AWS Academy Cloud Security Foundations

Lab 3.1 Report: Using Resource-Based Policies to Secure an S3 Bucket

Ketaj Youssef

3ACI Info

1. Lab Overview and Objectives

The objective of this lab was to understand the difference between IAM identity-based policies and resource-based policies (specifically S3 bucket policies). We explored how these policies interact to define fine-grained access control.

Key Goals:

- Distinguish between identity-based and resource-based policies.
- Demonstrate how an IAM user can assume an IAM role to change permissions.
- Analyze how policies affect console visibility and actions.

Architecture

Starting Architecture:

The lab began with a devuser (IAM user) who is a member of the `DeveloperGroup`.

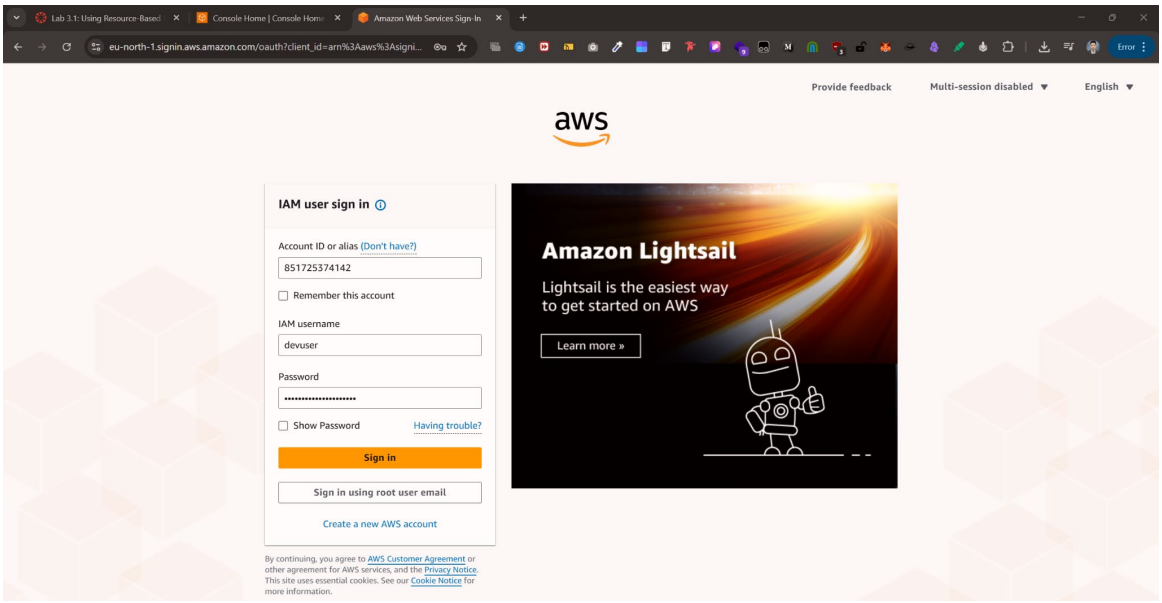
Target Architecture:

By the end, we utilized specific IAM roles (`BucketAccessRole` and `OtherBucketAccessRole`) to interact with restricted S3 buckets.

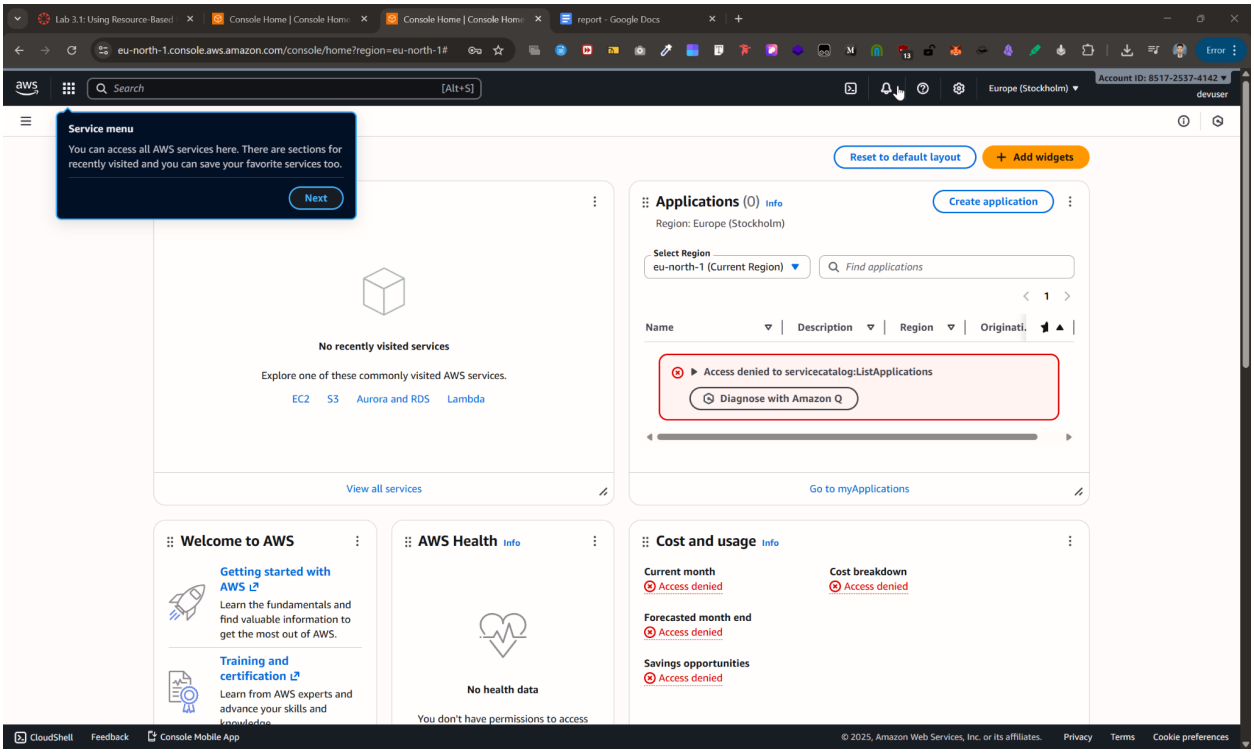
2. Detailed Task Walkthrough

Task 1: Accessing the Console

- **Action:** Logged into the AWS Management Console using the provided devuser credentials.



- **Observation:** The login was successful, placing us in the AWS Console Home.

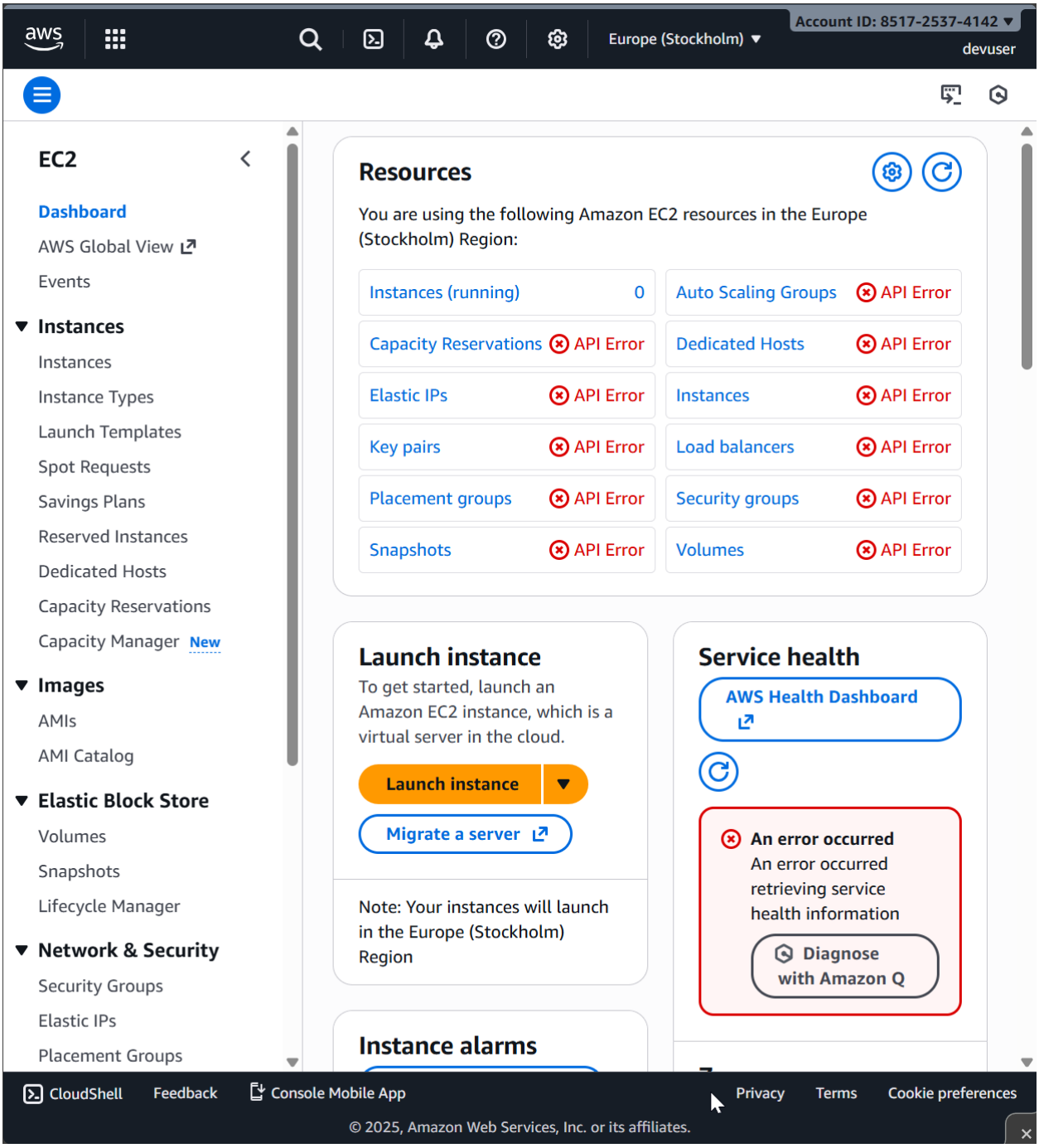


Task 2: Attempting Read -Level Access to AWS Services

Action 1: Accessing Amazon EC2

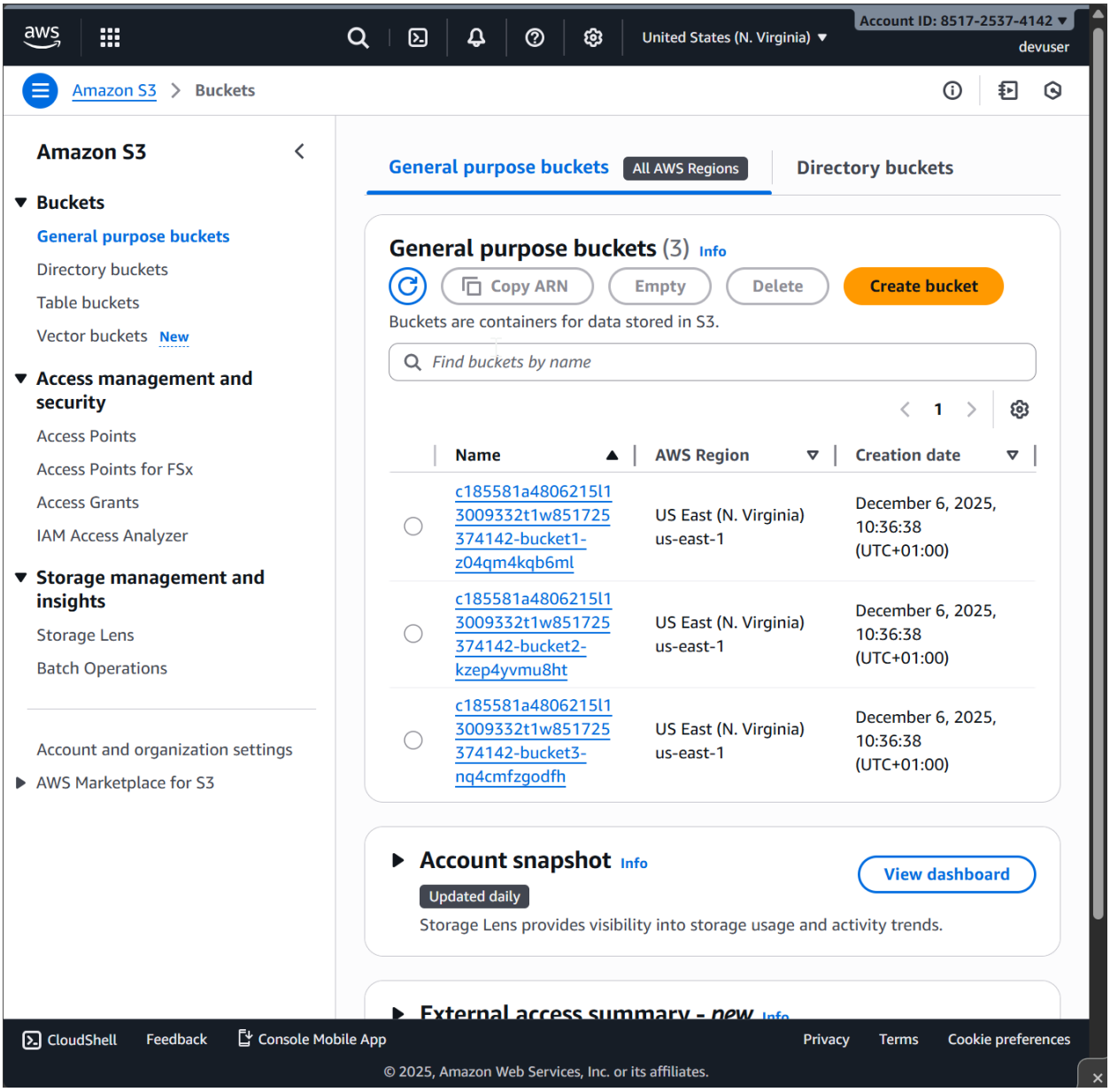
- **Step:** Navigated to the EC2 Dashboard to view instances.
- **Observation:** The console displayed multiple "API Error" messages. We were unable to list instances, view instance types, or see any EC2 resources.

- **Reason:** The devuser does not have the necessary permissions (e.g., `ec2:DescribeInstances`) attached to their identity policy.



Action 2: Accessing Amazon S3

- **Step:** Navigated to the S3 Console.



- **Observation:** We could see the list of buckets (bucket1, bucket2, bucket3), but the "Access" column showed "Insufficient permissions" or "Error".
- **Reason:** The devuser has permission to `ListAllMyBuckets`, allowing them to see the bucket names, but lacks permissions to query the specific configuration or public access settings of those buckets.

Task 3: Analyzing the Identity -Based Policy

Action:

- Navigated to the IAM Console -> Users -> devuser -> Groups -> DeveloperGroup ->

Permissions.

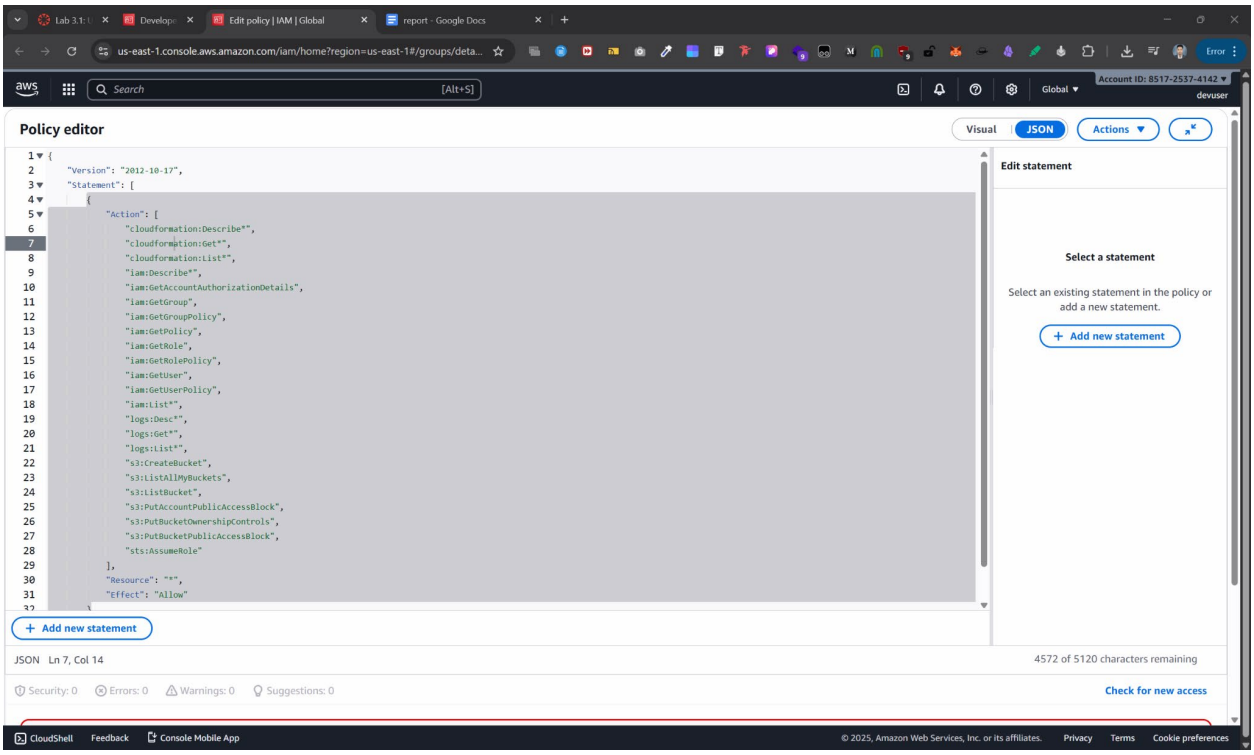
- Opened the `DeveloperGroupPolicy` to inspect the JSON.

Policy Analysis:

The policy attached to our group was an Identity-Based Policy.

- **Allowed Actions:** `s3:ListAllMyBuckets`, `s3:CreateBucket`, and `s3:ListBucket`.
- **Denied/Missing Actions:** It did *not* explicitly allow `ec2:*` actions (explaining the EC2 errors) and did not allow `s3:PutObject` globally.

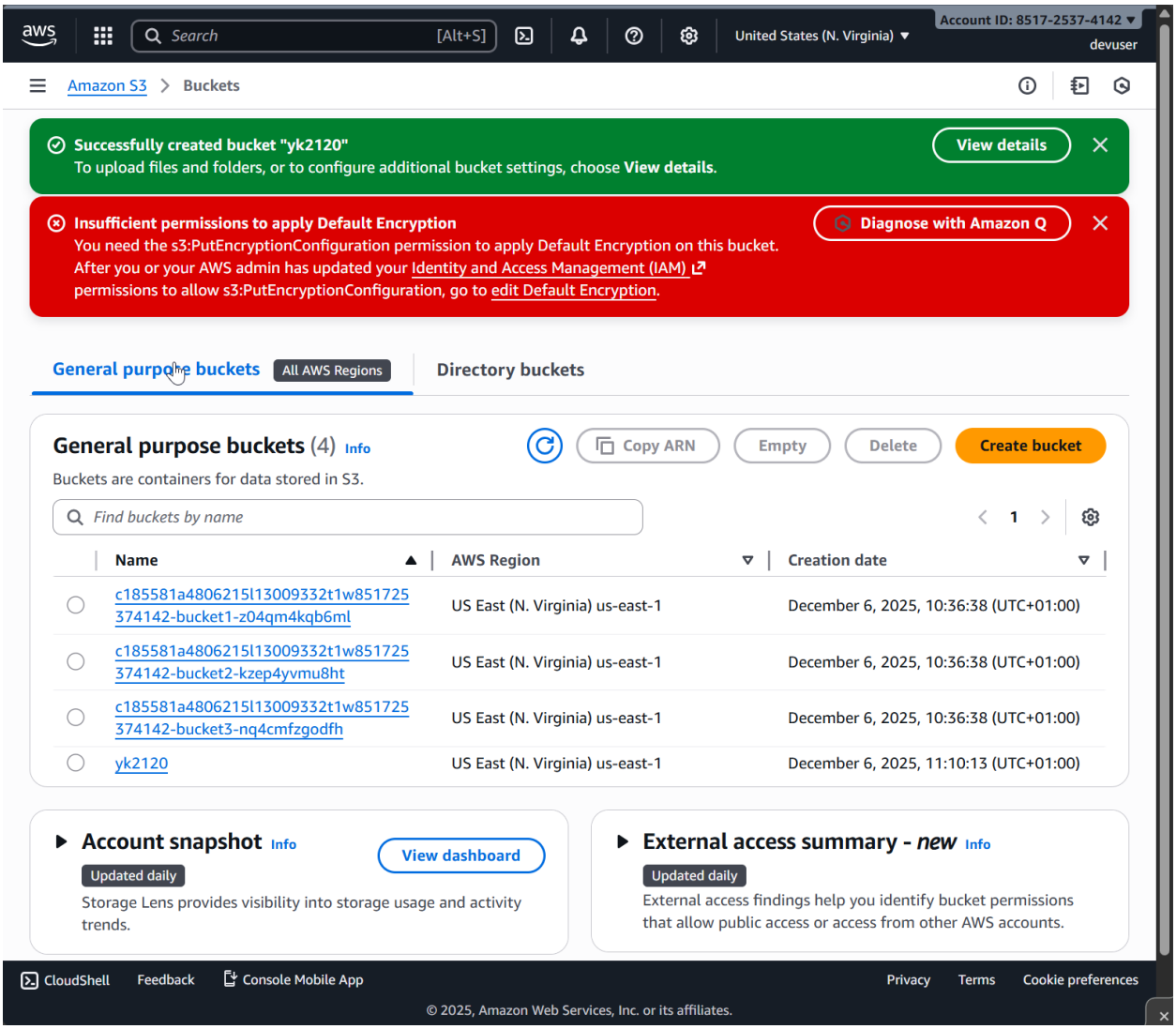
This explains why we could see the bucket names (List) but failed to perform other administrative tasks.



Task 4: Attempting Write -Level Access to AWS Services

Action 1: Creating a Bucket

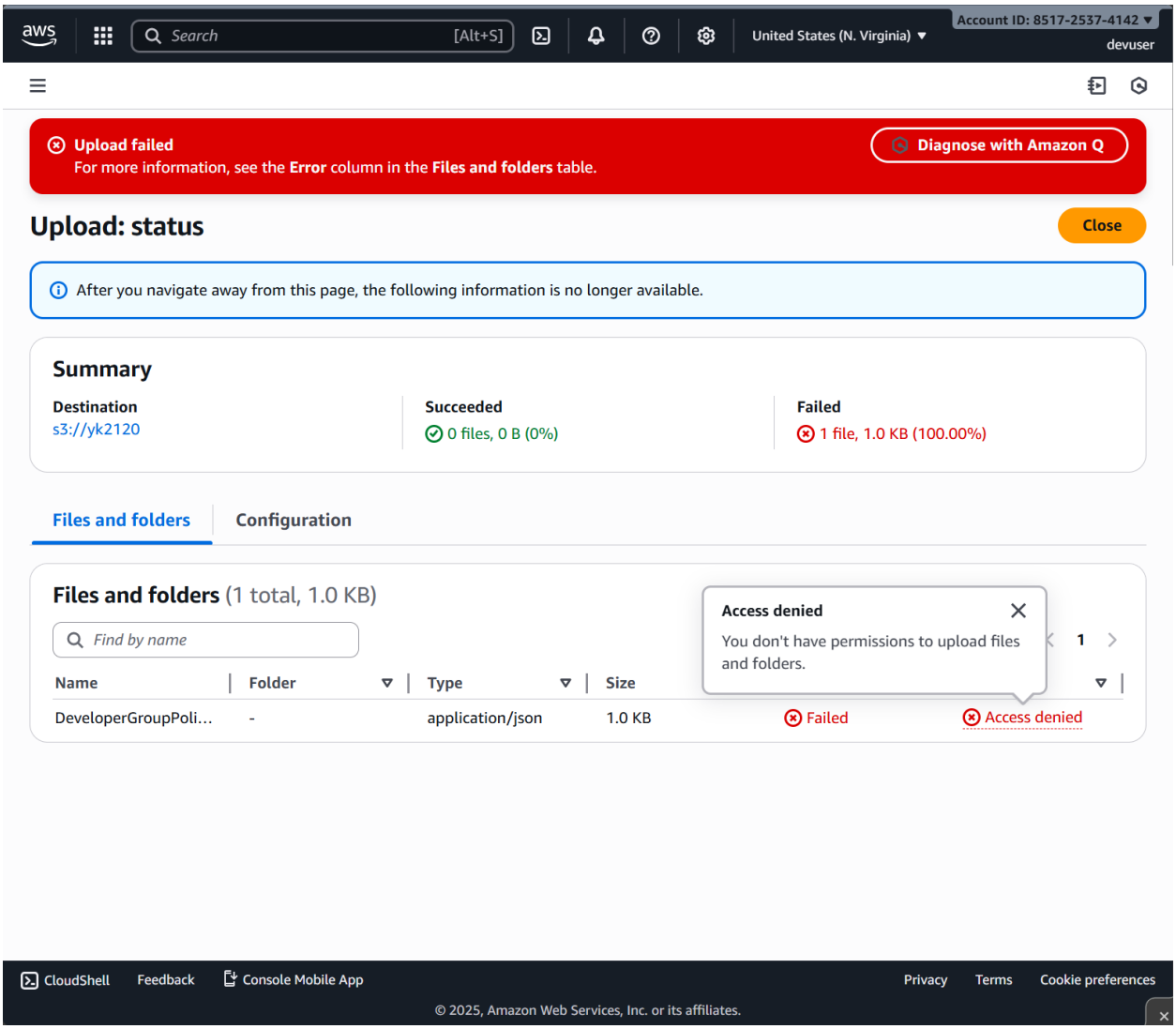
- **Step:** Clicked "Create bucket" and named it `yk2120`.



- **Result: Success** .
- **Reason:** The `DeveloperGroupPolicy` explicitly allowed `s3:CreateBucket`.

Action 2: Uploading an Object

- **Step:** Opened the newly created bucket and attempted to upload Image1.jpg.
- **Result: Failed** (Upload failed message).
- **Reason:** While we can create the bucket (owner), the IAM policy attached to devuser does not grant `s3:PutObject` permission. In AWS, permission to create a resource does not automatically imply permission to modify its contents if the policy is restrictive.



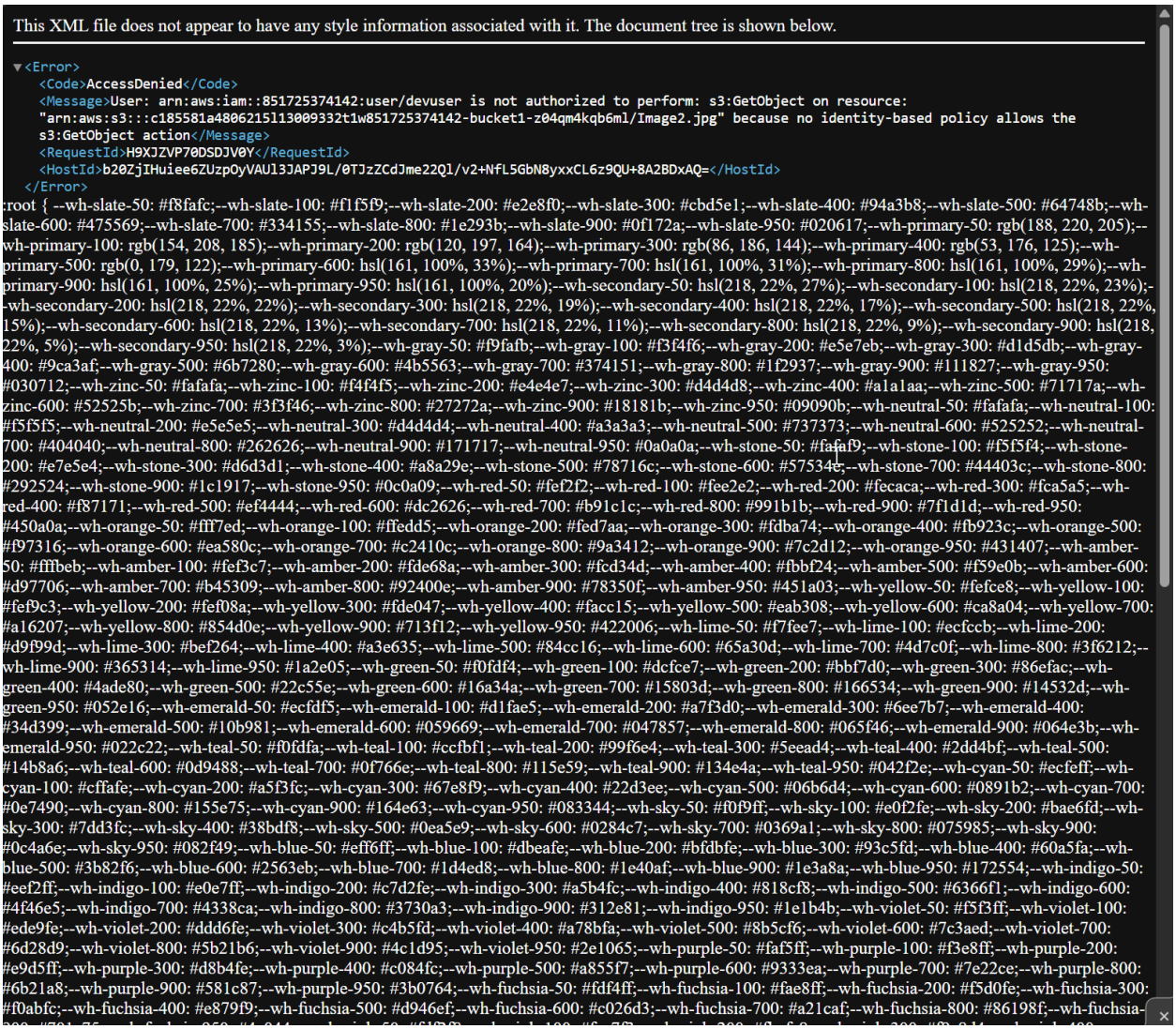
Task 5: Assuming an IAM Role

Testing Download Before Assuming Role

Action:

- Navigated to `bucket1` and attempted to download an existing file.

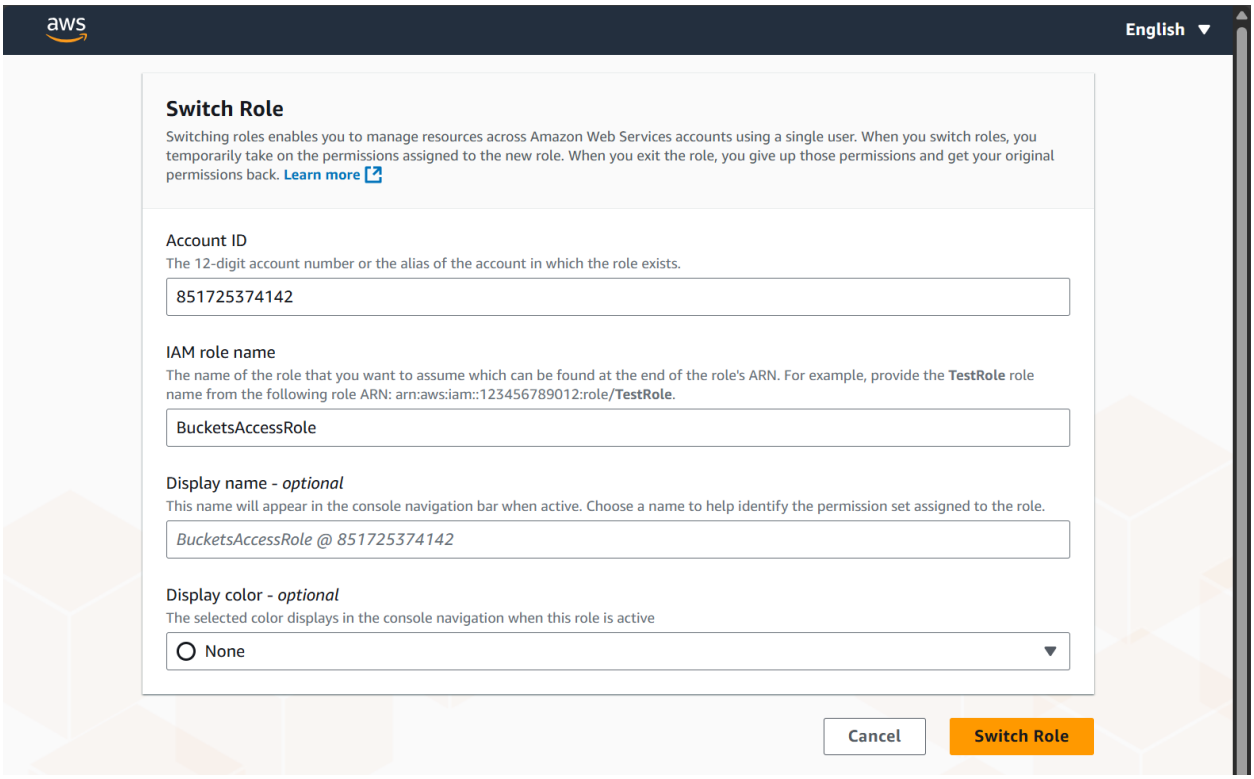
Result: Failed (Access Denied error).



Reason:

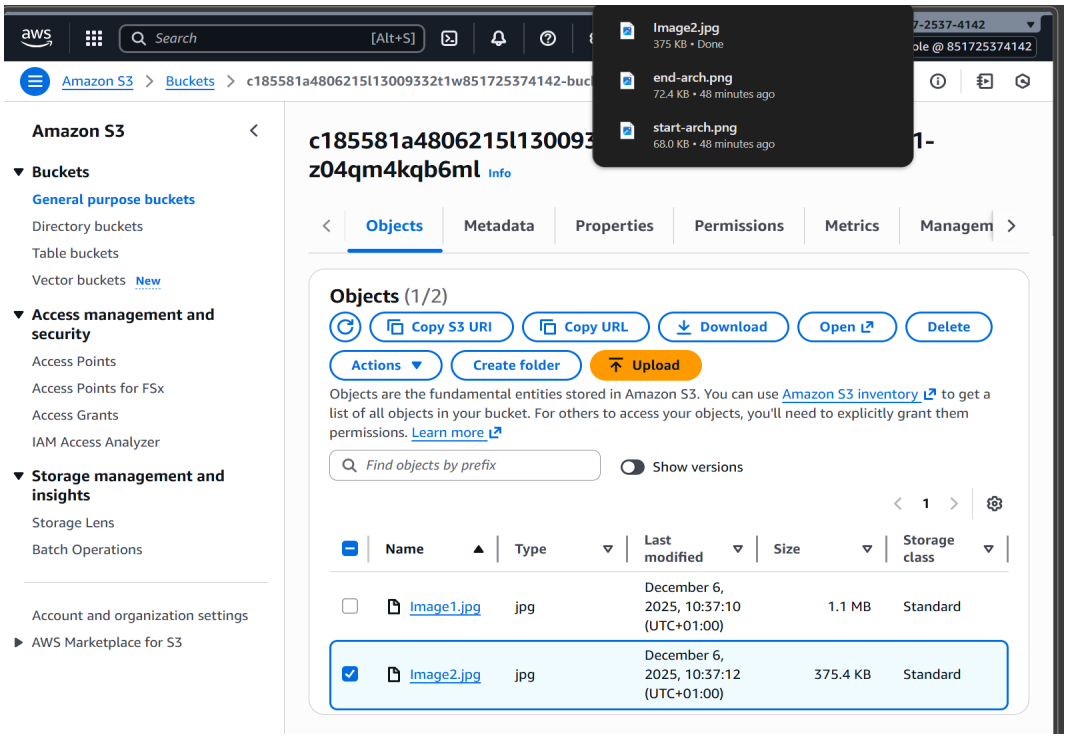
The `devuser`'s underlying Identity-Based Policy (DeveloperGroupPolicy) allowed `s3:ListAllMyBuckets` and `s3:ListBucket` but did **not** include the necessary `s3:GetObject` permission required to download an object from the bucket. This confirms that the default user permissions restrict read access to object data, even though they can see the list of buckets.

Action:

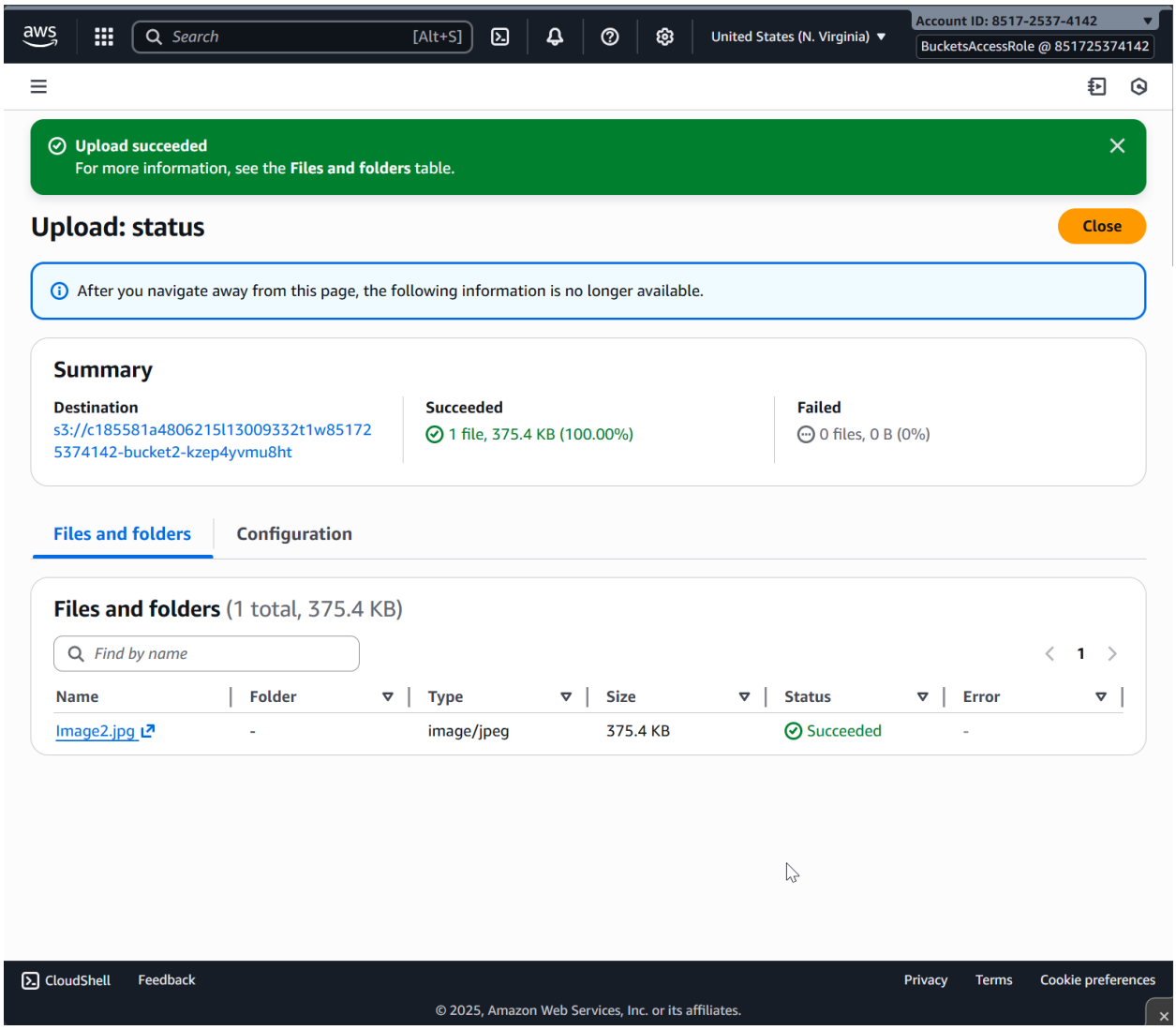


- Selected the user dropdown in the top right corner and clicked **Switch Role** .
- Entered the account ID and the role name:`BucketsAccessRole`.
- **Display Change:** The user identity in the top right changed to `BucketsAccessRole`.

Testing Access:

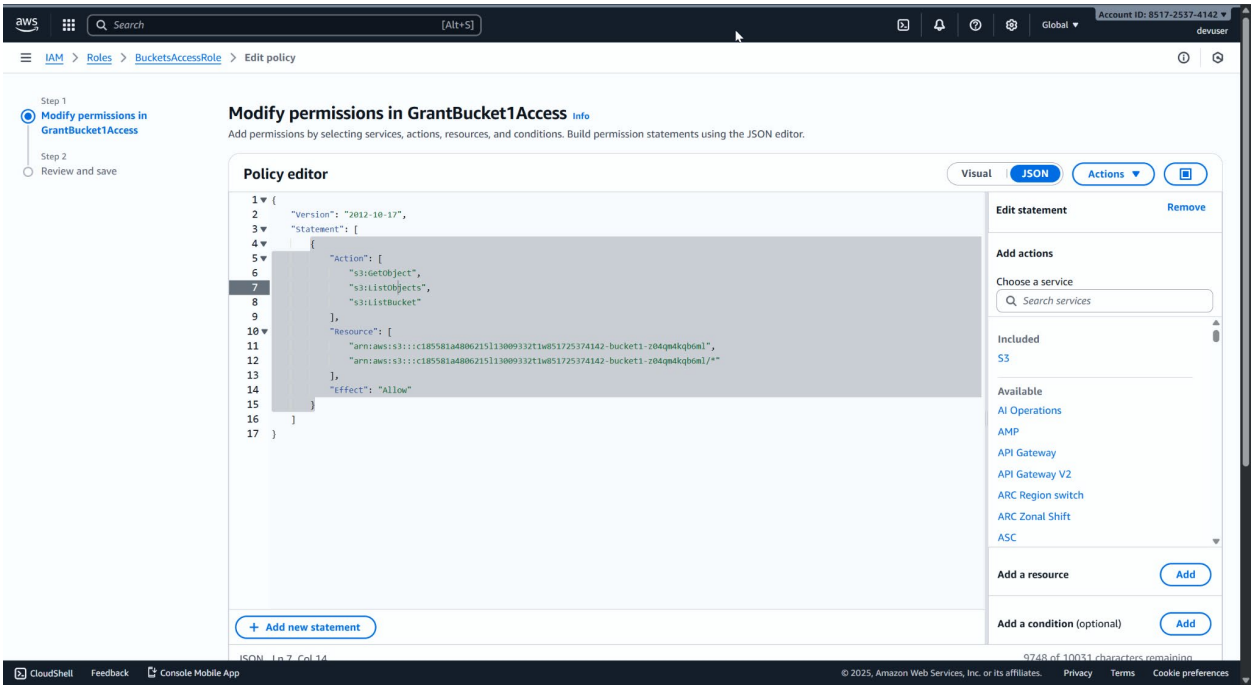


1. **Bucket 1:** We could **download** objects.
2. **Bucket 2:** We could **upload** objects (Permissions allowed `s3:PutObject` on bucket2).



Analysis:

When we assumed the role, we temporarily gave up the devuser permissions and adopted the `BucketsAccessRole` permissions. This role had a specific policy allowing read/write access to these specific buckets, which devuser did not have natively.



Task 6: Understanding Resource -Based Policies

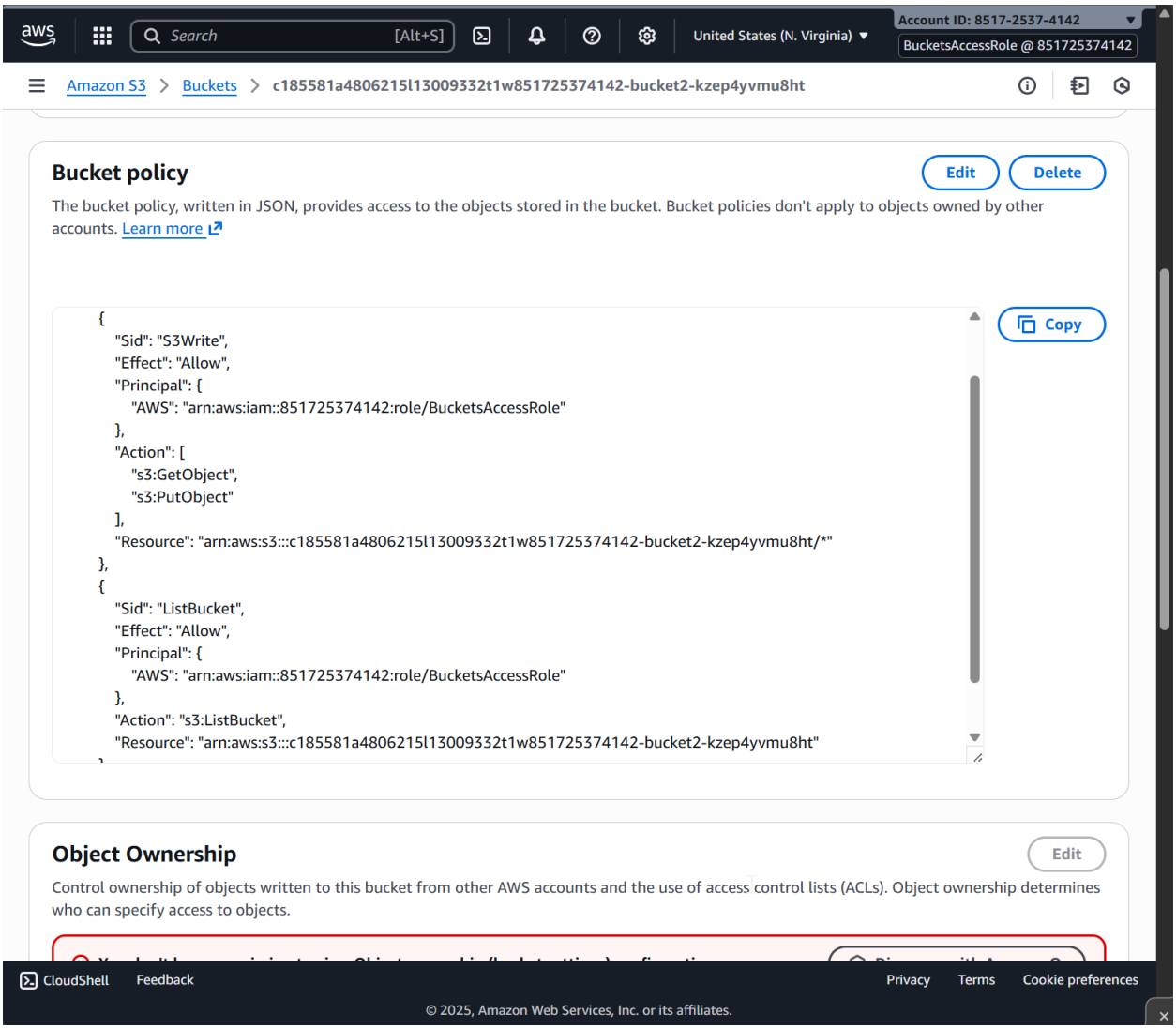
Action:

- While using the bucket-access-role, we navigated to **Bucket 2 -> Permissions** tab.
- Viewed the **Bucket Policy** .

Policy Analysis:

This is a Resource-Based Policy. Unlike the identity policy attached to a user, this policy is attached directly to the S3 bucket.

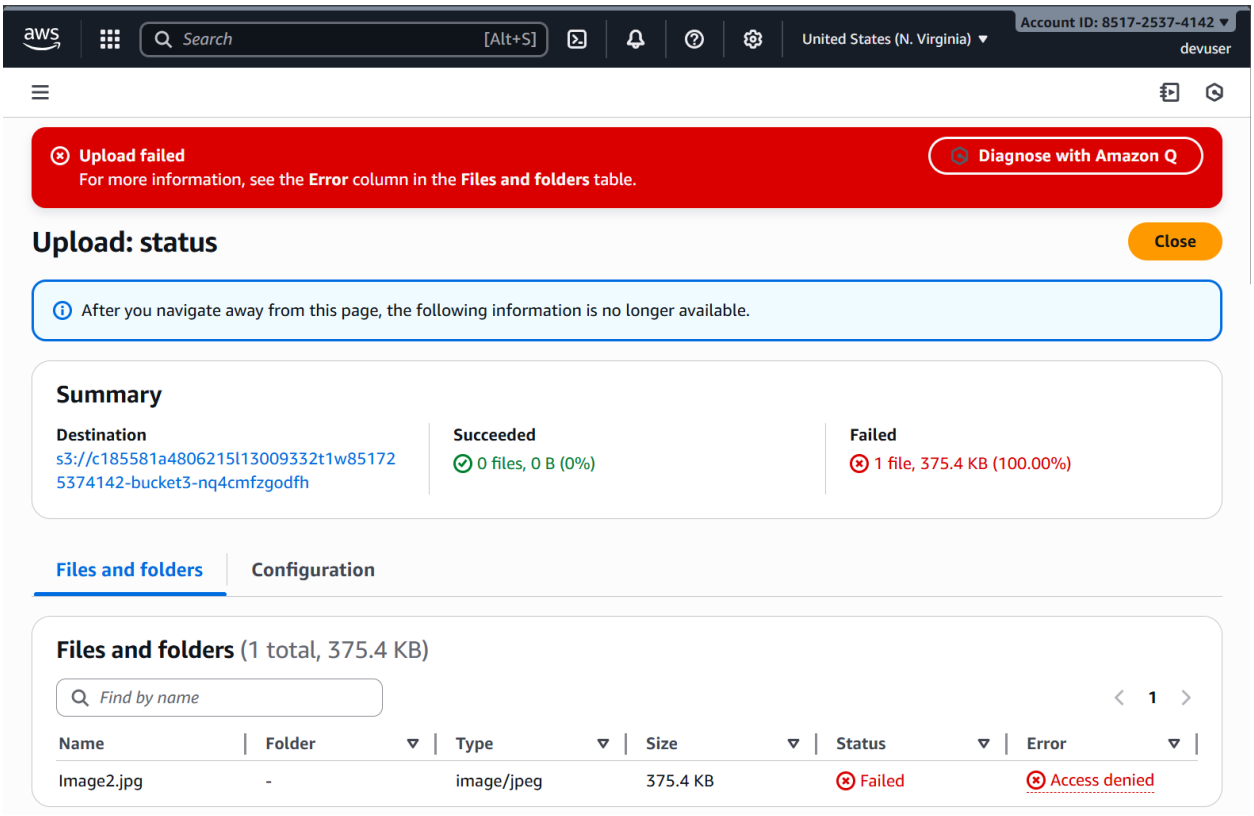
- It explicitly listed the `BucketsAccessRole` as a Principal.
- This confirms that access can be granted either by the user's IAM policy OR by the resource's policy (if in the same account), or requires both (if cross-account, though in this lab it was primarily about how the role was authorized).



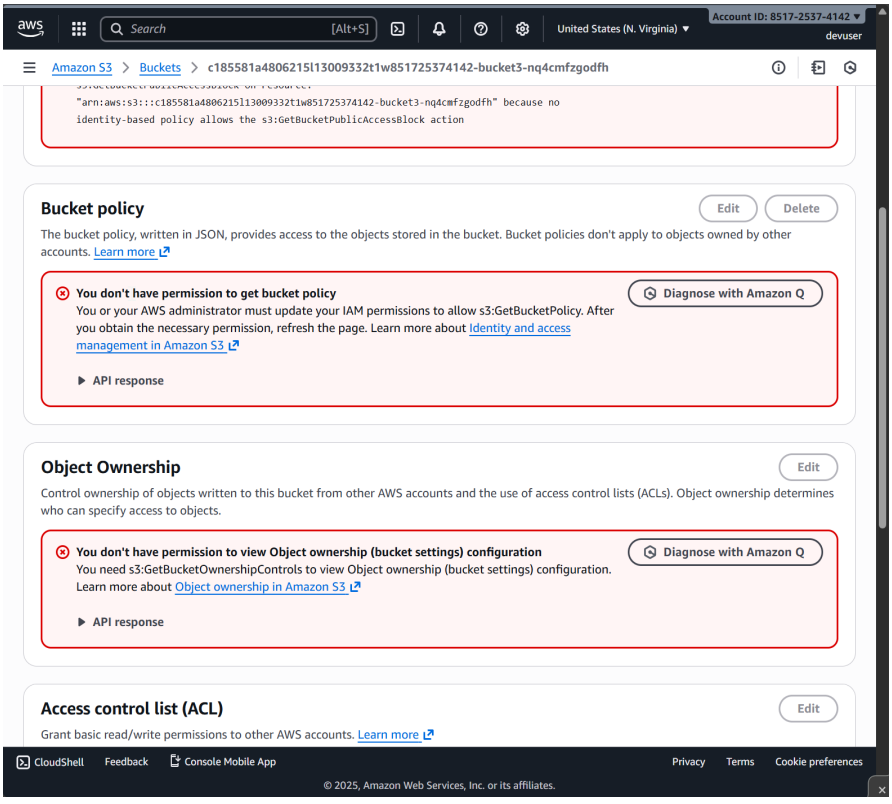
3. Challenge Task: Uploading to Bucket 3

Objective: Upload Image2.jpg to bucket3.

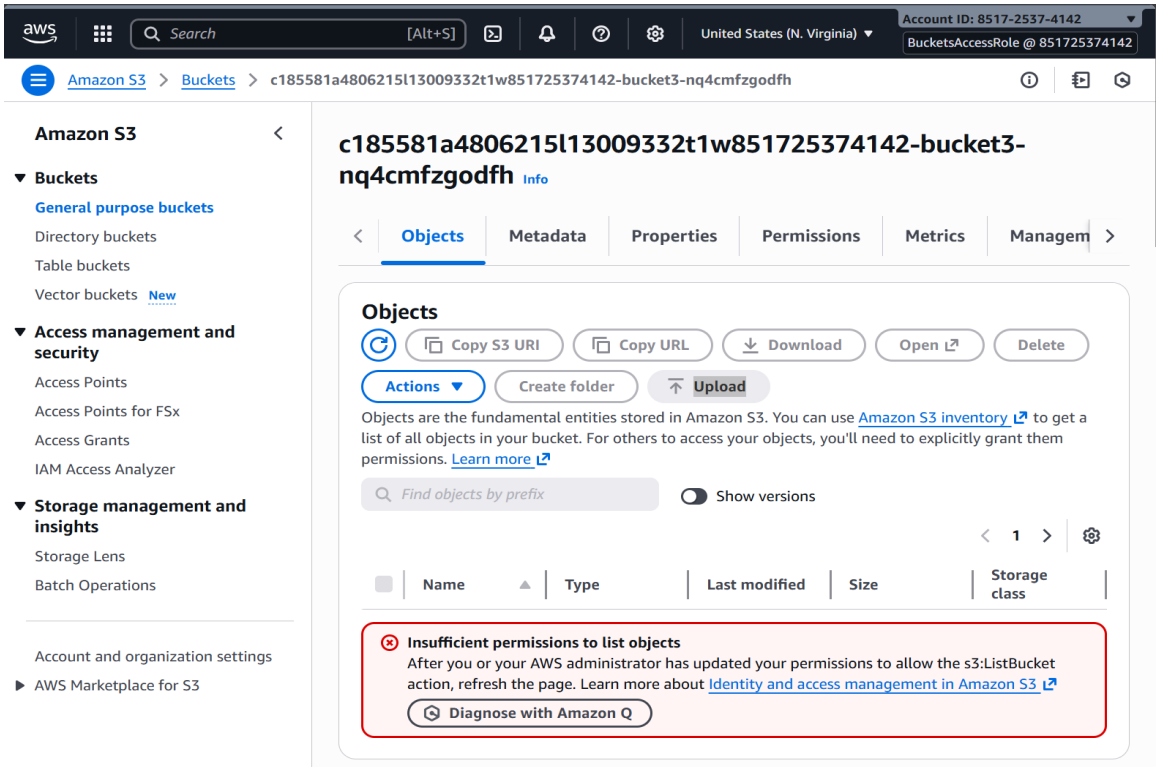
Attempt 1: As devuser



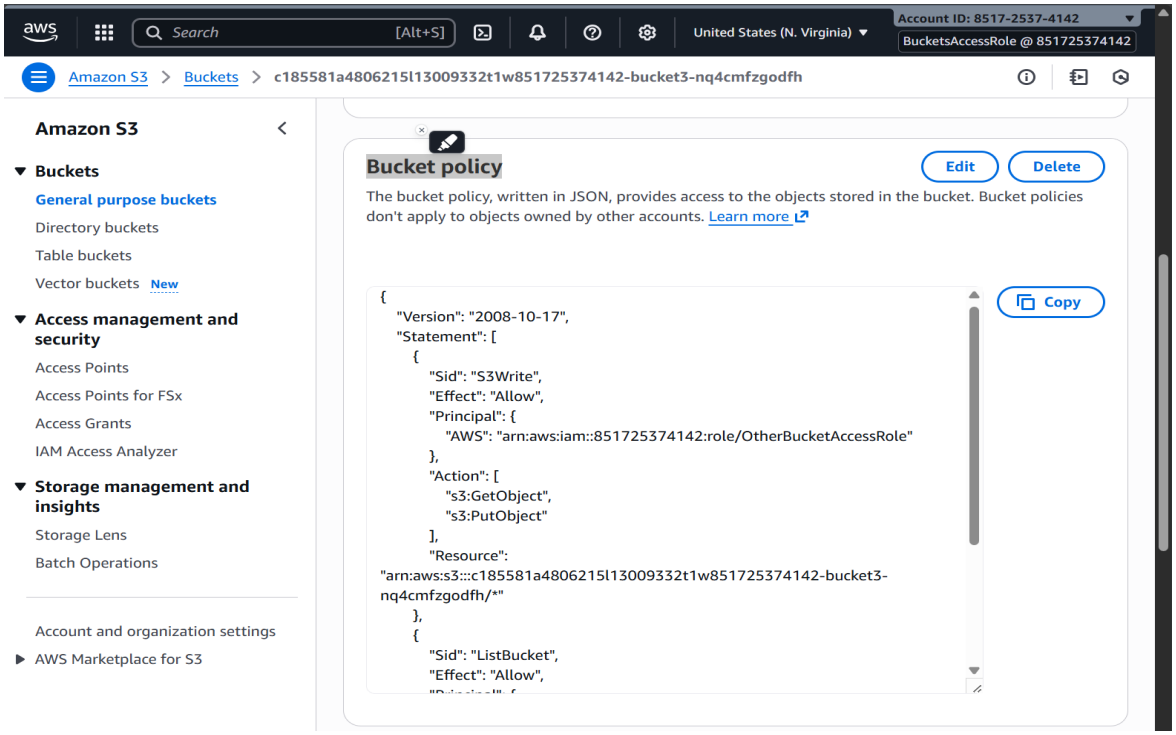
- **Result:** Failed. No write permissions and devuser can't view bucket policy.



Attempt 2: As BucketAccessRole

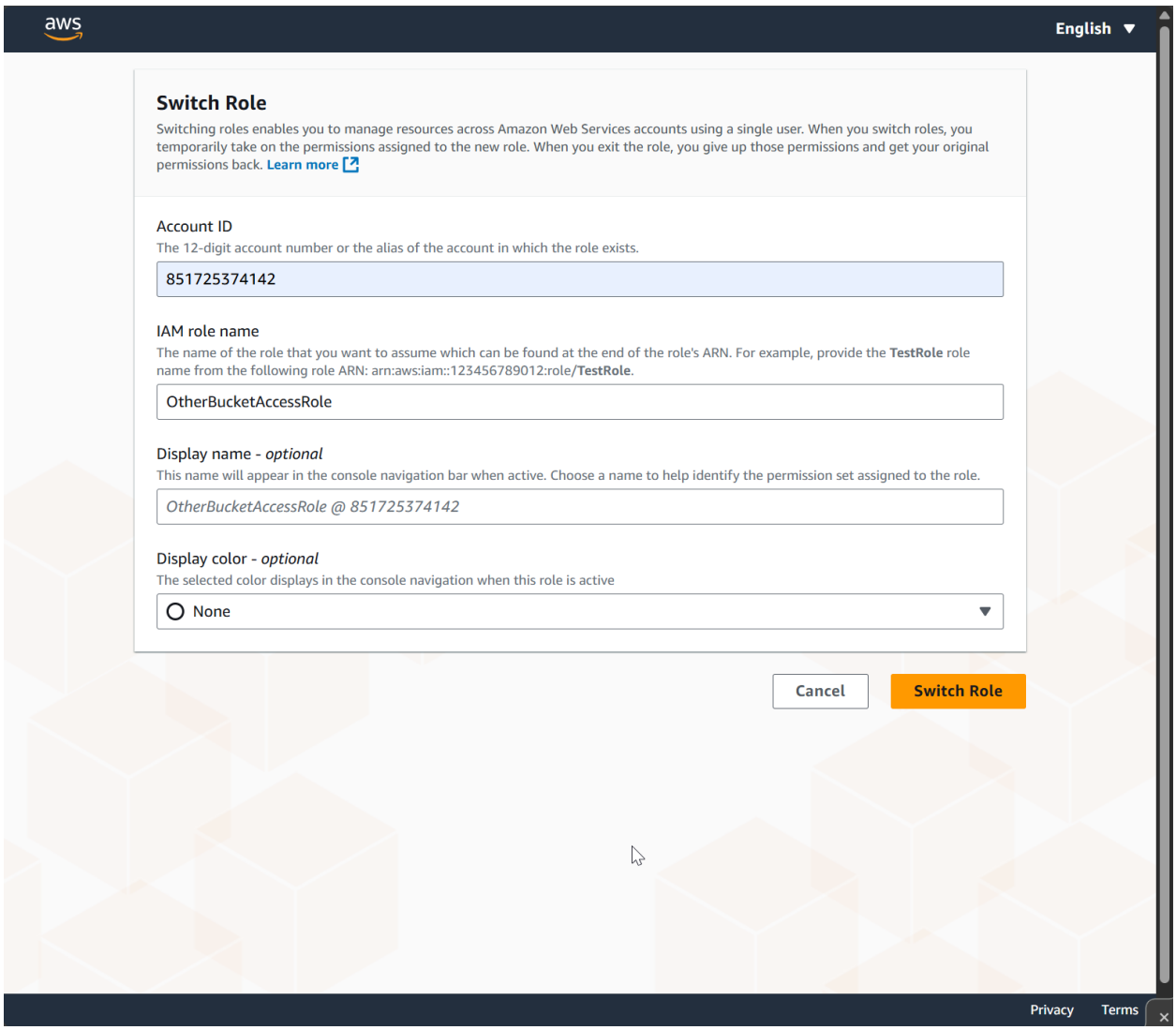


- **Result:** Failed. The role's policy did not include permissions for bucket3 but we can read the bucket policy .



Solution:

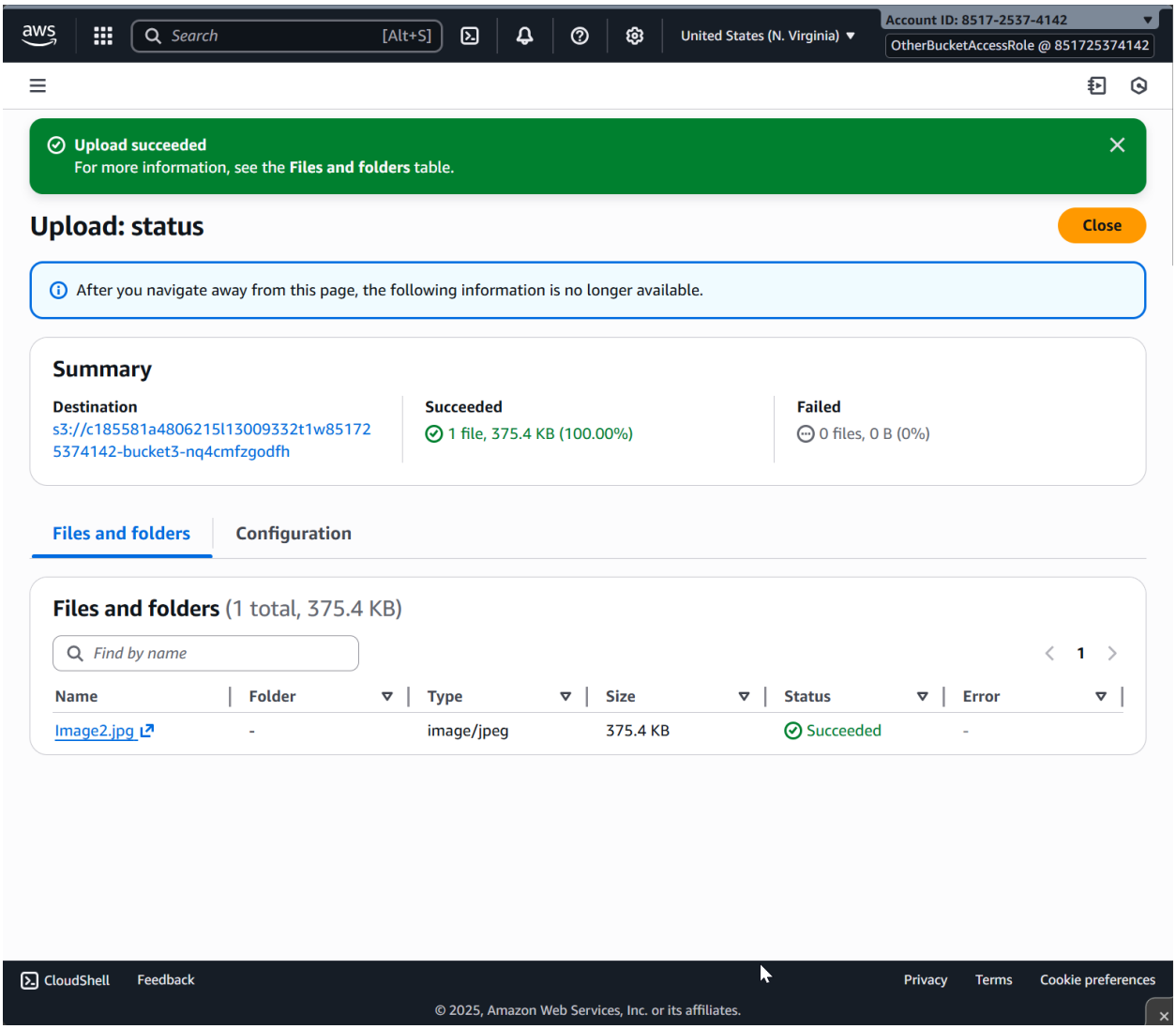
1. We identified a second role available in the account: `OtherBucketAccessRole`.
2. We used the **Switch Role** feature again to switch from `BucketsAccessRole` to `OtherBucketAccessRole`.



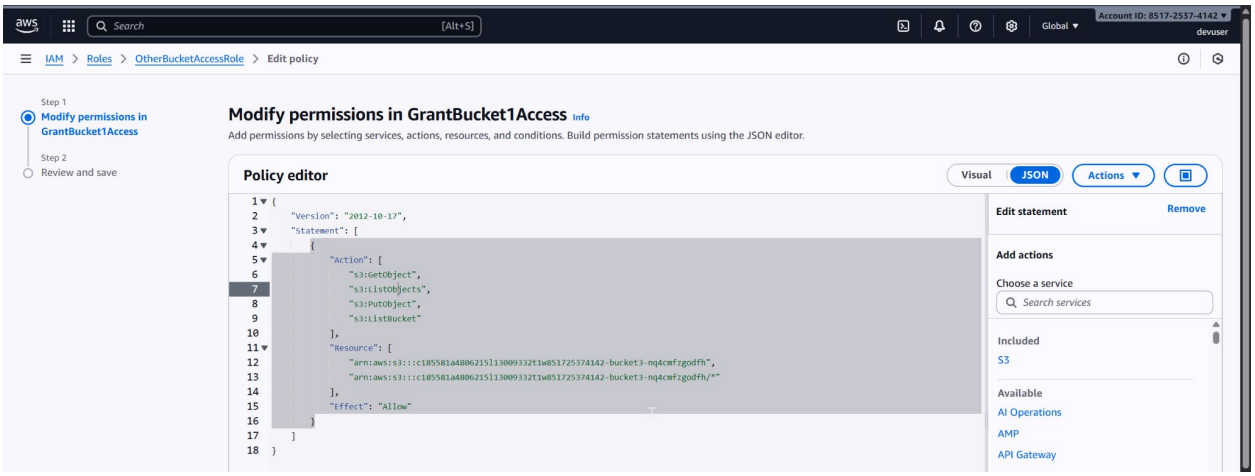
3. Navigated to bucket3.
4. Clicked **Upload** , selected Image2.jpg, and submitted.
5. **Result: Success** .

Explanation:

The `OtherBucketAccessRole` had a policy that specifically allowed `s3:PutObject` on the `bucket3` resource. By assuming this specific role, we gained the necessary credentials to complete the challenge.



Analysis of OtherBucketAccessRole Policy



Inferred Policy Analysis:

Component	Observation/Inference
Principal	The devuser was allowed to assume this role (defined in the role's Trust Policy).
Allowed Actions (S3)	include `s3:PutObject` and `s3:GetObject` and `s3:ListBucket`.
Resource Scope	The permissions were explicitly scoped to `bucket3` (arn:aws:s3:::c185581a4806215l13009332t1w851725374142-bucket3-nq4cmfzgodfh).
Key Difference	Unlike the `BucketsAccessRole`, this role granted the specific write permissions necessary to interact with `bucket3`, confirming the principle of least privilege where each role serves a targeted function.

This analysis confirms that the role's identity policy was the key mechanism enabling the successful upload to the restricted `bucket3`.

4. Conclusion

In this lab, we successfully demonstrated the flexibility of AWS access control.

1. **Identity Policies** attached to devuser provided basic entry access but restricted powerful actions like creating EC2 instances or writing to arbitrary buckets.
2. **IAM Roles** allowed us to elevate permissions temporarily to perform specific tasks (managing specific buckets) without permanently granting those privileges to the user.
3. **Resource Policies** on the S3 buckets acted as a second layer of access control, defining exactly which principals (users/roles) could interact with the bucket data.