Require SSE-KMS Encryption

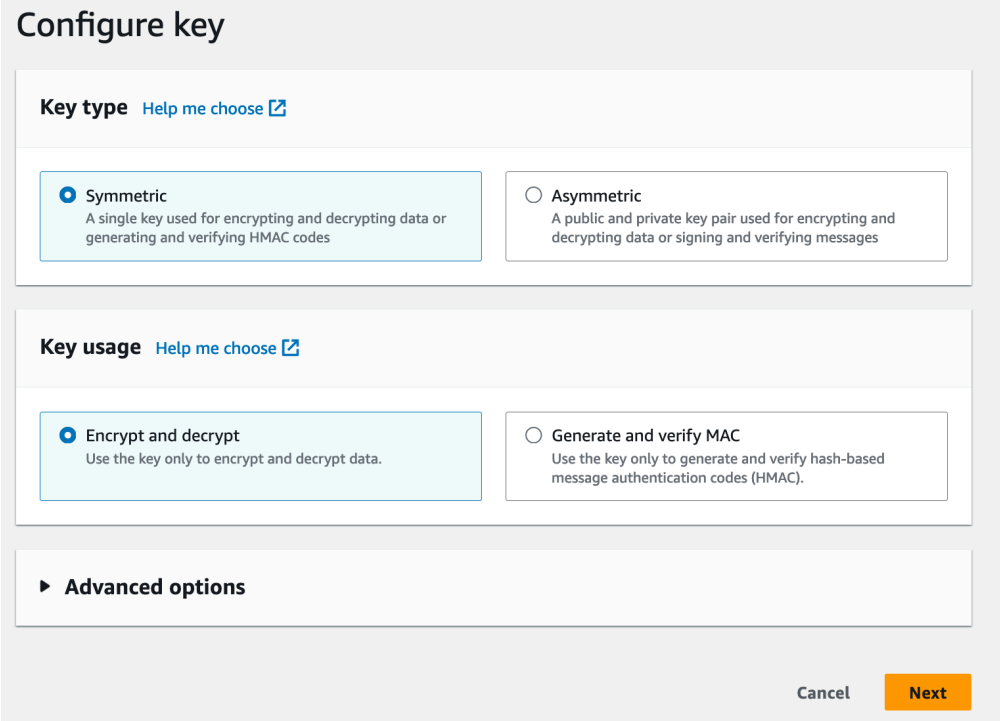
In this exercise, we will configure default bucket encryption to SSE-KMS with customer managed keys to encrypt your data at rest and enforce encryption with a bucket policy.

In following with best practices, we will leverage both SSE-KMS and create a customer managed key. This is generally a best practice because it allows key usage auditing, ability to define key policies for access, and enables cross account sharing as needed in the organization.

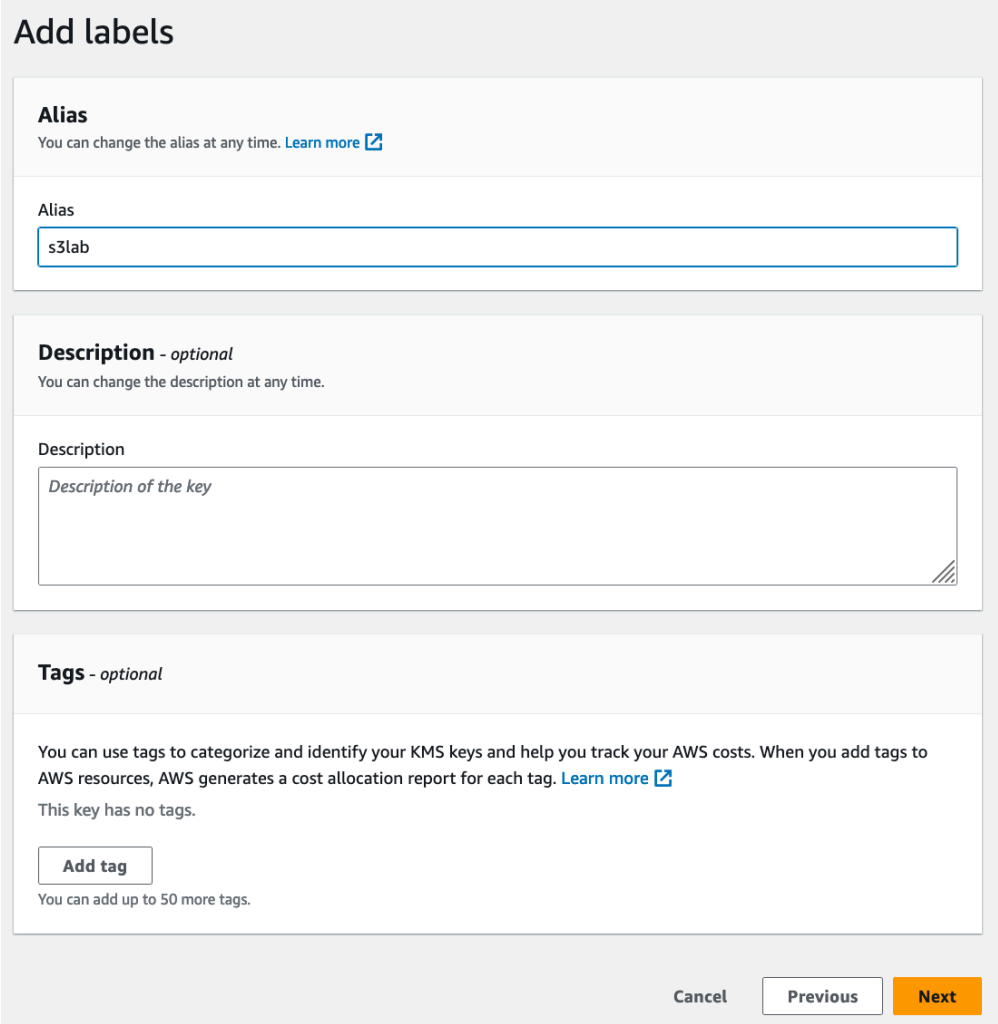
We will start by creating a KMS key in the account and assigning it the proper permissions for encryption and decryption.

From the AWS console in the top search bar, search and select `kms` for Key Management Service

- Click the `Create a Key` button to get started
- We will keep the defaults and click `Next` as shown below:

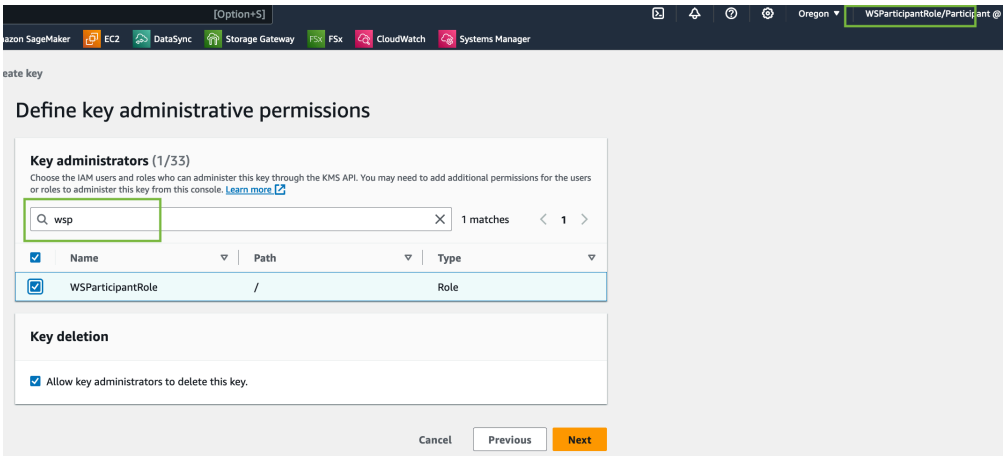


In the next section, under `Alias` we will name it "s3lab" as shown.



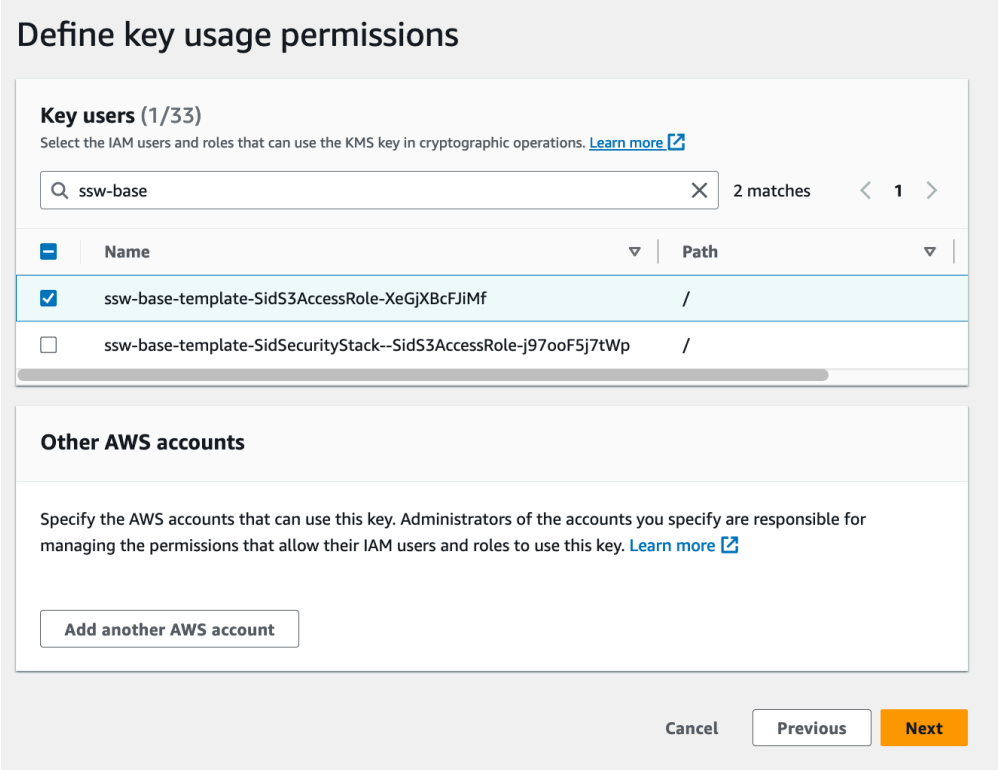
Click `Next`

The next page is labeled `Define key administrative permissions` this is where we will enable an administrator who can perform different actions on the key, like modifying attributes or even deleting the key. Notice in the screenshot below we are logged in as the `WSParticipantRole` as noted in the top right and we typed "wsp" in the in the search box. Click on the checkbox next to the `WSParticipantRole` and click `Next`



In the next section labeled `Define key usage permissions` we will create the IAM User or Role that is able to use the key. This defines who can perform decrypt and encrypt actions using the key. It is important to note, this allows customers to choose who has administrative access to the key and able to do cryptographic operations on the key, in our case this is two different users/roles. This is a key aspect to least privilege access and distinguishing different roles and responsibilities from a human perspective.

Type in `ssw-base` in the search bar and check the box next to `ssw-base-template-SidS3AccessRole-xxxx` where xxxx may be an alpha-numeric string. Click `Next`



At the last screen keep everything as default and scroll down to the bottom and click `Finish`.

You should now see a key created with an alias of `s3lab`. We will use this key later in the section.

Open an SSH session to the SID-security-instance using EC2 Instance Connect if it is not already open. Run the following commands to create a test file.

```
cd ~
echo "123456789abcdefg" > textfile
```

Run the following command in your SSH session to put a text01 object to your bucket.

```
aws s3api put-object --key text01 --body textfile --bucket ${bucket}
```

The request should succeed, notice how the encryption is AES256 (SSE-S3)

```
[ec2-user@storage-workshop ~]$
[ec2-user@storage-workshop ~]$ aws s3api put-object --key text01 --body textfile --bucket ${bucket}
{
    "ETag": "\"3ca451faac980583cffaadf8b63e6820\"",
    "ServerSideEncryption": "AES256"
}
[ec2-user@storage-workshop ~]$
```

This is because the current bucket default encryption is configured as SSE-S3.

Let's update our bucket encryption default to use SSE-KMS so new objects will inherit SSE-KMS as it's default encryption.

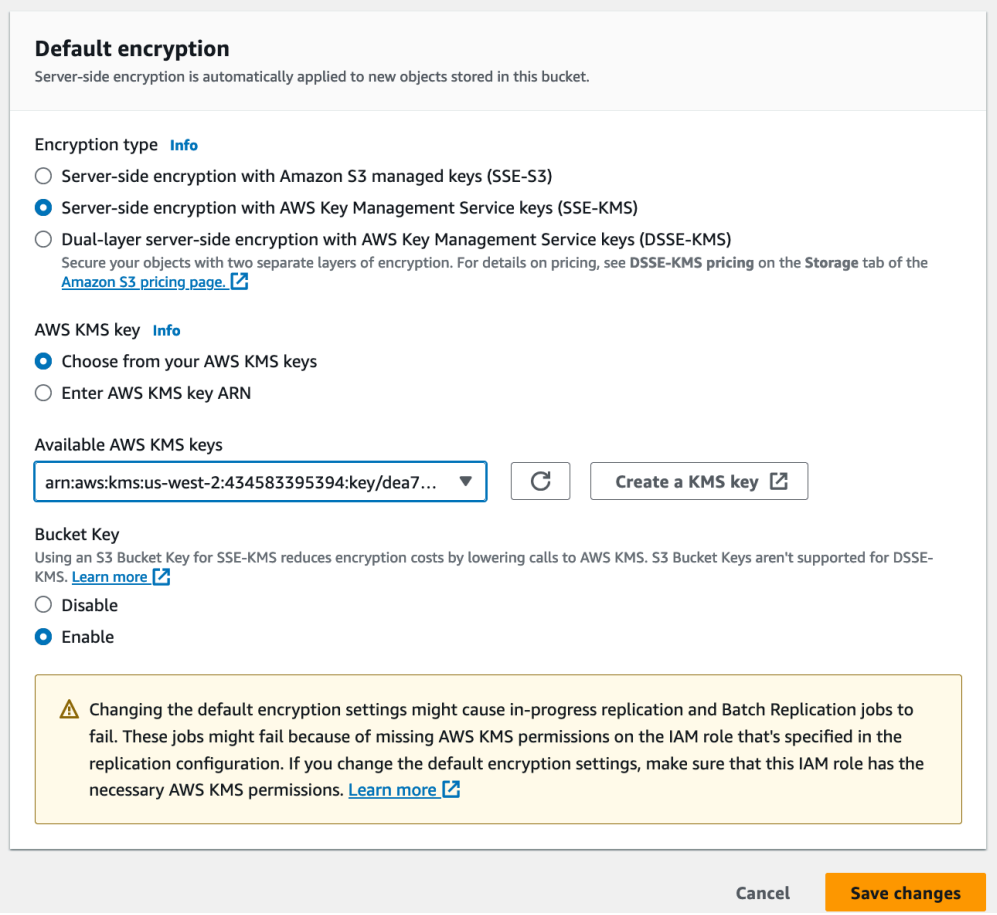
From the AWS console in the top search bar, search and select `S3`.

- Click the bucket name starting with `sid-security-xxxxxxxx`.
- Click on the `Properties` tab.
- Under `Default encryption` click `Edit`

Within the `Edit Default Encryption`, select the following:

- Encryption type, select **Server-side encryption with AWS Key Management Service keys (SSE-KMS)**
- AWS KMS key, select **Choose from your AWS KMS keys**
- Available AWS KMS keys, select **alias s3lab**

- Bucket Key, select **Enable**



Click `Save Changes`.

Run the following command in your EC2 SSH session to put a new text02 object to your bucket.

```
aws s3api put-object --key text02 --body textfile --bucket ${bucket}
```

The request should succeeded, notice how the server side encryption is `aws:kms` instead of AES256

```
[ec2-user@storage-workshop ~]$ aws s3api put-object --key text02 --body textfile --bucket ${bucket}
{
    "SSEKMSKeyId": "arn:aws:kms:us-west-2:329328435893:key/fa8ca065-5d27-4d56-a685-dfa9781dd7c3",
    "ETag": "\"8c48255c9fa02c784dec50297dcdf832\"",
    "ServerSideEncryption": "aws:kms"
}
[ec2-user@storage-workshop ~]$
```

Changing the bucket default encryption will affect any new objects written to the bucket, but existing objects will retain their existing encryption.

Run the following commands in your EC2 SSH session to compare your text01 and text02 to your bucket.

```
aws s3api head-object --key text01 --bucket ${bucket}
aws s3api head-object --key text02 --bucket ${bucket}
```

text01 was written before we changed the bucket default, hence it is AES256 (SSE-S3)

text02 was written after we changed the bucket default, confirming it has inherited the `aws:kms` KMS) encryption.

```
[ec2-user@storage-workshop ~]$ aws s3api head-object --key text01 --bucket ${bucket}
{
    "AcceptRanges": "bytes",
    "ContentType": "binary/octet-stream",
    "LastModified": "Mon, 09 Oct 2023 00:06:43 GMT",
    "ContentLength": 17,
    "ETag": "\"3ca451faac980583cffaadf8b63e6820\"",
    "ServerSideEncryption": "AES256",
    "Metadata": {}
}
[ec2-user@storage-workshop ~]$ aws s3api head-object --key text02 --bucket ${bucket}
{
    "AcceptRanges": "bytes",
    "ContentType": "binary/octet-stream",
    "LastModified": "Mon, 09 Oct 2023 00:15:07 GMT",
    "ContentLength": 17,
    "ETag": "\"8c48255c9fa02c784dec50297dcdf832\"",
    "ServerSideEncryption": "aws:kms",
    "SSEKMSKeyId": "arn:aws:kms:us-west-2:329328435893:key/fa8ca065-5d27-4d56-a685-dfa9781dd7c3",
    "Metadata": {}
}
[ec2-user@storage-workshop ~]$
```

For our scenario, let us assume we want certain prefixes to enforce SSE-KMS but still allow any encryption for other prefixes.

We will use a bucket policy to enforce that any new objects placed into our bucket under the `ssekms-only/` prefix must have SSE-KMS encryption.

From the AWS console in the top search bar, search and select `S3`.

- Click the bucket name starting with `sid-security-xxxxxxxx`.
- Click on the `Permissions` tab.
- Under `Bucket Policy` click `Edit`

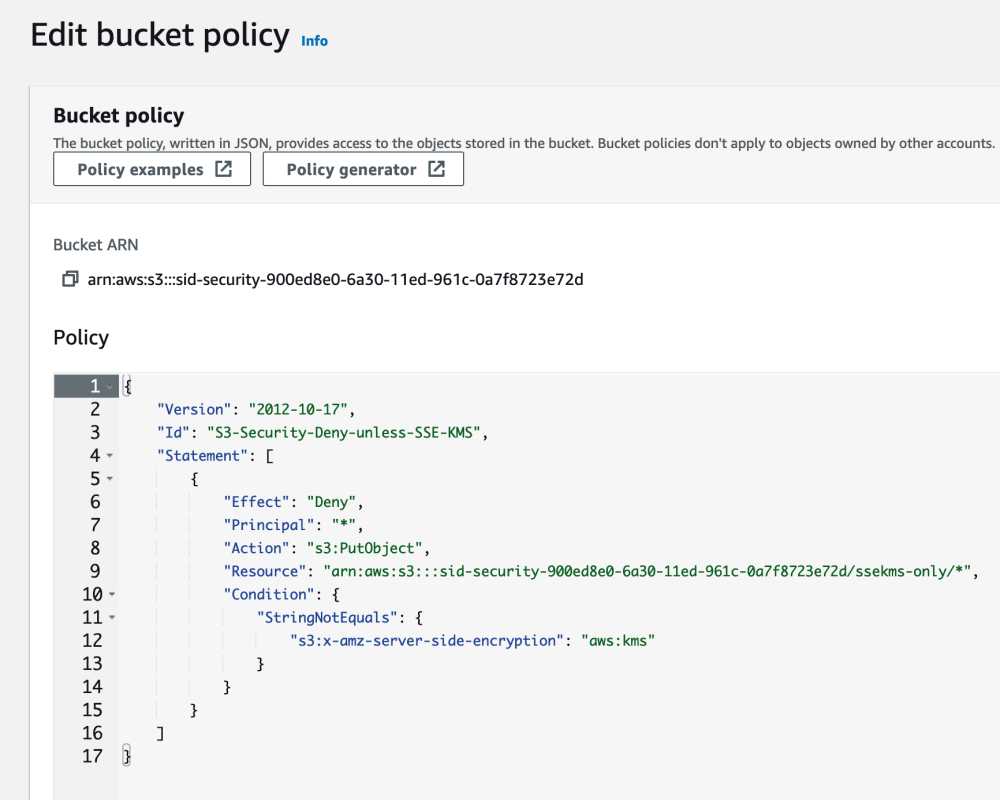
Remove the current policy and replace it with the bucket policy below and paste into the Bucket Policy Editor.

```
{
    "Id": "S3-Security-Deny-unless-SSE-KMS",
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Deny",
        "Principal": "*",
        "Action": "s3:PutObject",
        "Resource": "arn:aws:s3:::BUCKET_NAME/ssekms-only/*",
        "Condition": {
            "StringNotEquals": {
                "s3:x-amz-server-side-encryption": "aws:kms"
            }
        }
    }]
}
```

Replace `BUCKET_NAME` with the bucket name you copied to your text editor and click `Save changes`.

> Ensure your bucket policy keeps the prefix ssekms-only/* after the `BUCKET_NAME`

Your bucket policy should look similar to this example.



Run the following commands in your EC2 SSH session to put a new text03, text04, text05 object to your bucket.

```
aws s3api put-object --key text03 --body textfile --bucket ${bucket}
aws s3api put-object --key text04 --body textfile --server-side-encryption AES256 --bucket ${b
aws s3api put-object --key text05 --body textfile --server-side-encryption aws:kms --bucket ${
```

Behavior is as follows:

- Put object for text03 will succeed, and is aws:kms because it inherits the default encryption SSE-KMS as no encryption method is specified.
- Put object for text04 will succeed, and is AES256 because we specify it to use SSE-S3. Since server-side-encryption is explicitly configured, it will take precedence over the default bucket encryption SSE-KMS.
- Put object for text05 will succeed, and is aws:kms because we specify it to use SSE-KMS.

```
[ec2-user@storage-workshop ~]$
[ec2-user@storage-workshop ~]$ aws s3api put-object --key text03 --body textfile --bucket ${bucket}
{
    "SSEKMSKeyId": "arn:aws:kms:us-west-2:329328435893:key/fa8ca065-5d27-4d56-a685-dfa9781dd7c3",
    "ETag": "\"fd3416ddd85d49cdbb2af7a6311268fc\"",
    "ServerSideEncryption": "aws:kms"
}
[ec2-user@storage-workshop ~]$ aws s3api put-object --key text04 --body textfile --server-side-encryption AES256 --bucket ${bucket}
{
    "ETag": "\"3ca451faac980583cffaadf8b63e6820\"",
    "ServerSideEncryption": "AES256"
}
[ec2-user@storage-workshop ~]$ aws s3api put-object --key text05 --body textfile --server-side-encryption aws:kms --bucket ${bucket}
{
    "SSEKMSKeyId": "arn:aws:kms:us-west-2:329328435893:key/fa8ca065-5d27-4d56-a685-dfa9781dd7c3",
    "ETag": "\"b2ff160f750328185e97461fc2c7fb74\"",
    "ServerSideEncryption": "aws:kms"
}
[ec2-user@storage-workshop ~]$
```

Let's create new objects under the prefix `ssekms-only/*` to see how the bucket policy changes the behavior.

```
aws s3api put-object --key ssekms-only/text06 --body textfile --bucket ${bucket}
aws s3api put-object --key ssekms-only/text07 --body textfile --server-side-encryption AES256
aws s3api put-object --key ssekms-only/text08 --body textfile --server-side-encryption
```

Behavior in the `ssekms-only/*` prefix is as follows:

- Put object for text06 will fail, although we have default encryption as SSE-KMS the bucket policy requires explicit definition for SSE-KMS.

- Put object for text07 will fail, since AES256 will use SSE-S3 and we only permit SSE-KMS encryption in this prefix per the bucket policy.
- Put object for text08 will succeed, since SSE-KMS encryption is explicitly defined and allowed by the bucket policy.

```
[ec2-user@storage-workshop ~]$ aws s3api put-object --key ssekms-only/text06 --body textfile --bucket ${bucket}

An error occurred (AccessDenied) when calling the PutObject operation: User: arn:aws:sts::092526071050:assumed-ro
le/ssw-base-template-SidS3AccessRole-xgqCqtNjPrb8/i-06ee8c015c1d6e501 is not authorized to perform: s3:PutObject
on resource: "arn:aws:s3:::sid-security-ca51ea00-acec-11ef-ab19-02dc4668901f/ssekms-only/text06" with an explicit
 deny in a resource-based policy
[ec2-user@storage-workshop ~]$
[ec2-user@storage-workshop ~]$ aws s3api put-object --key ssekms-only/text07 --body textfile --server-side-encryp
tion AES256 --bucket ${bucket}

An error occurred (AccessDenied) when calling the PutObject operation: User: arn:aws:sts::092526071050:assumed-ro
le/ssw-base-template-SidS3AccessRole-xgqCqtNjPrb8/i-06ee8c015c1d6e501 is not authorized to perform: s3:PutObject
on resource: "arn:aws:s3:::sid-security-ca51ea00-acec-11ef-ab19-02dc4668901f/ssekms-only/text07" with an explicit
 deny in a resource-based policy
[ec2-user@storage-workshop ~]$
[ec2-user@storage-workshop ~]$
[ec2-user@storage-workshop ~]$ aws s3api put-object --key ssekms-only/text08 --body textfile --server-side-encryp
tion aws:kms --bucket ${bucket}
{
    "SSEKMSKeyId": "arn:aws:kms:us-west-2:092526071050:key/7e293131-c1df-4d65-9e76-eb76d3521750",
    "ETag": "\"1a385d1dbe41b2d50fac538422d750a5\"",
    "ServerSideEncryption": "aws:kms"
}
[ec2-user@storage-workshop ~]$
```

The goal of this exercise was to demonstrate how to use either default encryption and bucket policies to ensure you can apply and enforce encryption to your buckets.

In summary, you have successfully:

- Configured default encryption SSE-KMS on an entire bucket.
- Configured a bucket policy to enforce SSE-KMS on a specific prefix within that bucket.

Previous Next