from the original bucket and to covert it from lambda function we have to pull the document from the first s3 original bucket
- So go to sqs→  → to pull the file



- Copy the name of the and semd the message



Your message has been sent and is ready to be received.

Message body

Enter the message to send to the queue.

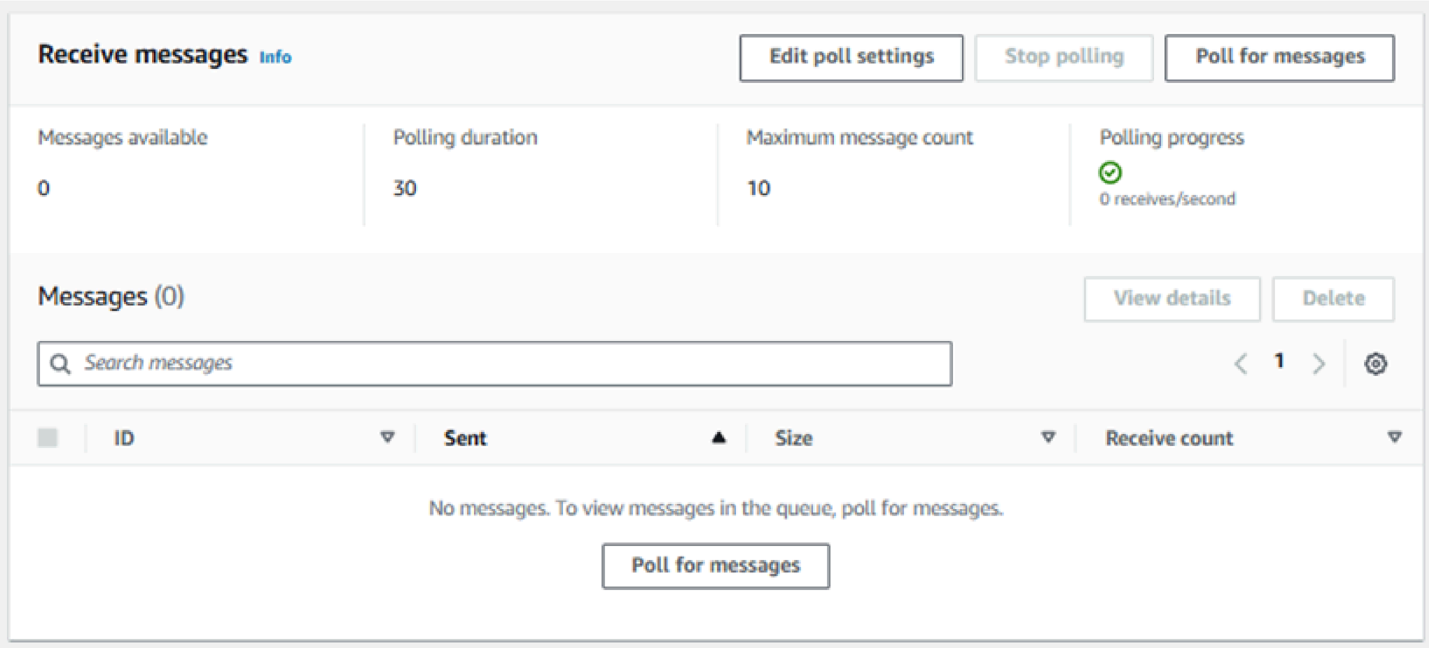
Unit 01_SV.docx

Delivery delay: 0 Seconds

Should be between 0 seconds and 15 minutes.

▸ Message attributes - *Optional*

- In Receive Message Click on Poll for Messages.



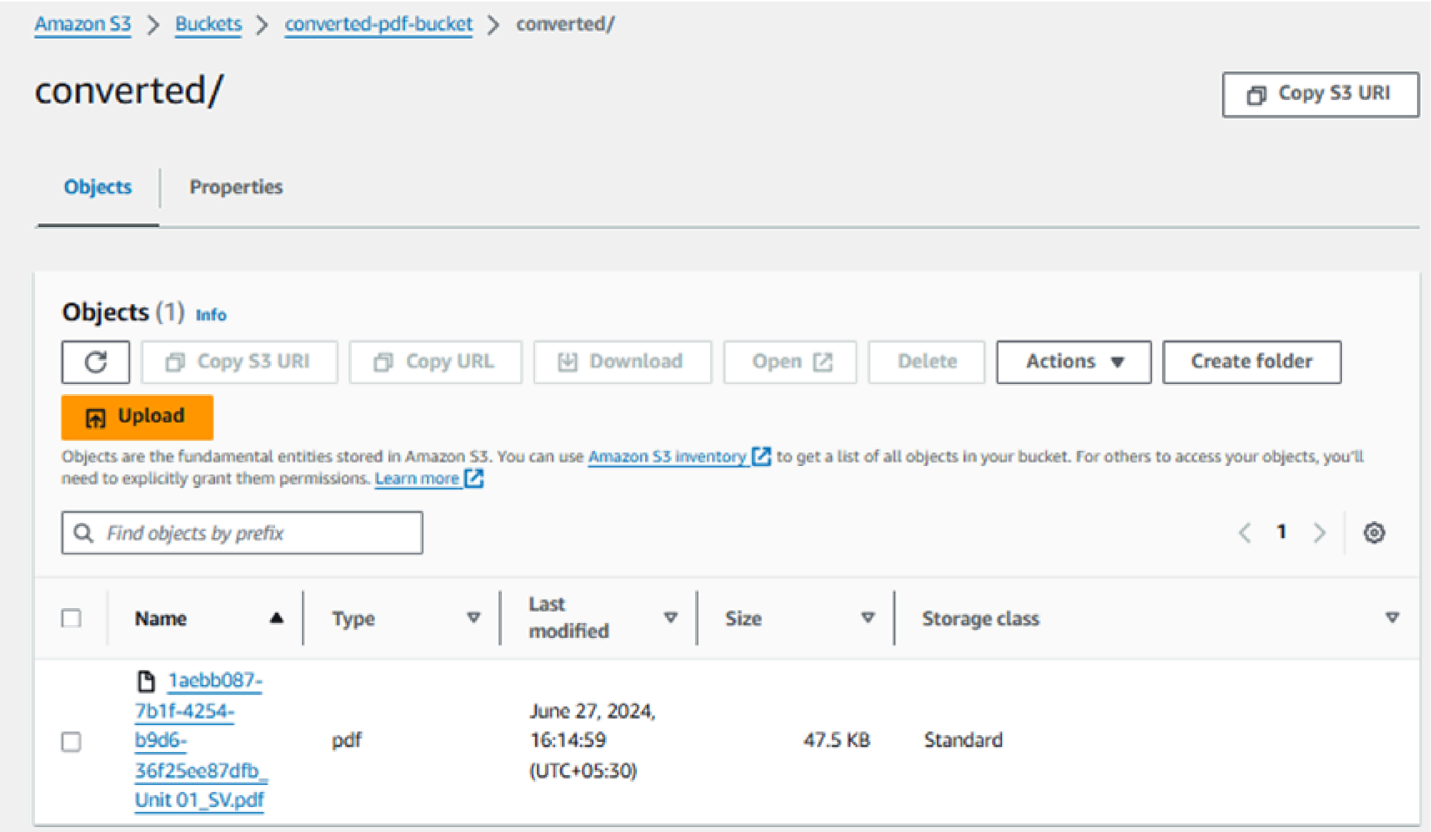
Messages available	Polling duration	Maximum message count	Polling progress
0	30	10	✓ 0 receives/second

Messages (0)

□	ID	Sent	Size	Receive count

No messages. To view messages in the queue, poll for messages.

- Then Go the the Destination bucket and check if the .docx file is converted into .pdf file.



Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

□	Name	Type	Last modified	Size	Storage class
□	1aebb087-7b1f-4254-b9d6-36f25ee87dfb_Unit 01_SV.pdf	pdf	June 27, 2024, 16:14:59 (UTC+05:30)	47.5 KB	Standard