

Securing AWS resources by using AWS KMS

Task 1.1: Create a customer managed key and configure key rotation

The screenshot shows the 'Create key' wizard in the AWS KMS console. The left sidebar lists the steps: Step 1: Configure key (selected), Step 2: Add labels, Step 3 - optional: Define key administrative permissions, Step 4 - optional: Define key usage permissions, Step 5 - optional: Edit key policy, and Step 6: Review. The main area is titled 'Configure key' and contains two sections: 'Key type' and 'Key usage'. In the 'Key type' section, 'Symmetric' is selected, with a description: 'A single key used for encrypting and decrypting data or generating and verifying HMAC codes'. In the 'Key usage' section, 'Encrypt and decrypt' is selected, with a description: 'Use the key only to encrypt and decrypt data.' There are also options for 'Asymmetric' key type and 'Generate and verify MAC' key usage. At the bottom, there is an 'Advanced options' section and 'Cancel' and 'Next' buttons.

Here I opened AWS KMS, which is the place where you create and control encryption keys. I started the process to make my own custom key that I'll later use to protect data in my account.

The screenshot shows the 'Customer managed keys' list in the AWS KMS console. A green success message at the top states: 'Success Your AWS KMS key was created with alias MyKMSKey and key ID c133904f-636d-4ad6-a7fa-69540856b216.' Below this, there is a table with the following columns: Aliases, Key ID, Status, Key type, and Key spec. The table contains one entry: MyKMSKey, c133904f-636d-4ad6-a7fa-69540856b216, Enabled, Symmetric, and SYMMETRIC_DEFAULT.

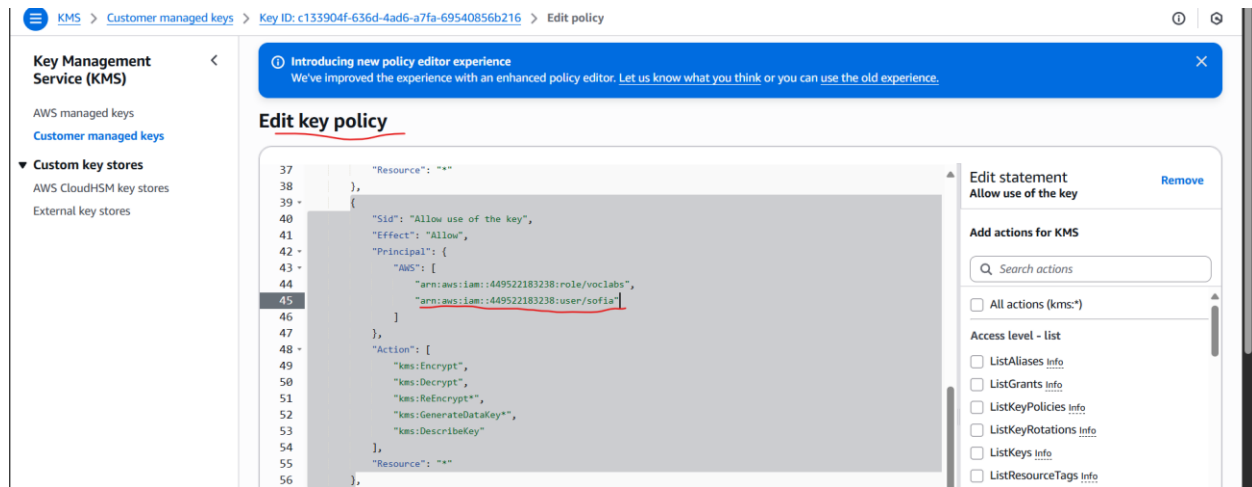
Aliases	Key ID	Status	Key type	Key spec
MyKMSKey	c133904f-636d-4ad6-a7fa-69540856b216	Enabled	Symmetric	SYMMETRIC_DEFAULT

Here I created a new symmetric encryption key named MyKMSKey and gave the voclabs role permission to both manage and use it. I followed the default settings to keep things simple. Once finished, my key was successfully created and ready to use for encrypting and decrypting data in AWS.

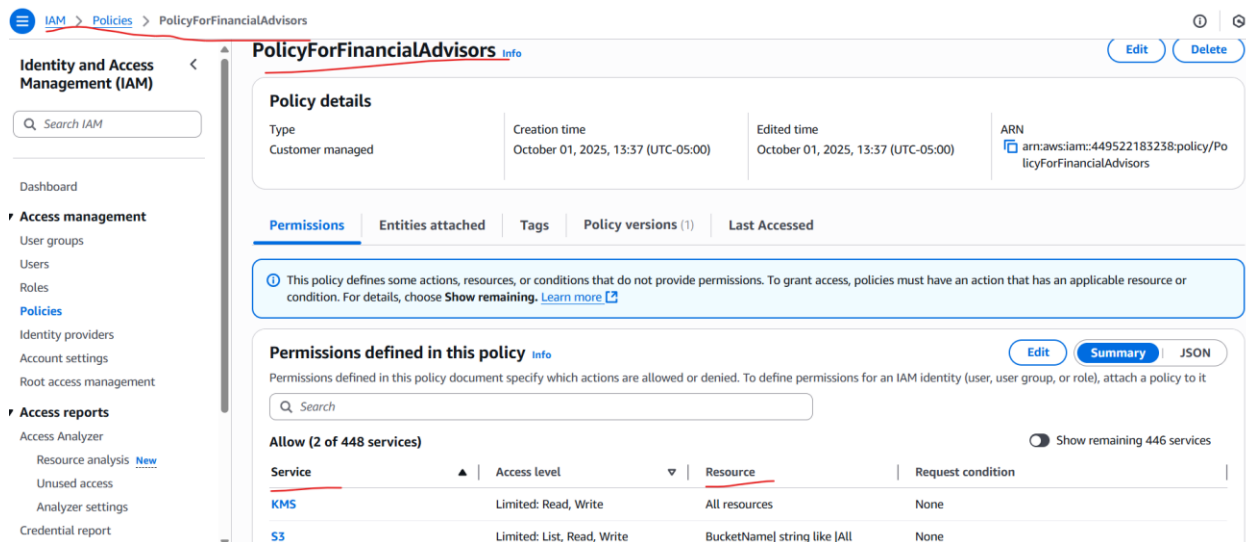
The screenshot shows the 'Key material and rotations' tab for the key MyKMSKey. The 'General configuration' section displays the key's details: Alias (MyKMSKey), ARN (arn:aws:kms:us-east-1:449522183238:key/c133904f-636d-4ad6-a7fa-69540856b216), Current key material ID (1378dddc675cddb4329e62b62891f60fa65ccb7d2a2633e6f58e113862082f34), Status (Enabled), Description (-), Creation date (Oct 15, 2025 20:17 CDT), and Regionality (Single Region). The 'Key material and rotations' section shows the 'Automatic key rotation' status as 'Enabled' with a rotation period of 365 days. The 'Date of last automatic rotation' is '-' and the 'Next rotation date' is 'Oct 15, 2026'. There is an 'Edit' button for the automatic key rotation settings.

Here I turned on the key rotation feature so AWS will automatically replace my key with a new one each year. This helps keep the encryption process more secure over time.

Task 1.2: Update the AWS KMS key policy and analyze an IAM policy

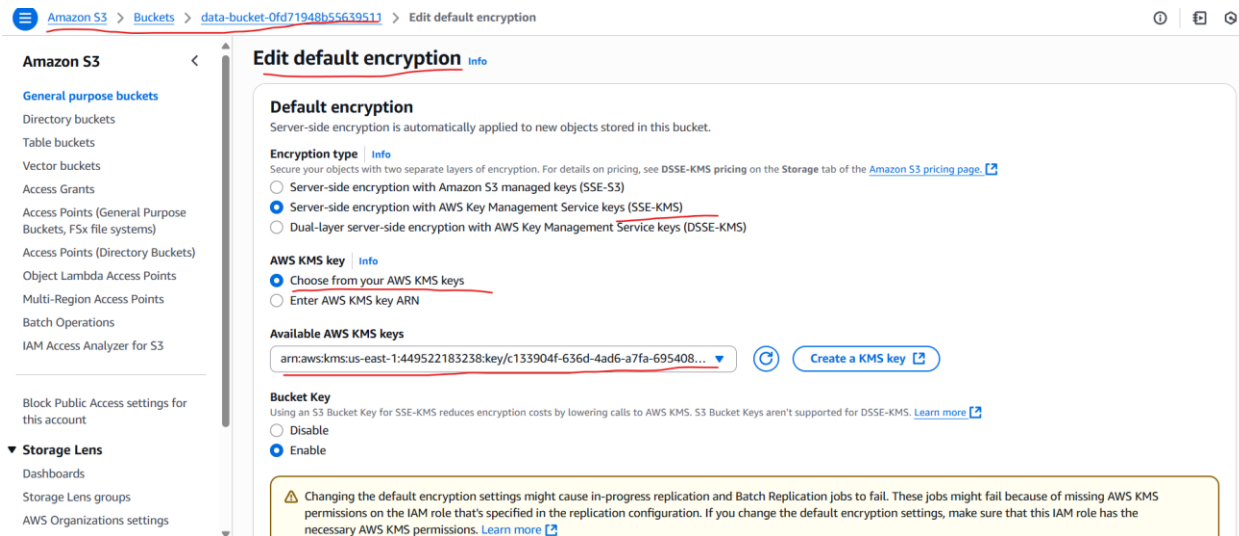


Here I updated the key policy to include both the voclabs role and the sofia user. Now, both can use this key to encrypt and decrypt data in AWS.

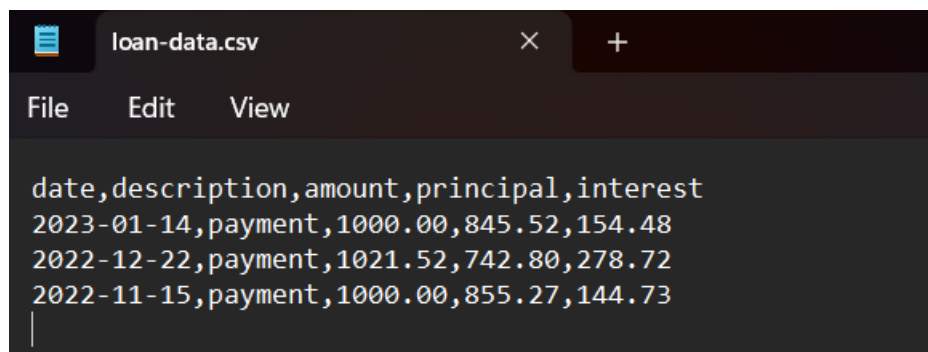


Here I looked at the PolicyForFinancialAdvisors IAM policy. It gives full control of all S3 buckets in the account and allows encrypting and decrypting objects. This policy is attached to the FinancialAdvisorGroup, which the sofia user is part of.

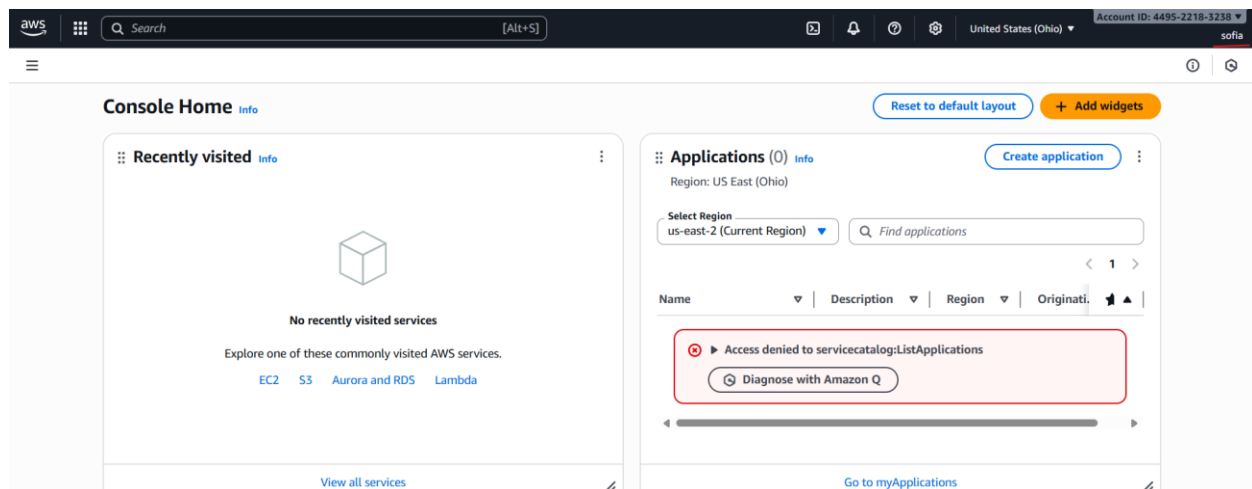
Task 1.3: Use AWS KMS to encrypt data in Amazon S3



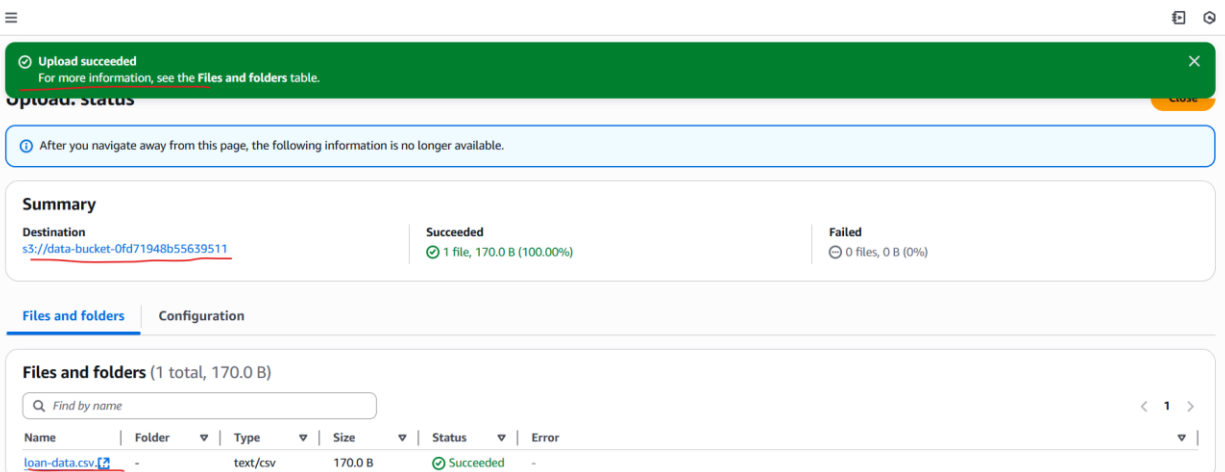
Here I opened my data-bucket in S3 and changed its default encryption setting to SSE-KMS using my custom KMS key called *MyKMSKey*. I did this so that any new files uploaded to this bucket are automatically encrypted using the key I created earlier. I found this setting under the bucket's *Properties* tab and confirmed the encryption type before saving.



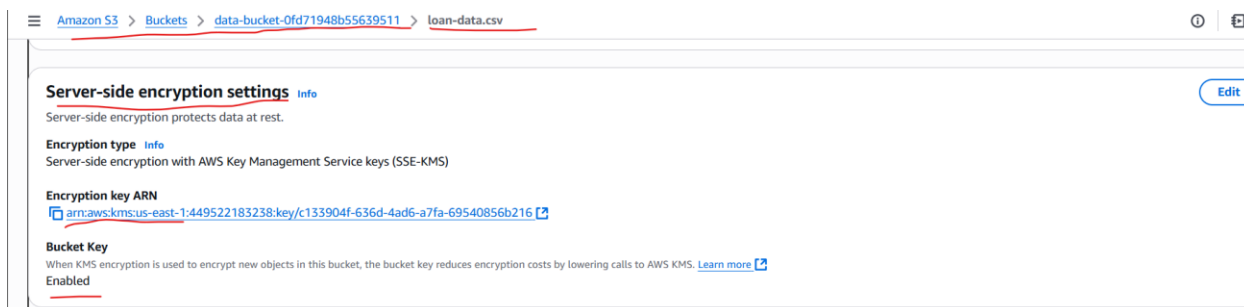
Here I created a simple CSV file called *loan-data.csv* that holds example loan payment data. This file will be used to test the encryption on my S3 bucket once I upload it. I saved it locally so I can easily upload it later as the test object.



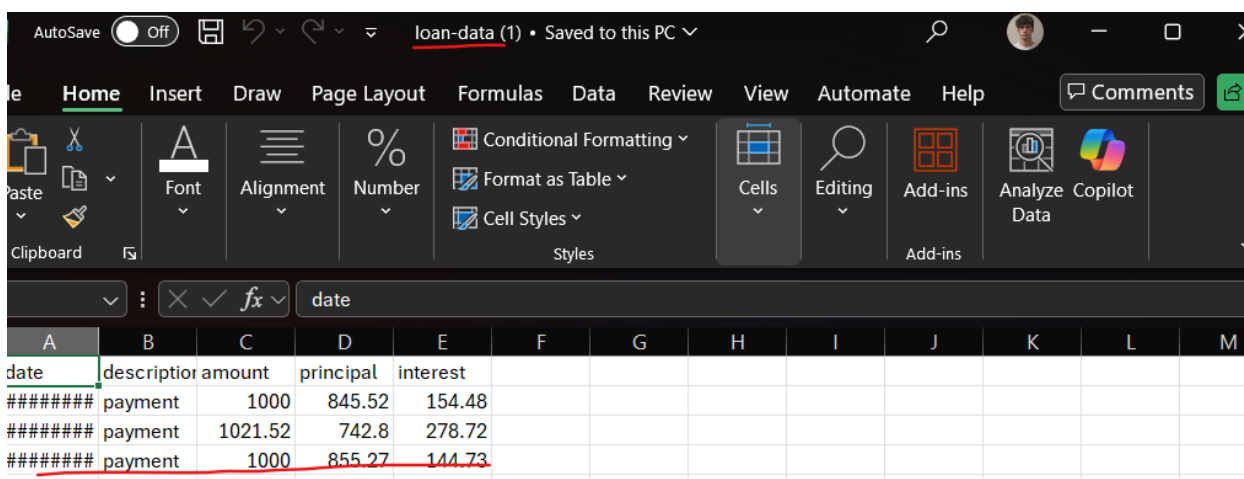
Here I logged in as the sofia user using a private window so I could test permissions separately from my main role. I used the IAM console's user sign-in link and the provided credentials to switch to Sofia's account safely.



Here I successfully uploaded *loan-data.csv* to my data-bucket as the sofia user. The upload worked because Sofia has the right permissions from three places — the S3 bucket policy, her IAM policy, and the KMS key policy that we edited earlier to include her in Task 3.2.



Here I confirmed that the uploaded file *loan-data.csv* is encrypted using SSE-KMS. I verified this by checking the file's properties in S3 and seeing my KMS key listed as the encryption method.



Here I tested whether Sofia could open or download the encrypted file. It worked, which shows that Sofia has the correct permissions to use the KMS key for decrypting data in this bucket.

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<Error>
  <Code>AccessDenied</Code>
  <Message>User: arn:aws:iam::449522183238:user/paulo is not authorized to perform: kms:Decrypt on
resource: arn:aws:kms:us-east-1:449522183238:key/c133904f-636d-4ad6-a7fa-69540856b216 because no
identity-based policy allows the kms:Decrypt action</Message>
  <RequestId>H1AYJREMBR96S5TP</RequestId>
  <HostId>WifMS5C42B8N9ONSFYSSCSdE+L6v7t5YXx0uxpjZxVhfpYsDFQANeJPP1K9Gxqk5em17wkoHB0g=</HostId>
</Error>
```

Here I tested access as the paulo user, and the download failed. This happened because Paulo isn't included in the KMS key policy, even though he can normally access other S3 files. That shows the KMS encryption adds an extra layer of security beyond normal S3 and IAM permissions.

Task 1.4: Use AWS KMS to encrypt the root volume of an EC2 instance

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

 [Add additional tags](#)

Here I started creating a new EC2 instance while signed in as the voclabs role and gave it the name *EncryptedInstance* to identify it easily later.

AWS Management Console: EC2 > Instances

Instances (1/4) [Info](#)

Find Instance by attribute or tag (case-sensitive)

	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability
<input type="checkbox"/>	WebServer2	i-0c82da38210757fa2	Running	t2.micro	2/2 checks passed	View alarms	us-east-1a
<input type="checkbox"/>	WebServer	i-08688a5f7c6a01ba5	Running	t2.micro	2/2 checks passed	View alarms	us-east-1a
<input checked="" type="checkbox"/>	EncryptedInstance	i-06b9768330cc12a22	Running	t3.micro	Initializing	View alarms	us-east-1a
<input type="checkbox"/>	aws-cloud9-Cloud9Instance-127ea2d3e4c...	i-05d686821296c60a5	Running	t2.micro	2/2 checks passed	View alarms	us-east-1c

i-06b9768330cc12a22 (EncryptedInstance)

To build *EncryptedInstance*, I selected the Amazon Linux 2 AMI and used t3.micro since the console wouldn't allow the t2.micro option for this image. It shouldn't make a difference in how the lab functions. I connected it to the NetworkFirewallVPC, used WebServerSubnet2, and applied the WebServer2SecurityGroup. I chose vockey for secure SSH access, then enabled encryption on the root volume using my customer-managed key (*MyKMSKey*). Finally, I attached WebServerInstanceProfile for AWS service access and launched the instance. These steps resulted in a running EC2 instance that's fully connected and encrypted with AWS KMS.


```
[ec2-user@webserver2 ~]$ aws kms list-keys
{
  "Keys": [
    {
      "KeyId": "125fe9e3-3a21-494f-9e44-ab41486b4079",
      "KeyArn": "arn:aws:kms:us-east-1:449522183238:key/125fe9e3-3a21-494f-9e44-ab41486b4079"
    },
    {
      "KeyId": "1b315478-d2c6-49ae-9c19-af5711469925",
      "KeyArn": "arn:aws:kms:us-east-1:449522183238:key/1b315478-d2c6-49ae-9c19-af5711469925"
    },
    {
      "KeyId": "477f0538-5058-48aa-8bcf-3c0cf832ebc4",
      "KeyArn": "arn:aws:kms:us-east-1:449522183238:key/477f0538-5058-48aa-8bcf-3c0cf832ebc4"
    },
    {
      "KeyId": "c133904f-636d-4ad6-a7fa-69540856b216",
      "KeyArn": "arn:aws:kms:us-east-1:449522183238:key/c133904f-636d-4ad6-a7fa-69540856b216"
    }
  ]
}

[ec2-user@webserver2 ~]$ result=$(aws kms generate-data-key --key-id alias/MyKMSKey --key-spec AES_256)
echo $result | python3 -m json.tool
{
  "CiphertextBlob": "AQIDAHgTeN3c21zdtDKeyrYokfYpPy7fSomM+b1jhE4YggvNAGngjhJCio/gbuEogNapt4xAAAAfjB8BgkqhkiG9w0BBwagbzBTAgEAMGCGCSqGS1b3DQEHA7AeBg1ghkgBZQMEAS4wEQQMTNjKwHsqqe+c4PM1AgEQgDt8TOVZnfwOM5QXfScPyghO8p4MTJfO8q59t3CYBOjaF6JqSxgiEpByMnCY6EHPeiferIUPUXQ9d6aaQ==",
  "Plaintext": "eDX9ih/uRuRwTE/Wuhh8glckXzBBB593ILg/VvGvPw=",
  "KeyId": "arn:aws:kms:us-east-1:449522183238:key/c133904f-636d-4ad6-a7fa-69540856b216",
  "KeyMaterialId": "1378dddc675cddb4329e62b62891f60fa65cbb7d2a2633e6f58e113862082f34"
}
```

i-0c82da38210757fa2 (WebServer2)
PublicIPs: 54.165.197.53 PrivateIPs: 10.1.3.4

Here I confirmed that my EC2 instance could talk to AWS KMS by listing available keys, then generated a new data key from *MyKMSKey*. The output showed both an encrypted (CiphertextBlob) and a plaintext version of this key, which I'll use to encrypt files.

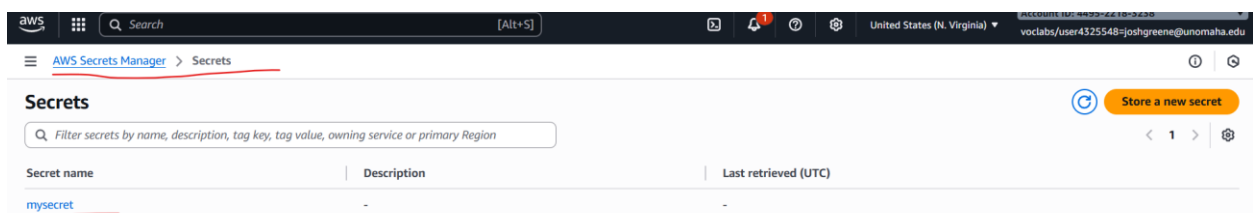
```
[ec2-user@webserver2 ~]$ dk_cipher=$(echo $result | jq '.CiphertextBlob' | cut -d '"' -f2)
echo $dk_cipher | base64 --decode > data_key_ciphertext
[ec2-user@webserver2 ~]$ aws kms decrypt --ciphertext-blob fileb:///data_key_ciphertext --query Plaintext --output text | base64 --decode > data_key_plaintext_encrypted

[ec2-user@webserver2 ~]$ ls
data_key_ciphertext  data_key_plaintext_encrypted  data_unencrypted.txt  data_unencrypted.txt~  index.html
[ec2-user@webserver2 ~]$ echo "let's encrypt these file contents. Sensitive data here." > data_unencrypted.txt
[ec2-user@webserver2 ~]$ openssl enc -aes-256-cbc -salt -pbkdf2 -in data_unencrypted.txt -out data_encrypted -pass file:data_key_plaintext_encrypted
[ec2-user@webserver2 ~]$ rm data_unencrypted.txt
[ec2-user@webserver2 ~]$ openssl enc -d -aes-256-cbc -pbkdf2 -in data_encrypted -out data_decrypted.txt -pass file:data_key_plaintext_encrypted
cat data_decrypted.txt
let's encrypt these file contents. Sensitive data here.
[ec2-user@webserver2 ~]$
```

i-0c82da38210757fa2 (WebServer2)
PublicIPs: 54.165.197.53 PrivateIPs: 10.1.3.4

In this task, I used AWS KMS envelope encryption on my WebServer2 instance to protect data in place. I connected to the instance using EC2 Instance Connect, generated a new data key from my customer-managed key (*MyKMSKey*), and saved both the encrypted and decrypted versions of the key on disk. I then used OpenSSL with the plaintext data key to encrypt a text file called *data_unencrypted.txt*, turning it into unreadable ciphertext. After that, I deleted the unencrypted version for security and successfully decrypted the encrypted file back to its original readable form. This showed that my KMS key worked properly to protect and recover data using envelope encryption.

Task 3.6: Use AWS KMS to encrypt a Secrets Manager secret



Here I created a new secret in AWS Secrets Manager with the key-value pair secret: my secret data. I encrypted it using my customer-managed key (*MyKMSKey*) so the contents are protected by AWS KMS. I named the secret *mysecret*, completing the setup for secure storage.


```

"SecretList": [
  {
    "ARN": "arn:aws:secretsmanager:us-east-1:449522183238:secret:mysecret-4AN0q1",
    "Name": "mysecret",
    "KmsKeyId": "arn:aws:kms:us-east-1:449522183238:key/c133904f-636d-4ad6-a7fa-69540856b216",
    "LastChangedDate": "2025-10-16T18:44:59.556000+00:00",
    "Tags": [],
    "SecretVersionsToStages": {
      "29e2073b-1be8-40d5-84d5-18c75d7cd6b2": [
        "AWSCURRENT"
      ]
    },
    "CreateDate": "2025-10-16T18:44:59.479000+00:00"
  }
]
}
ec2-user@webserver2 ~]$ aws secretsmanager get-secret-value --secret-id mysecret
{
  "ARN": "arn:aws:secretsmanager:us-east-1:449522183238:secret:mysecret-4AN0q1",
  "Name": "mysecret",
  "VersionId": "29e2073b-1be8-40d5-84d5-18c75d7cd6b2",
  "SecretString": "i\\secret\\:\\my secret data\\",
  "VersionStages": [
    "AWSCURRENT"
  ],
  "CreateDate": "2025-10-16T18:44:59.550000+00:00"
}
ec2-user@webserver2 ~]$

```

i-0c82da38210757fa2 (WebServer2)
PublicIPs: 54.165.197.53 PrivateIPs: 10.1.3.4

Here I connected to my WebServer2 instance and used the AWS CLI to confirm that my secret (*mysecret*) existed. Then I retrieved it successfully using the `get-secret-value` command, which displayed the key-value pair stored in Secrets Manager and decrypted it using my *MyKMSKey*.

Project Summary:

In this project, I learned how to use AWS Key Management Service (KMS) to secure data across different AWS services. I started by creating my own key called *MyKMSKey*, turned on automatic key rotation, and gave the right roles and users permission to use it. Then I used the key to encrypt objects stored in S3, which made sure only certain users could access them. I also launched an EC2 instance with its root volume encrypted using my KMS key, which helped me understand how encryption protects data that sits on storage volumes.

Later, I practiced envelope encryption on my WebServer2 instance using the AWS CLI and OpenSSL to encrypt and decrypt a test file. I also created a Secrets Manager secret, encrypted it with *MyKMSKey*, and confirmed that I could safely retrieve it through the CLI. This phase helped me see how AWS KMS brings all these services together to keep data protected, while giving me control over who can access or decrypt sensitive information.