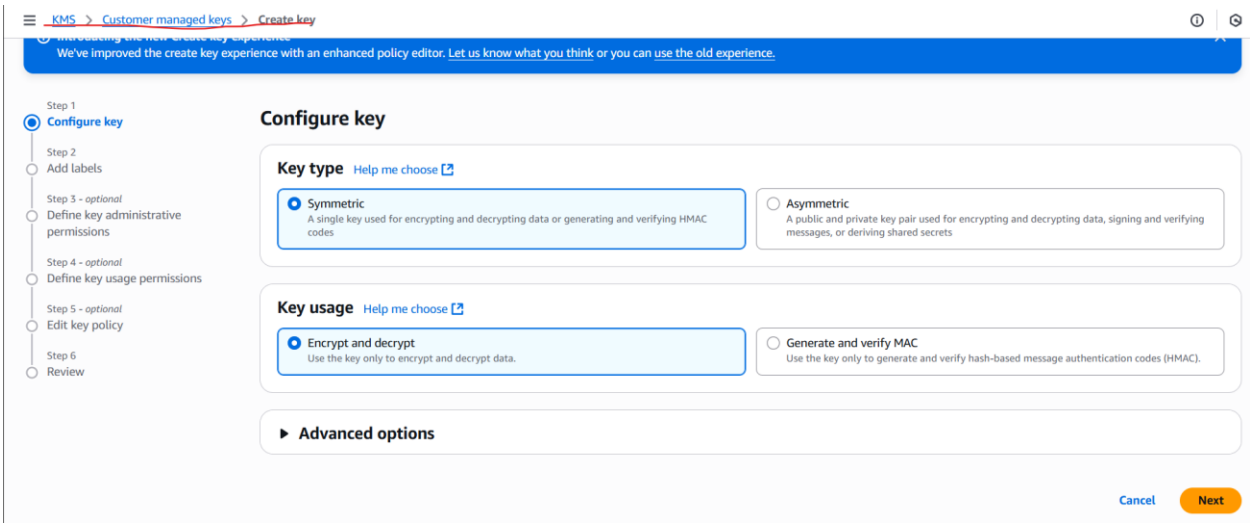
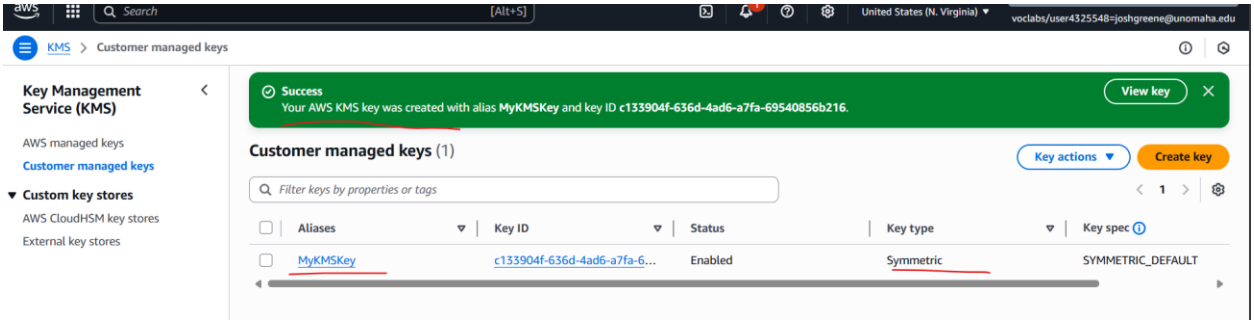
Securing AWS resources by using AWS KMS

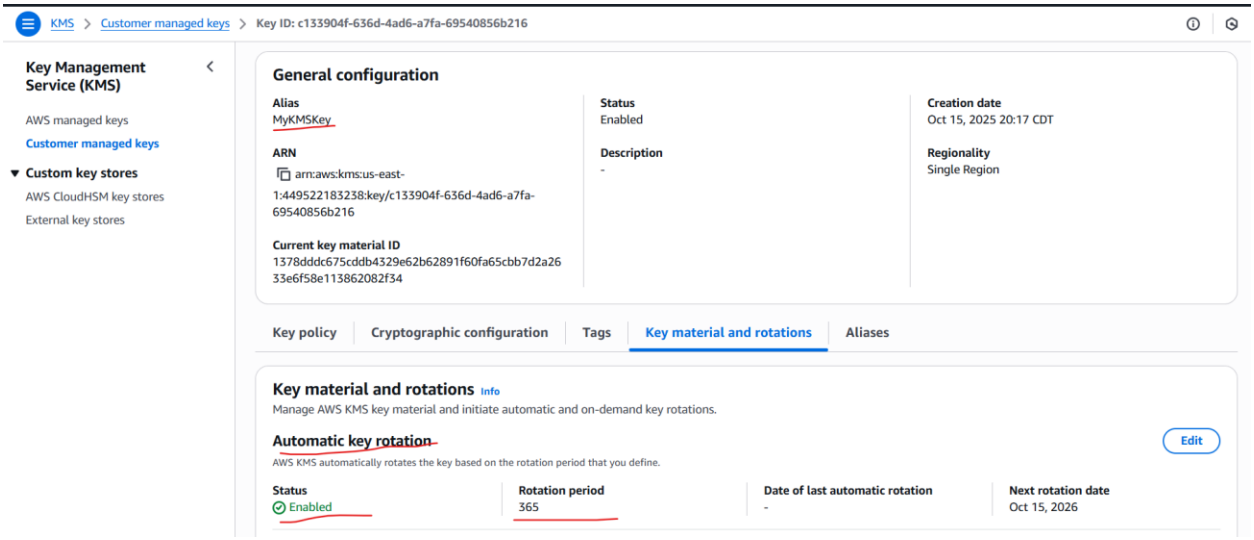
Task 1.1: Create a customer managed key and configure key rotation



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.

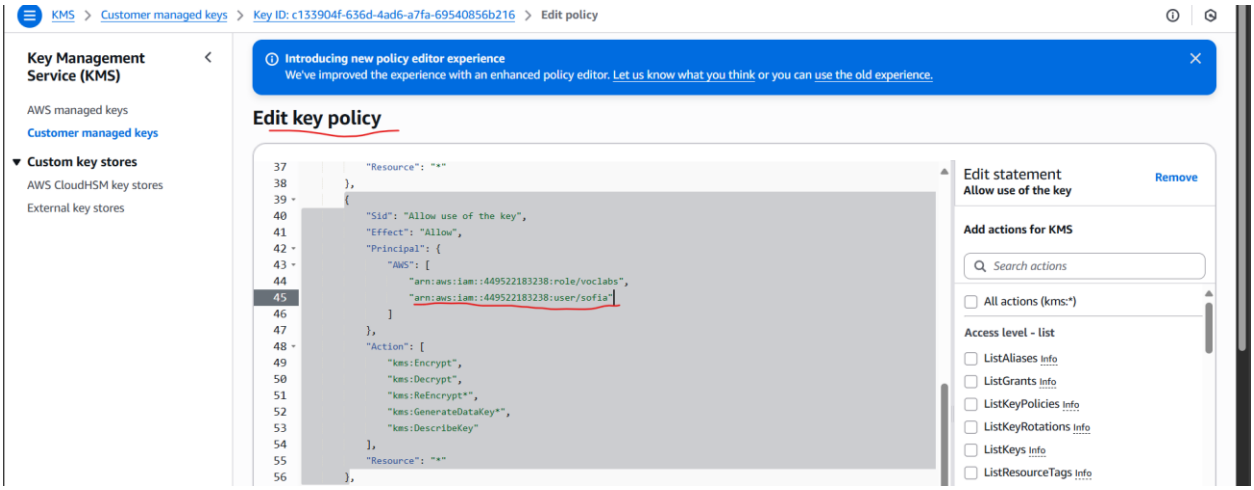


Here I created a new symmetric encryption key named MyKMSKey and gave the voclabs role permission to both manage and use it. I followed the default settings to keep things simple. Once finished, my key was successfully created and ready to use for encrypting and decrypting data in AWS.

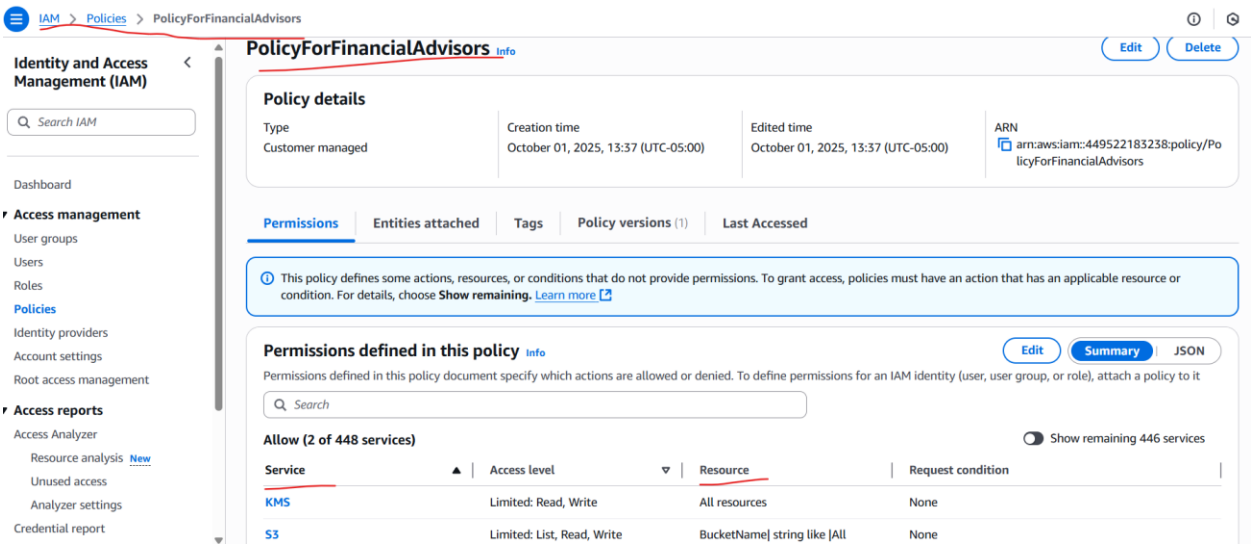


Here I turned on the key rotation feature so AWS will automatically replace my key with a new one each year. This helps keep the encryption process more secure over time.

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

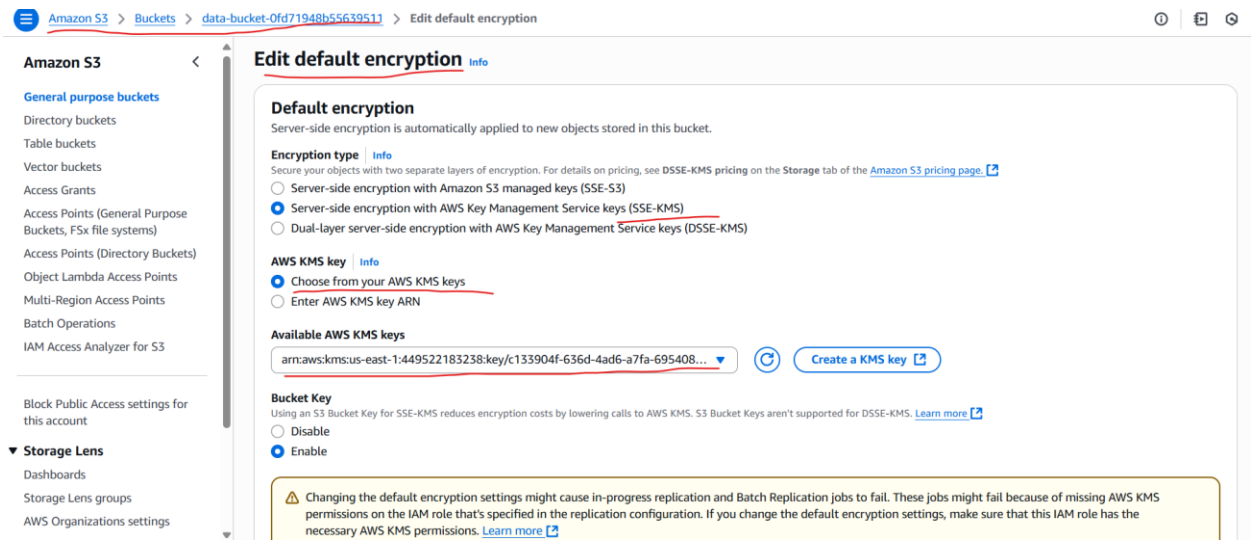


Here I updated the key policy to include both the voclabs role and the sofia user. Now, both can use this key to encrypt and decrypt data in AWS.

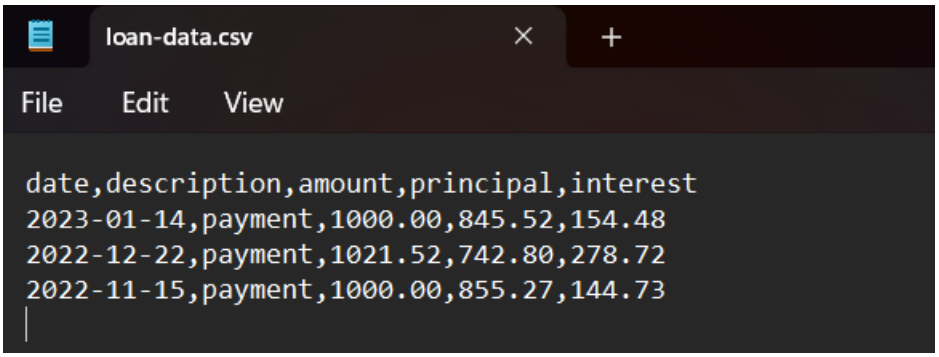


Here I looked at the PolicyForFinancialAdvisors IAM policy. It gives full control of all S3 buckets in the account and allows encrypting and decrypting objects. This policy is attached to the FinancialAdvisorGroup, which the sofia user is part of.

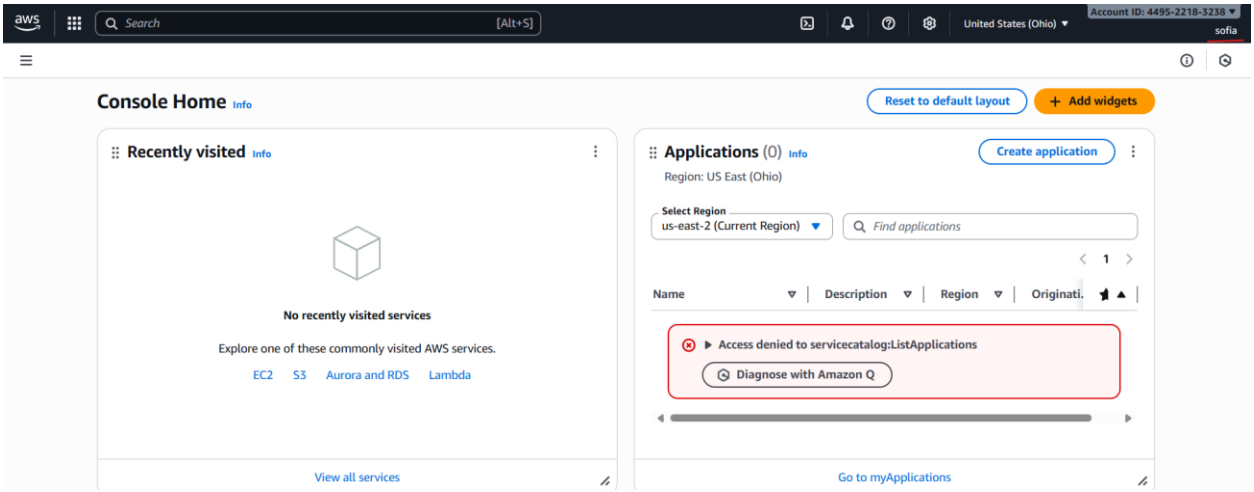
Task 1.3: Use AWS KMS to encrypt data in Amazon S3



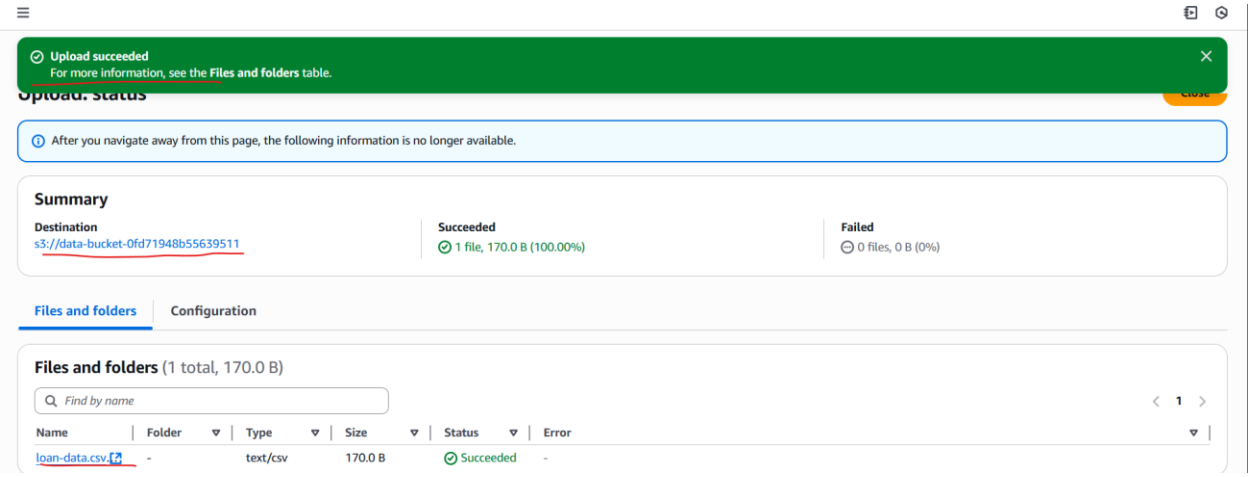
Here I opened my data-bucket in S3 and changed its default encryption setting to SSE-KMS using my custom KMS key called *MyKMSKey*. I did this so that any new files uploaded to this bucket are automatically encrypted using the key I created earlier. I found this setting under the bucket's *Properties*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



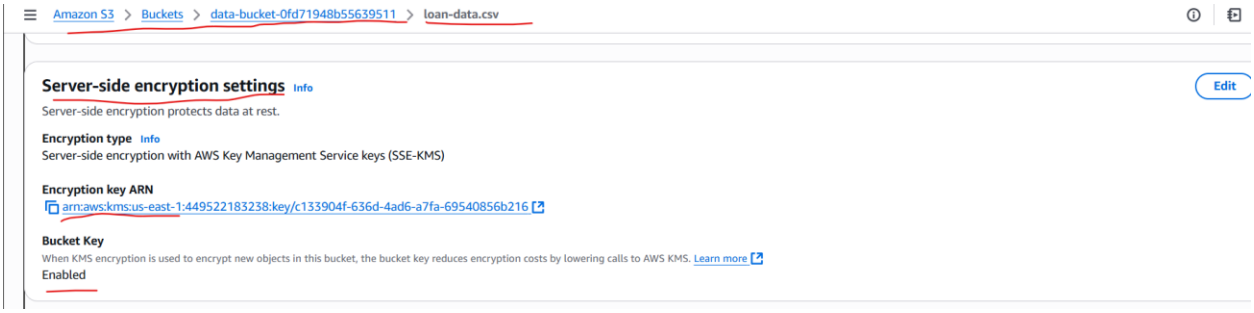
Here I created a simple CSV file called *loan-data.csv* that holds example loan payment data. This file will be used to test the encryption on my S3 bucket once I upload it. I saved it locally so I can easily upload it later as the test object.



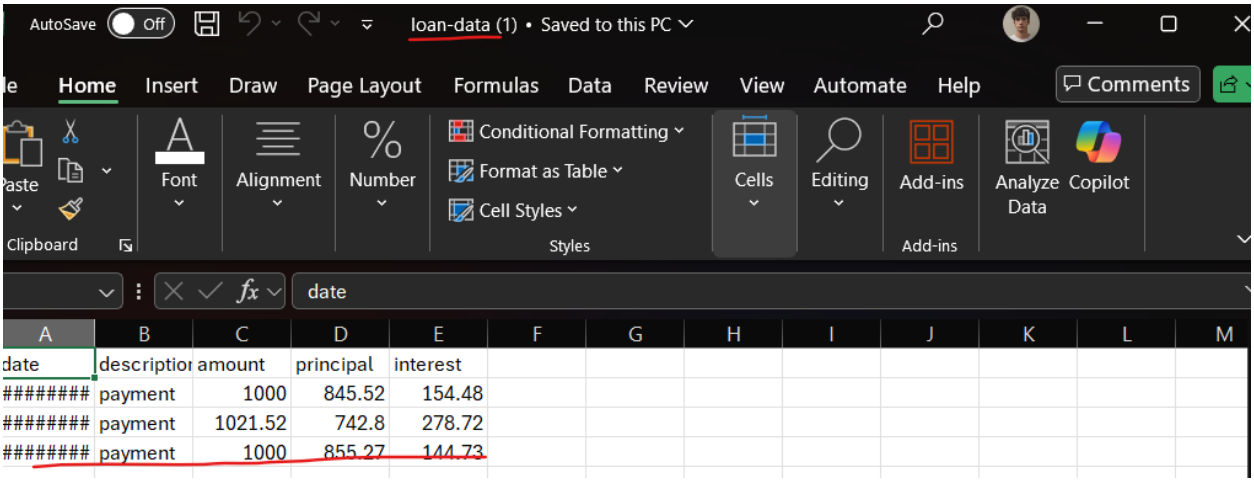
Here I logged in as the sofia user using a private window so I could test permissions separately from my main role. I used the IAM console's user sign-in link and the provided credentials to switch to Sofia's account safely.



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.



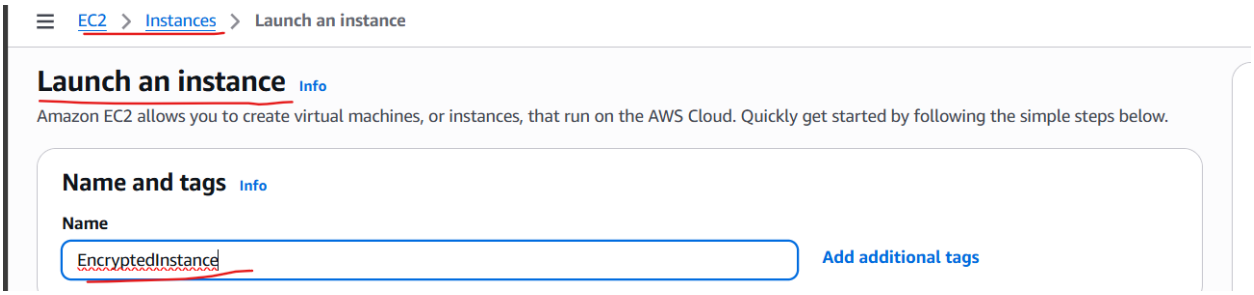
Here I confirmed that the uploaded file *loan-data.csv* is encrypted using SSE-KMS. I verified this by checking the file's properties in S3 and seeing my KMS key listed as the encryption method.



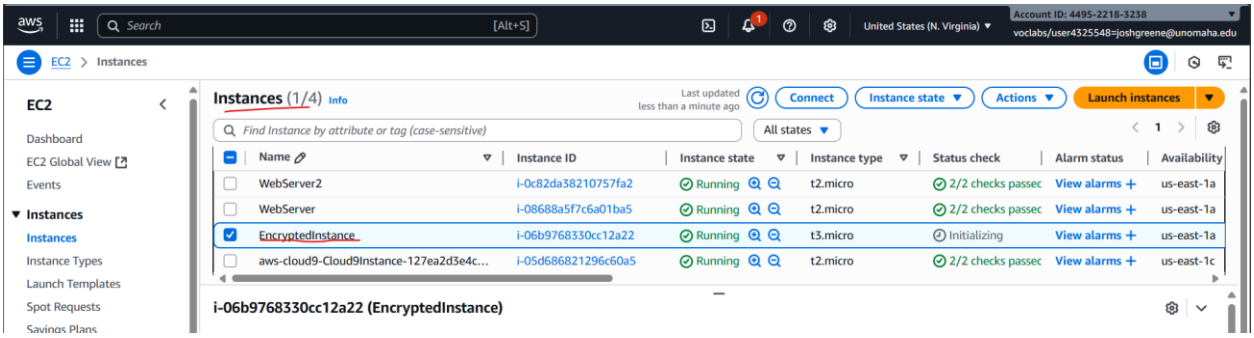
Here I tested whether Sofia could open or download the encrypted file. It worked, which shows that Sofia has the correct permissions to use the KMS key for decrypting data in this bucket.

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.

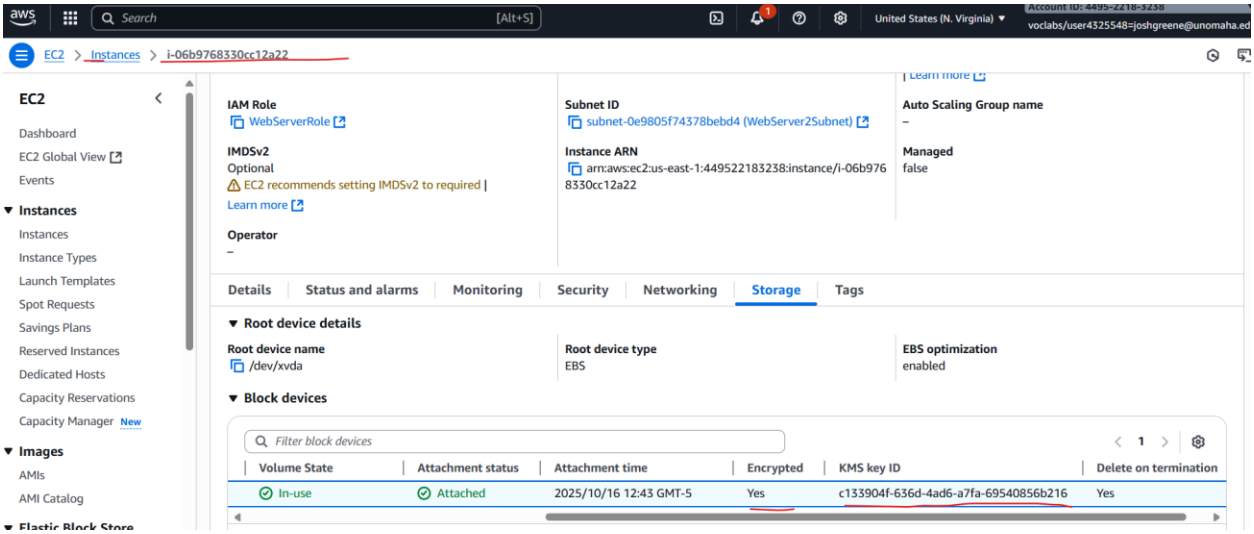
Task 1.4: Use AWS KMS to encrypt the root volume of an EC2 instance



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.

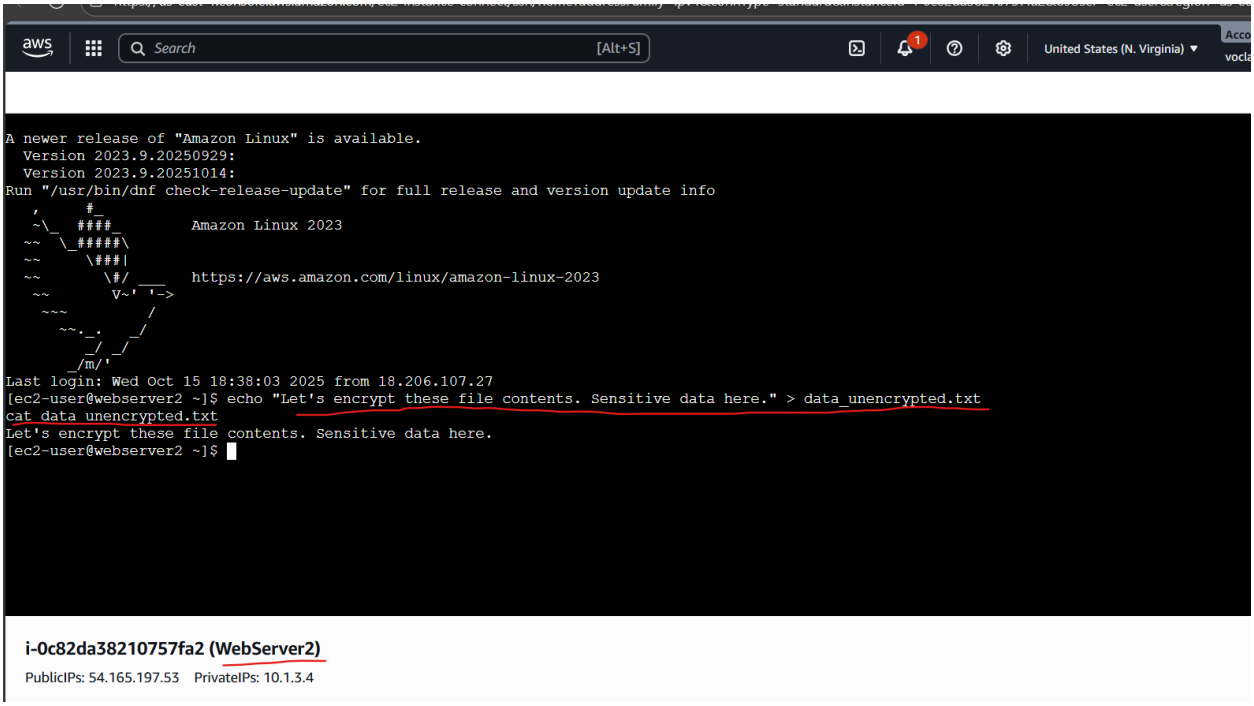


To build *EncryptedInstance*, I selected the Amazon Linux 2 AMI and used t3.micro since the console wouldn't allow the t2.micro option for this image. It shouldn't make a difference in how the lab functions. I connected it to the NetworkFirewallVPC, used WebServerSubnet2, and applied the WebServer2SecurityGroup. 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.

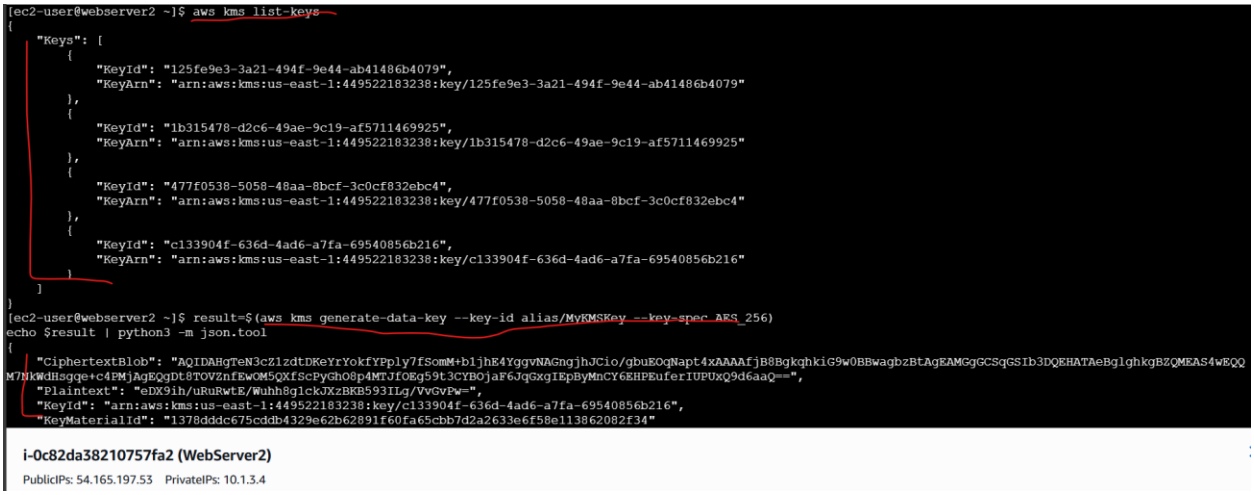


Here I verified that the instance's root volume shows *Encrypted: Yes* with my *MyKMSKey* ARN, confirming the EC2 instance is fully encrypted using AWS KMS.

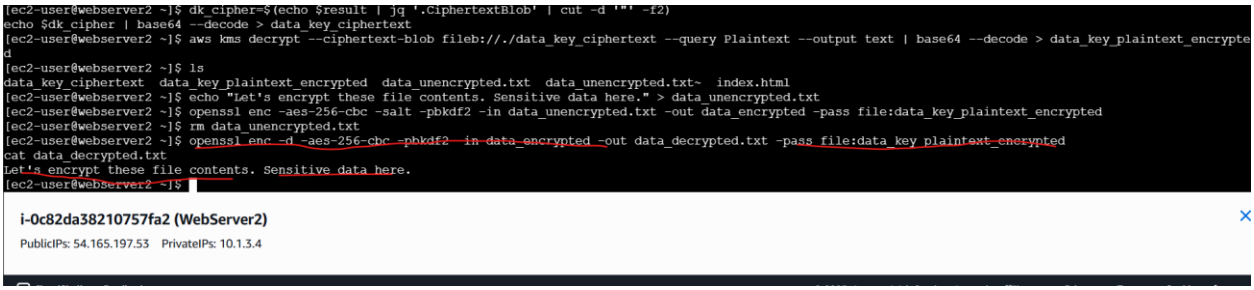
Task 1.5: Use AWS KMS envelope encryption to encrypt data in place



Here I connected to my existing EC2 instance (*WebServer2*) using EC2 Instance Connect. This lets me run commands directly in the instance without using SSH keys. Here I made a new file called *data_unencrypted.txt* containing a simple message that will act as my sample "sensitive data." I checked it with cat to confirm the file was created successfully.

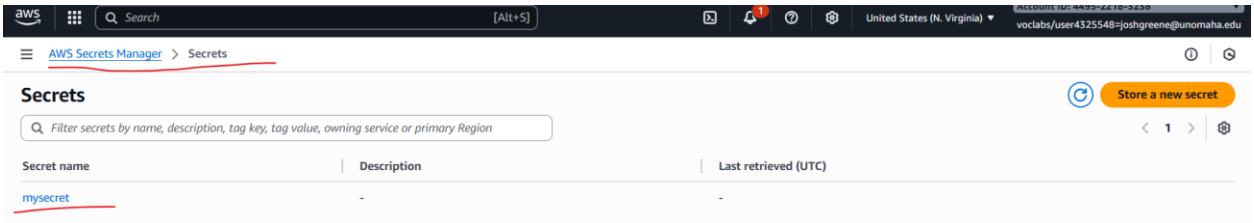


Here I confirmed that my EC2 instance could talk to AWS KMS by listing available keys, then generated a new data key from *MyKMSKey*. The output showed both an encrypted (CiphertextBlob) and a plaintext version of this key, which I'll use to encrypt files.

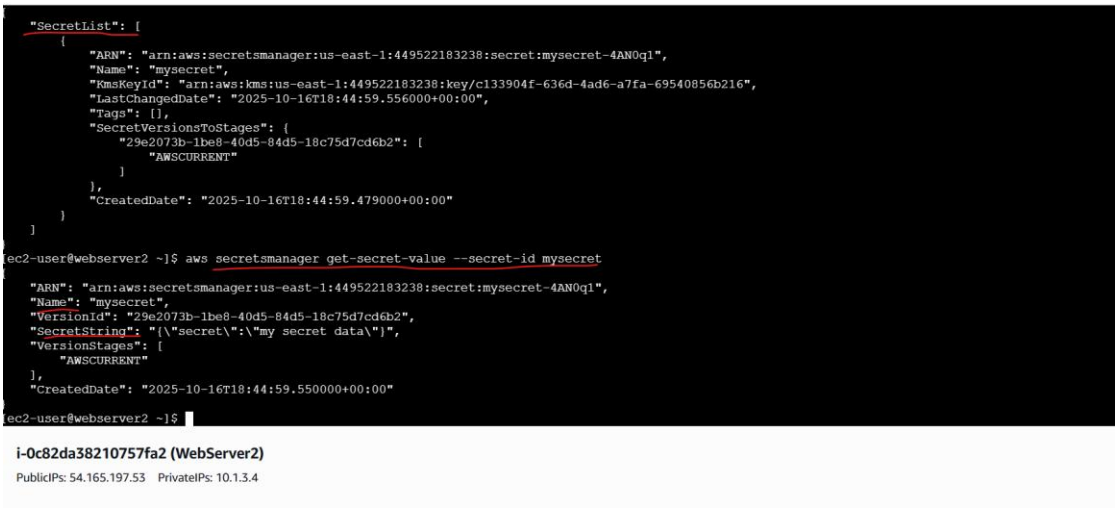


In this task, I used AWS KMS envelope encryption on my WebServer2 instance to protect data in place. I connected to the instance using EC2 Instance Connect, generated a new data key from my customer-managed key (*MyKMSKey*), and saved both the encrypted and decrypted versions of the key on disk. I then 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.



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.