Lab: EC2 Instance Management and IAM Roles

DS 5220 — Advanced Cloud Computing

Faizan Khan

February 6, 2026

Part 1: Web Server Instance

Configuration Summary

The following EC2 instance was launched and configured to serve a web page using nginx:

Setting	Value
Instance Name	webserver-lab
AMI	Ubuntu Server 24.04 LTS
Instance Type	t3.micro
Public IPv4	18.205.192.194
Security Group	webserver-sg (SSH from My IP, HTTP from 0.0.0.0/0)
Key Pair	ds5220

Screenshot: nginx Welcome Page

The default nginx welcome page was successfully served at http://18.205.192.194:

Figure 1: Default nginx welcome page confirming the web server is running and accessible via HTTP.

Checkpoint: HTTPS Access Attempt

Accessing https://18.205.192.194 (with HTTPS) results in a connection timeout, as shown below:

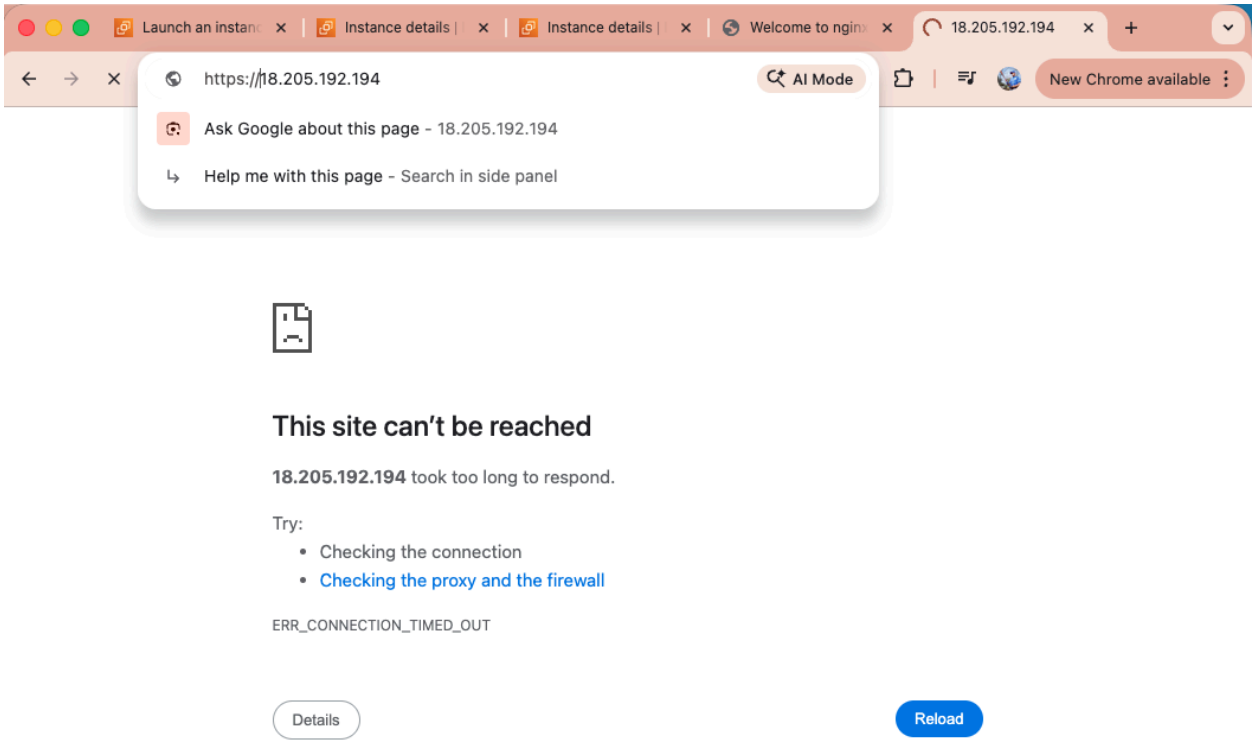

Figure 2: ERR_CONNECTION_TIMED_OUT when attempting HTTPS access.

This fails because the instance is configured to serve HTTP traffic on port 80 only. HTTPS requires port 443 to be open in the security group, a valid TLS/SSL certificate installed on the server, and an nginx server block configured for SSL termination. Since none of these are in place, the browser cannot establish a secure connection and the request times out.

Part 2: EC2 with IAM Role and S3 Access

Configuration Summary

A second EC2 instance was launched with an attached IAM role to test scoped S3 access:

Setting	Value
Instance Name	s3-access-lab
Public IPv4	18.215.181.54
IAM Role	EC2-S3-Limited-Access
IAM Policy	S3-Limited-Bucket-Access
S3 Bucket	lab-bucket-xdy6sg

IAM Role Verification

Running aws sts get-caller-identity confirmed the instance was assuming the correct role:

```
  "UserId": "AROAWNV3KJL77XH37ZLSA:i-07c6a602207db0eb6",
  "Account": "441700731647",
  "Arn":
"arn:aws:sts::441700731647:assumed-role/EC2-S3-Limited-Access/i-07c6a6022
07db0eb6"
```

Step 7: Initial Permission Tests

With the initial policy granting only s3:ListBucket, s3:PutObject, and s3:GetObject on the specific bucket:

Action	Result	Reason
Upload file to bucket	Success	s3:PutObject granted
List objects in bucket	Success	s3:ListBucket granted
Download file from bucket	Success	s3:GetObject granted
List all buckets (aws s3 ls)	AccessDenied	s3:ListAllMyBuckets not granted
Delete object from bucket	AccessDenied	s3:DeleteObject not granted

Screenshot: AccessDenied Errors (Before Policy Update)

The following screenshot shows the full sequence of initial permission tests, including the AccessDenied errors for listing all buckets and deleting an object:

Figure 3: Initial S3 tests showing successful uploads/downloads and AccessDenied for ListAllMyBuckets and DeleteObject.

Checkpoint: Why Did Listing All Buckets Fail?

The command aws s3 ls calls the ListAllMyBuckets API, which is an account-level S3 action requiring s3:ListAllMyBuckets with "Resource": "*". The initial policy only granted permissions scoped to arn:aws:s3:::lab-bucket-xdy6sg and its objects. Because no identity-based policy allowed the ListAllMyBuckets action, AWS correctly returned an AccessDenied error.

Checkpoint: Why Did Deleting an Object Fail?

The delete command failed because the initial policy did not include s3:DeleteObject in the allowed actions. The error message explicitly stated that no identity-based policy allows the s3:DeleteObject action on the specified resource. The missing permission was s3:DeleteObject on arn:aws:s3:::lab-bucket-xdy6sg/*.

Steps 8–9: Updated Policy and Retesting

The IAM policy was updated to add s3:DeleteObject on the bucket's objects and a new statement granting s3:ListAllMyBuckets on "*". After the policy propagated, both previously denied actions succeeded:

Action	Result	Reason
List all buckets (aws s3 ls)	Success	s3:ListAllMyBuckets now granted
Upload file to bucket	Success	s3:PutObject still granted
Delete object from bucket	Success	s3:DeleteObject now granted

Screenshot: Successful Output After Policy Update

The following screenshots show the successful operations after updating the IAM policy:

Figure 4: Delete operation succeeding after adding s3:DeleteObject to the policy.

Figure 5: Successful upload, list, and delete operations after the full policy update with all three statements.

Reflection Questions

1. Why Restrict SSH Access to Your IP?

Restricting SSH to your own IP dramatically reduces the attack surface of your instance. When SSH is open to 0.0.0.0/0, anyone on the internet can attempt to connect to port 22, which invites automated scans, brute-force attacks, and exploitation of any future SSH vulnerabilities. Even with strong key-based authentication, exposing a critical management interface to the entire internet increases risk unnecessarily.

By locking SSH to a specific trusted IP, you add a network-level layer of defense on top of authentication. This follows a defense-in-depth strategy: the security group acts as a firewall, the SSH key provides authentication, and OS-level hardening provides a third layer. Rather than relying on a single control, you combine multiple controls so that a failure in one does not compromise the system.

2. Difference Between the Two Resource ARNs

These two ARNs target different levels of the S3 resource hierarchy:

arn:aws:s3:::lab-bucket-<id> refers to the bucket itself as a container. Bucket-level actions such as s3:ListBucket (listing the contents of the bucket) and s3:GetBucketLocation operate at this level. In the lab policy, s3:ListBucket was correctly attached to this ARN because listing is a container-level operation.

arn:aws:s3:::lab-bucket-<id>/* refers to the objects stored inside the bucket. Actions that interact with individual files—such as s3:GetObject, s3:PutObject, and s3:DeleteObject—require the /* suffix. In the lab policy, these object-level actions were attached to this ARN because they operate on items within the container, not the container itself.

S3 treats buckets and objects as distinct resource types, so IAM policies must explicitly scope permissions to the correct ARN for each type of action.

3. Why Does ListAllMyBuckets Require "Resource": "*"?

The s3:ListAllMyBuckets action does not operate on a single bucket—it returns a list of all buckets owned by the account. Because it is an account-level service action rather than a resource-specific one, there is no individual bucket ARN that can meaningfully scope it. AWS models this as a global S3 operation, so the only valid resource is "*", meaning the action applies across the S3 service for the account.

This was demonstrated in the lab: the initial policy only granted permissions on arn:aws:s3:::lab-bucket-xdy6sg, so running aws s3 ls returned AccessDenied. The action only succeeded after adding a statement with s3:ListAllMyBuckets on "*".

4. Principle of Least Privilege

The principle of least privilege states that any identity—whether a user, role, service, or instance—should be granted only the minimum permissions necessary to perform its required tasks, and nothing more. This minimizes the blast radius if credentials are compromised and limits the potential for accidental misuse or misconfiguration.

This lab demonstrated least privilege in several concrete ways. The IAM role was initially scoped to a single bucket with only three allowed actions (ListBucket, PutObject, GetObject). Attempts to list all buckets or delete objects were correctly denied. When additional capabilities were needed, the policy was expanded incrementally—adding only s3:DeleteObject and s3:ListAllMyBuckets—rather than granting broad access like

s3:*. The entire approach started from zero permissions and added only what was required, rather than starting with full access and dialing back.

5. Read-Only Access to All S3 Buckets

To give an EC2 instance read-only access to all S3 buckets in the account, the IAM policy would need three statements:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["s3:ListAllMyBuckets"],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": ["s3:ListBucket"],
            "Resource": "arn:aws:s3:::*"
        },
        {
            "Effect": "Allow",
            "Action": ["s3:GetObject"],
            "Resource": "arn:aws:s3:::*/*"
        }
    ]
}
```

The first statement allows listing all buckets in the account. The second allows listing the contents of any individual bucket. The third allows reading any object in any bucket. Critically, no write or delete actions (PutObject, DeleteObject) are included, ensuring the access is truly read-only. Alternatively, AWS provides the managed policy AmazonS3ReadOnlyAccess, which could be attached directly to the role.

Real-World Scenario

A common real-world use case is a data processing application running on EC2 that needs to read input files from a specific S3 bucket and write processed results back to that same bucket. For example, a nightly ETL pipeline might pull raw server logs from s3://app-logs-bucket/raw/, transform and aggregate the data on EC2, and store the processed output in s3://app-logs-bucket/processed/. Assigning an IAM role with tightly scoped S3 permissions ensures the instance can access only the required bucket and only through the approved actions (GetObject, PutObject, ListBucket). This minimizes risk: if the instance is compromised, the attacker cannot access other buckets, delete production data, or escalate to other AWS services.

Cleanup

All resources created during this lab were terminated and deleted upon completion:

- EC2 instances (webserver-lab and s3-access-lab) terminated
- S3 bucket (lab-bucket-xdy6sg) emptied and deleted
- IAM policy (S3-Limited-Bucket-Access) deleted
- IAM role (EC2-S3-Limited-Access) deleted
- Security group (webserver-sg) deleted