

# AWS Assignment

- Login to AWS CLI using an access key.

## Access key best practices & alternatives Info

Avoid using long-term credentials like access keys to improve your security. Consider the following use cases and alternatives.

### Use case

**Command Line Interface (CLI)**

You plan to use this access key to enable the AWS CLI to access your AWS account.

**Local code**

You plan to use this access key to enable application code in a local development environment to access your AWS account.

**Application running on an AWS compute service**

You plan to use this access key to enable application code running on an AWS compute service like Amazon EC2, Amazon ECS, or AWS Lambda to access your AWS account.

**Third-party service**

You plan to use this access key to enable access for a third-party application or service that monitors or manages your AWS resources.

**Application running outside AWS**

You plan to use this access key to authenticate workloads running in your data center or other infrastructure outside of AWS that needs to access your AWS resources.

**Other**

Your use case is not listed here.

## Set description tag - *optional* Info

The description for this access key will be attached to this user as a tag and shown alongside the access key.

### Description tag value

Describe the purpose of this access key and where it will be used. A good description will help you rotate this access key confidently later.

Maximum 256 characters. Allowed characters are letters, numbers, spaces representable in UTF-8, and: \_ . : / = + - @

[Cancel](#)[Previous](#)[Create access key](#)

- Enter the user Access ID and Secret Access key

```
AWS Access Key ID [None]: AKIAQ3EGRH55WRAKSXOV
AWS Secret Access Key [None]: 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
Default region name [None]: us-east-1
Default output format [None]: json
```

- Create a role with trust policy to use EC2 instance
  - Role name - devProjectRole
- A. Create a S3 bucket in aws cli
    - Create a IAM role with s3 full access policy
    - Policy created
    - Attach the policy to the role

```
--policy-name S3FullAccessPolicy \
--policy-document file://s3FullAccessPolicy.json

{

  "Policy": {
    "PolicyName": "S3FullAccessPolicy",
    "PolicyId": "ANPAQ3EGRH557MQ4NK3QS",
    "Arn": "arn:aws:iam::058264207227:policy/S3FullAccessPolicy",
    "Path": "/",
    "DefaultVersionId": "v1",
    "AttachmentCount": 0,
    "PermissionsBoundaryUsageCount": 0,
    "IsAttachable": true,
    "CreateDate": "2024-05-03T06:54:22Z",
    "UpdateDate": "2024-05-03T06:54:22Z"
  }
}
```

## 1. B. Create a EC2 instance with the above role

- Create Instance profile
- Create EC2 instance that will be used throughout the project
- Pass the appropriate Amazon Machine Image, key Name and other required credentials.

```
{
  "InstanceProfile": {
    "Path": "/",
    "InstanceProfileName": "MyInstanceProfile",
    "InstanceProfileId": "AIPAQ3EGRH55YXAMGPBCX",
    "Arn": "arn:aws:iam::058264207227:instance-profile/MyInstanceProfile",
    "CreateDate": "2024-05-03T07:20:51Z",
    "Roles": []
  }
}
```

```

--image-id ami-07caf09b362be10b8 \
--instance-type t2.micro \
--key-name projectKey \
--iam-instance-profile Arn=arn:aws:iam::058264207227:instance-profile/MyInstanceProfile;

{
    "Groups": [],
    "Instances": [
        {
            "AmiLaunchIndex": 0,
            "ImageId": "ami-07caf09b362be10b8",
            "InstanceId": "i-0ab82a34977e11579",
            "InstanceType": "t2.micro",
            "KeyName": "projectKey",
            "LaunchTime": "2024-05-03T07:21:11.000Z",
            "Monitoring": {
                "State": "disabled"
            },
            "Placement": {
                "AvailabilityZone": "us-east-1b",
                "GroupName": ""
            },
            "VirtualizationType": "hvm",
            "CpuOptions": {
                "CoreCount": 1,
                "ThreadsPerCore": 1
            },
            "CapacityReservationSpecification": {
                "CapacityReservationPreference": "open"
            },
            "MetadataOptions": {
                "State": "pending",
                "HttpTokens": "required",
                "HttpPutResponseHopLimit": 2,
                "HttpEndpoint": "enabled",
                "HttpProtocolIpv6": "disabled",
                "InstanceMetadataTags": "disabled"
            },
            "EnclaveOptions": {
                "Enabled": false
            },
            "BootMode": "uefi-preferred",
            "PrivateDnsNameOptions": {
                "HostnameType": "ip-name",
                "EnableResourceNameDnsARecord": false,
                "EnableResourceNameDnsAAAARecord": false
            },
            "MaintenanceOptions": {
                "AutoRecovery": "default"
            },
            "CurrentInstanceBootMode": "legacy-bios"
        }
    ],
    "OwnerId": "058264207227",
    "ReservationId": "r-046bea443b5677130"
}

```

- EC2 instance (MyEC2) successfully created and running.

The screenshot shows the AWS EC2 Instances page. At the top, there's a search bar with 'Find Instance by attribute or tag (case-sensitive)' and a dropdown for 'All states'. Below the search bar is a filter section with 'Instance state = running' and a 'Clear filters' button. The main table has columns for Name, Instance ID, Instance state, Instance type, Status check, Alarm status, Availability Zone, and Put. One row is selected, showing 'MyEC2' with Instance ID 'i-0ab82a34977e11579', Instance state 'Running', Instance type 't2.micro', Status check 'Initializing', Alarm status 'View alarms +', Availability Zone 'us-east-1b', and Put 'ec2'. Below the table is a large empty space. Under the table, there's a section for the selected instance 'i-0ab82a34977e11579 (MyEC2)'. It includes tabs for Details, Status and alarms, Monitoring, Security, Networking, Storage, and Tags. The Details tab is active. It shows an 'Instance summary' with fields like Instance ID (i-0ab82a34977e11579), Public IPv4 address (3.84.227.19), Private IPv4 address (172.31.26.247), Public IPv6 DNS (ec2-3-84-227-19.compute-1.amazonaws.com), and Hostname type (IP name: ip-172-31-26-247.ec2.internal). There are also sections for Instance state (Running) and Private IP DNS name (ip-172-31-26-247.ec2.internal).

1. C. Create a bucket from aws cli
  - Create bucket using CLI command
  - BucketName - devprojectbucket

Bucket successfully created.

The screenshot shows the AWS S3 Objects page. At the top, there are tabs for Objects, Properties, Permissions, Metrics, Management, and Access Points. The Objects tab is active. Below the tabs is a toolbar with buttons for Copy S3 URI, Copy URL, Download, Open, Delete, Actions, and Create folder. A prominent 'Upload' button is highlighted in orange. A message below the toolbar says '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](#)'. There's also a search bar with 'Find objects by prefix' and a pagination control with '1' and '>'. The main content area has columns for Name, Type, Last modified, Size, and Storage class. A message at the bottom says 'No objects' and 'You don't have any objects in this bucket.' with another 'Upload' button.

- 2 . Put files in aws s3 using lambda.

## 2. A. Create a custom role for aws lambda which will have only put access

```
--policy-name LambdaS3PutPolicy \
--policy-document file://s3LamdaPolicy.json;

{

    "Policy": {

        "PolicyName": "LambdaS3PutPolicy",
        "PolicyId": "ANPAQ3EGRH55WG5PT2FXD",
        "Arn": "arn:aws:iam::058264207227:policy/LambdaS3PutPolicy",
        "Path": "/",
        "DefaultVersionId": "v1",
        "AttachmentCount": 0,
        "PermissionsBoundaryUsageCount": 0,
        "IsAttachable": true,
        "CreateDate": "2024-05-03T07:42:41Z",
        "UpdateDate": "2024-05-03T07:42:41Z"
    }
}
```

```
--role-name LambdaS3PutRole \
--assume-role-policy-document '{

    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "lambda.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}'
```

## 2. B. Add role to generate and access cloud watch logs.

```
--policy-name LambdaS3CloudWatchPolicy \
--policy-document file://cloudWatchPolicy.json

{

    "Policy": {

        "PolicyName": "LambdaS3CloudWatchPolicy",
        "PolicyId": "ANPAQ3EGRH55TWHIU46XI",
        "Arn": "arn:aws:iam::058264207227:policy/LambdaS3CloudWatchPolicy",
        "Path": "/",
        "DefaultVersionId": "v1",
        "AttachmentCount": 0,
        "PermissionsBoundaryUsageCount": 0,
        "IsAttachable": true,
        "CreateDate": "2024-05-03T08:06:28Z",
        "UpdateDate": "2024-05-03T08:06:28Z"
    }
}
```

- Create the policy and attach the policy to the Lambda S3 Put Role.

2. C. In Python Script, generate json in given format and save.json file in bucket created.

- Create a AWS Lambda function using a python file.

```
{  
    "FunctionName": "my-lambda-function",  
    "FunctionArn": "arn:aws:lambda:us-east-1:058264207227:function:my-lambda-function",  
    "Runtime": "python3.8",  
    "Role": "arn:aws:iam::058264207227:role/LambdaS3PutRole",  
    "Handler": "lambda_function.lambda_handler",  
    "CodeSize": 735,  
    "Description": "",  
    "Timeout": 3,  
    "MemorySize": 128,  
    "LastModified": "2024-05-03T09:36:03.755+0000",  
    "CodeSha256": "i758F6RdPdVbJ21A60dRXtMkTI7jSQMJ9i/qBuJX9tg=",  
    "Version": "$LATEST",  
    "TracingConfig": {  
        "Mode": "PassThrough"  
    },  
    "RevisionId": "64b7bc2c-8ccb-4a45-9240-ba1d40330322",  
    "State": "Pending",  
    "StateReason": "The function is being created.",  
    "StateReasonCode": "Creating",  
    "PackageType": "Zip",  
    "Architectures": [  
        "x86_64"  
    ],  
    "EphemeralStorage": {  
        "Size": 512  
    },  
    "SnapStart": {  
        "ApplyOn": "None",  
    }  
}
```

- Invoke a Lambda expression

```
"  
{  
    "StatusCode": 200,  
    "ExecutedVersion": "$LATEST"  
}
```

- The lambda function code.
- Added str(datetime.now()) in the timestamp field so that each log will have its own custom timestamp.

```

import boto3
import os
from datetime import datetime

def lambda_handler(event, context):
    # Get current execution count from environment variable
    execution_count = int(os.environ.get('EXECUTION_COUNT', 0))

    # Increment execution count
    execution_count += 1

    # Update execution count in environment variable
    os.environ['EXECUTION_COUNT'] = str(execution_count)

    # Stop execution after 3 runs
    if execution_count > 3:
        return {
            'statusCode': 200,
            'body': 'Execution stopped after 3 runs'
        }

    # Generate JSON data
    json_data = {
        "transaction_id": 12345,
        "payment_mode": "card/netbanking/upi",
        "amount": 200.0,
        "customer_id": 101,
        "timestamp": str(datetime.now())
    }

    # Save JSON data to S3

```

## 2. D. Schedule the job to run every three minutes and stop execution after 3 times.

The screenshot shows the AWS CloudWatch Metrics and Log Streams interface. On the left, there's a sidebar with navigation links for CloudWatch (Favorites and recents, Dashboards, Alarms, Logs, Log groups, Log Anomalies, Live Tail, Logs Insights), Metrics (X-Ray traces, Events, Application Signals, Network monitoring, Insights), and Settings (Getting Started, What's new). The main area displays log group details for 'arn:aws:logs:us-east-1:058264207227:log-group:/aws/lambda/my-lambda-function:\*'. It shows metrics like ARN, Metric filters (0), Subscription filters (0), Contributor Insights rules, KMS key ID, and Sensitive data count. Below this, the 'Log streams' section shows three log streams with their last event times and creation times:

Log stream	Last event time	Creation time
<a href="#">2024/05/03/[\$LATEST]5c263f0c5d1144c58993879f21ae77cc</a>	2024-05-03 15:41:07 (UTC+05:30)	29 minutes ago
<a href="#">2024/05/03/[\$LATEST]a228f77a5c654ae9a9f8e17e6d86b690</a>	2024-05-03 15:30:55 (UTC+05:30)	-
<a href="#">2024/05/03/[\$LATEST]f2a439a930a340f397c626b58c093389</a>	2024-05-03 15:24:34 (UTC+05:30)	-

## 2. E. Check if the cloud watch logs are generated.

- Logs are successfully pushed to the S3 bucket.

The screenshot shows the AWS S3 console with the 'Objects' tab selected. The interface includes buttons for Copy S3 URI, Copy URL, Download, Open, and Delete, along with Actions, Create folder, and Upload buttons. A search bar allows finding objects by prefix. The main table lists three objects:

	Name	Type	Last modified
	<a href="#">transaction_2024-05-03_10-11-07.json</a>	json	May 3, 2024, 15:41:11 (UTC+05:30)
	<a href="#">transaction_2024-05-03_10-12-04.json</a>	json	May 3, 2024, 15:42:11 (UTC+05:30)
	<a href="#">transaction_2024-05-03_10-13-08.json</a>	json	May 3, 2024, 15:43:11 (UTC+05:30)

## 3. A. Modify lambda function to accept parameters and return fileName.

- Lambda code changed and function tested.

```
import json
import boto3
from datetime import datetime

def lambda_handler(event, context):
    # Get parameters from the request body
    transaction_id = event.get('transaction_id')
    payment_mode = event.get('payment_mode')
    amount = event.get('amount')
    customer_id = event.get('customer_id')

    # Check if required parameters are present
    if not all([transaction_id, payment_mode, amount, customer_id]):
        return {
            'statusCode': 400,
            'body': json.dumps({'error': 'Missing required parameters'})
        }

    # Generate JSON data with current timestamp
    timestamp = datetime.now().isoformat()
    json_data = {
        "transaction_id": transaction_id,
        "payment_mode": payment_mode,
        "amount": amount,
        "customer_id": customer_id,
        "timestamp": timestamp
    }

    # Save JSON data to S3
```

```

file_name = f"transaction_{timestamp}.json"

s3 = boto3.client('s3')
s3.put_object(Bucket=bucket_name, Key=file_name, Body=json.dumps(json_data))

return {
    'statusCode': 200,
    'body': json.dumps({'file_name': file_name})
}

```

3. B. Create a POST api from gateway, pass parameters as request body to Lambda function. Return file name and status code as response.

```
{
  "id": "hjnr2aka14",
  "name": "MyFileAPI",
  "createdDate": 1714735910,
  "apiKeySource": "HEADER",
  "endpointConfiguration": {
    "types": [
      "EDGE"
    ],
  },
  "disableExecuteApiEndpoint": false,
  "rootResourceId": "rcqj7hini9"
}
```

```
--path-part files --region us-east-1
{
  "id": "t5s8dt",
  "parentId": "rcqj7hini9",
  "pathPart": "files",
  "path": "/files"
}
```

```
i9 --http-method POST --type AWS_PROXY --integration-http-method POST --uri 'arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/arn:aws:lambda:us-east-1:058264207227:function:my-lambda-function/invocations' --region us-east-1
{
  "type": "AWS_PROXY",
  "httpMethod": "POST",
  "uri": "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/arn:aws:lambda:us-east-1:058264207227:function:my-lambda-function/invocations",
  "passthroughBehavior": "WHEN_NO_MATCH",
  "timeoutInMillis": 29000,
  "cacheNamespace": "rcqj7hini9",
  "cacheKeyParameters": []
}
```

- Created the API, connected the API with lambda.

### 3. C. Consume the API from local machine and pass unique data to lambda.

```
lambda-function' \
-d '{"operation": "create", "payload": {"Item": {"transaction_id": 12345, "payment_mode": "card/netbanking/upi", "amount": 100, "customer_id": 101, "timestamp": "77"}}, \
{"file_name": "transaction_2024-05-03T12:13:39.852779.json"}'
```

```
import boto3
from datetime import datetime

def lambda_handler(event, context):
    #

    # Get parameters from the request body
    transaction_id = event.get('transaction_id')

    payment_mode = event.get('payment_mode')
    amount = event.get('amount')
    customer_id = event.get('customer_id')
    # return {'statusCode':200}

    if transaction_id!=None and payment_mode!=None and amount!=None and customer_id!=None:
        return {
            "statusCode":400,
            'body':'Missed'
        }

    # Generate JSON data with current timestamp
    timestamp = datetime.now().isoformat()
    json_data = {
        "transaction_id": transaction_id,
        "payment_mode": payment_mode,
        "amount": amount,
        "customer_id": customer_id,
        "timestamp": timestamp
    }

    # Save JSON data to S3
```

- Lambda code appropriately changed.

### 3. D. Check if cloud watch logs are generated.

CloudWatch > Log groups > /aws/lambda/my-lambda-function > 2024/05/03/[SLATEST]58f965ad875343b6b8faf04a520ea75f

**Log events**

You can use the filter bar below to search for and match terms, phrases, or values in your log events. [Learn more about filter patterns](#)

Filter events - press enter to search | 1m | 1h | Local timezone | Display |

Timestamp	Message
No older events at this moment. <a href="#">Retry</a>	
2024-05-03T17:47:35.790+05:30	INIT_START Runtime Version: python:3.8.v48 Runtime Version ARN: arn:aws:lambda:us-east-1::runtime:830ab..
2024-05-03T17:47:36.076+05:30	START RequestId: 01564a2e-6c0f-4298-9b64-33d564232789 Version: \$LATEST
2024-05-03T17:47:38.565+05:30	END RequestId: 01564a2e-6c0f-4298-9b64-33d564232789
2024-05-03T17:47:38.565+05:30	REPORT RequestId: 01564a2e-6c0f-4298-9b64-33d564232789 Duration: 2489.57 ms Billed Duration: 2490 ms Me..
2024-05-03T17:49:11.143+05:30	START RequestId: df038413-caf7-4cb6-9435-91ccb8fb435f Version: \$LATEST
2024-05-03T17:49:11.465+05:30	END RequestId: df038413-caf7-4cb6-9435-91ccb8fb435f