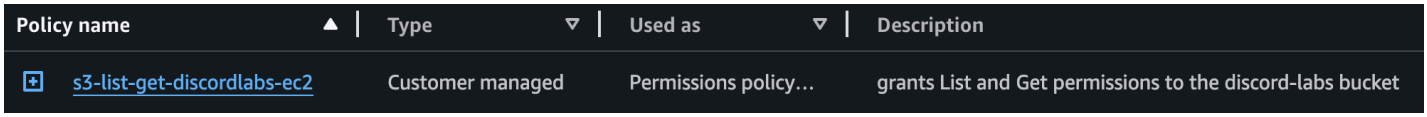
Lab 3 Submission - b.ho.o
07/10/2025

4.1. Create a Custom IAM Policy

Created IAM policy which grants list and get permissions to lab bucket

As seen In AWS Console:



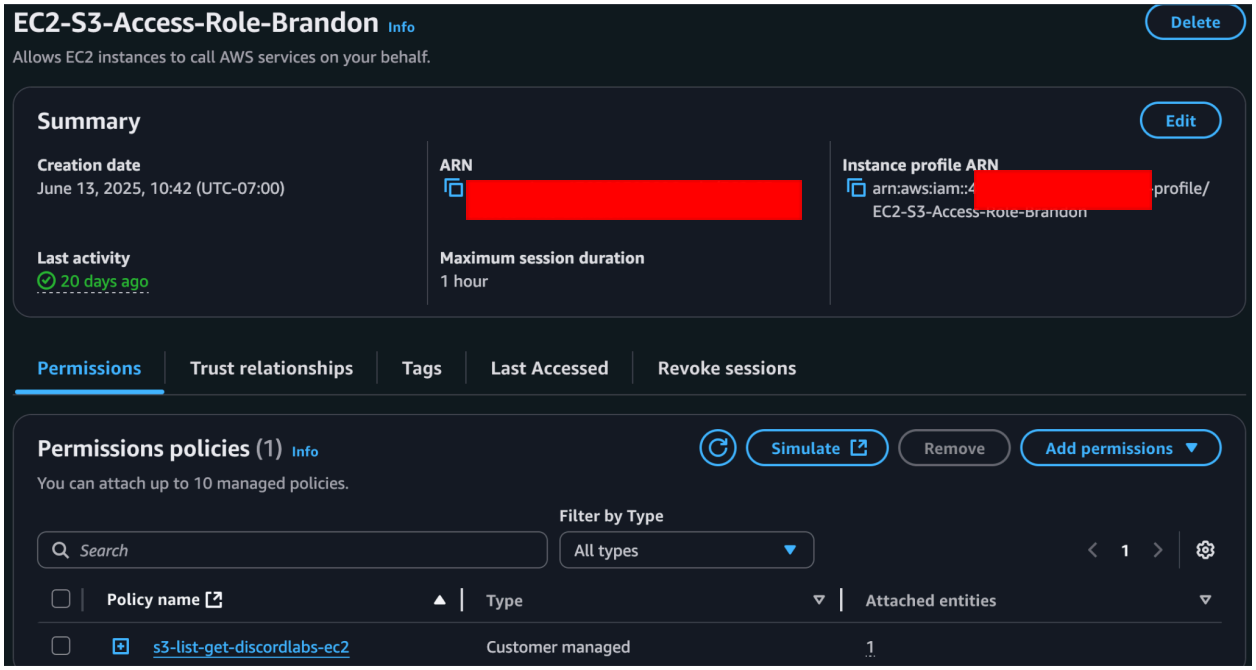
Policy name	Type	Used as	Description
s3-list-get-discordlabs-ec2	Customer managed	Permissions policy...	grants List and Get permissions to the discord-labs bucket

Policy:

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

4.2. Create an IAM Role for EC2

Created IAM role and attached policy to it



4.3. Attach the Role to Your EC2 Instance

I previously attached a role to the EC2 in lab2, and needed to replace it with the newly created IAM role above. First, had to get the existing AssociationId (using describe-iam-instance-profile-associations)

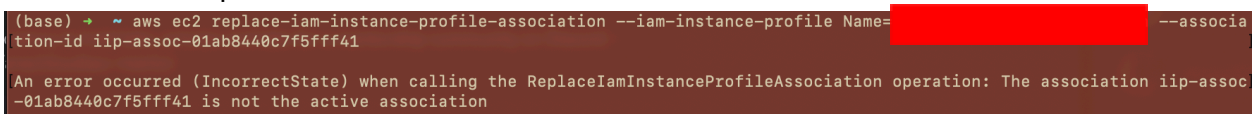
```
aws ec2 describe-iam-instance-profile-associations \
  --filters Name=instance-id,Values=i-0d0a94a78842272ff
```

After getting the association ID, attempted the below options to replace but kept running into errors from both methods:

- Replace-iam-instance-profile-association
- Replacing instance profile from AWS console

Error from AWS Console replacement:

Error from CLI replacement:



I ended up having to fully dissociate the role using:

aws ec2 disassociate-iam-instance-profile

Then associated the newly created role with:

Aws ec2 associate-iam-instance-profile

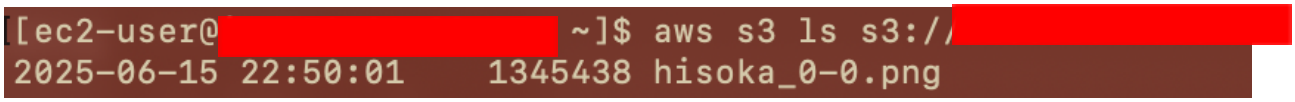
https://docs.aws.amazon.com/cli/latest/reference/ec2/associate-iam-instance-profile.html

4.4. Verify Access from the EC2 Instance

Validating access to my bucket:

```
ssh -i /path/to/key.pem ec2-user@<EC2_PUBLIC_IP>
```

After SSHing in, was able to list bucket contents



As I made incremental changes to policy while working through the lab, I used simulate-principal-policy to test access (e.g. in example below, testing ability to get specific object in bucket)

```
aws iam simulate-principal-policy \
  --policy-source-arn arn:aws:iam::476569303606:role/EC2-S3-Access-Role-Brandon \
  --action-names s3:GetObject \
  --resource-arns arn:aws:s3:::discord-labs/hisoka_0-0.png
```

5. Stretch Goals

Create bucket policy that blocks all public access but allows the previously created IAM role

In the lab .md, a policy statement that denies insecure transport (denies all access over HTTP) is provided. I combined the provided policy statement with a statement that allows my IAM role to access the bucket

- Principal: the IAM role
- Action: all s3 commands
    - I later tweaked this to GetObject and ListBucket actions
- Resource: lab bucket
- Condition: requiring secure transport (i.e. over HTTPS), this is redundant when paired with the first policy statement
    - I later removed this condition just for clarity, but unsure if this is best practice

I included Sids for each statement for clarity, and made the changes from the AWS console (bucket -> permissions -> bucket policy -> edit -> pasted in contents from VSCode file shown below)

Restrict to your IP Address:
I created a statement which allows requests coming from my IP to list or read objects in the bucket.

I then adapted it for the existing statement (which allowed access to my IAM role), by changing the Principal from * to my IAM role in order to tighten up access.

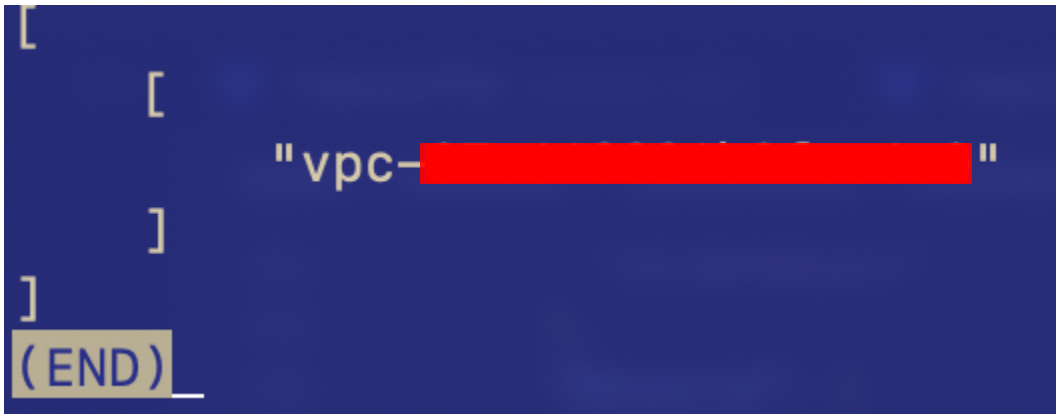
To do the same from CL (taken from lab instructions):

Save the JSON above to a file `bucket-ip-policy.json`, then run:

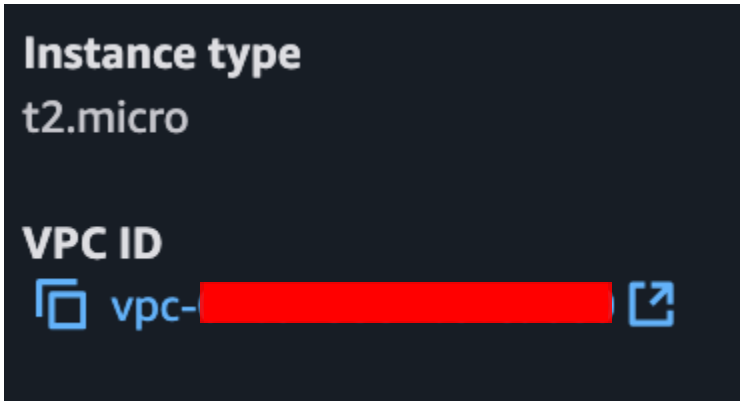
```
aws s3api put-bucket-policy \ --bucket your-bucket-name \
--policy file://bucket-ip-policy.json
```

Experiment with requiring MFA or VPC conditions
To determine VPC ID from CLI:

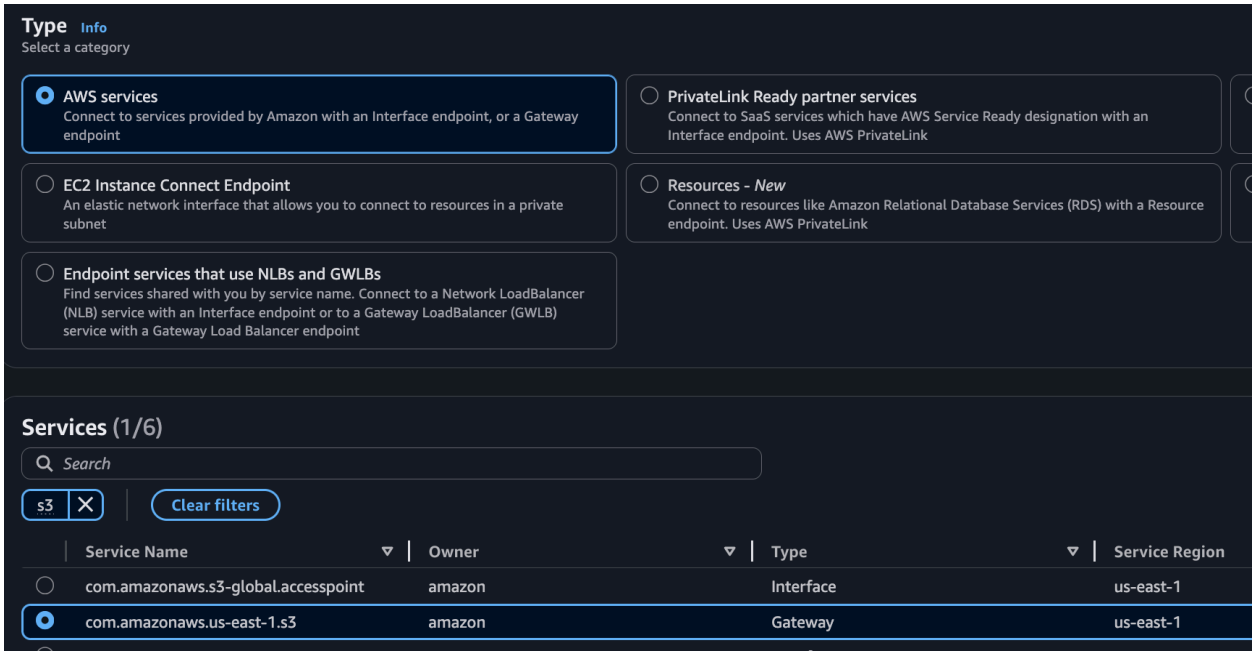
```
aws ec2 describe-instances \
  --instance-ids <my instance ID>\
  --query "Reservations[*].Instances[*].VpcId" \
  --region us-east-1
```

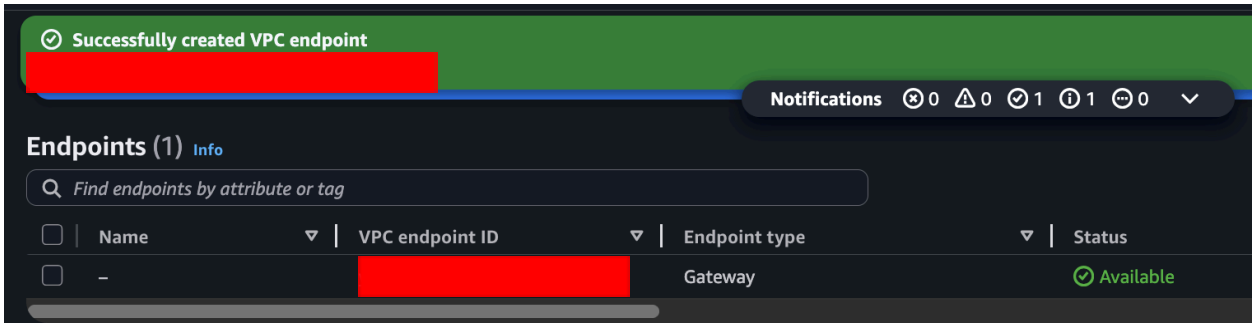


Alternatively, to determine VPC from AWS Console, navigate to the EC2 instance details page:



Creating VPC Endpoint (with com.amazonaws.us-east-1.s3 gateway service with existing vpc)

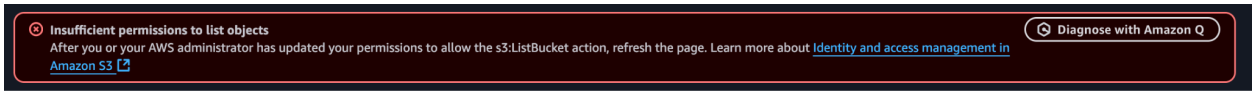




After setting up a VPC Endpoint, I decided to edit the bucket policy in a way such that any traffic NOT coming from my VPC was denied, rather than add it as an additional condition to the policy statement permitting access. This will prevent any sources outside of the VPC from accessing the S3 (even if they have the proper IAM role).

```
{
  "Sid": "DenyOutsideMyVPC",
  "Effect": "Deny",
  "Principal": "*",
  "Action": "s3:*",
  "Resource": [
    "arn:aws:s3:::[redacted]",
    "arn:aws:s3:::[redacted]/*"
  ],
  "Condition": {
    "StringNotEquals": {
      "aws:SourceVpc": "vpc-[redacted]"
    }
  }
},
```

An expected side effect of this change is I was no longer able to view the bucket contents from the AWS console:



I also experimented with temporarily changing the condition's SourceVpc value to a different VPC ID to validate that the policy was indeed being evaluated:

```
An error occurred (AccessDenied) when calling the ListObjectsV2 operation: User: arn:aws:sts::476
[redacted]:assumed-role/EC2-S3-Access-Role-Brandon/i[redacted] is not authorized to perfor
m: s3:ListBucket on resource: "arn:aws:s3::[redacted] with an explicit deny in a resource-bas
ed policy
```

Host a Static Site

TODO: Was not able to get to this, but see below for brief outline and description of steps that I hope to tackle in the future.

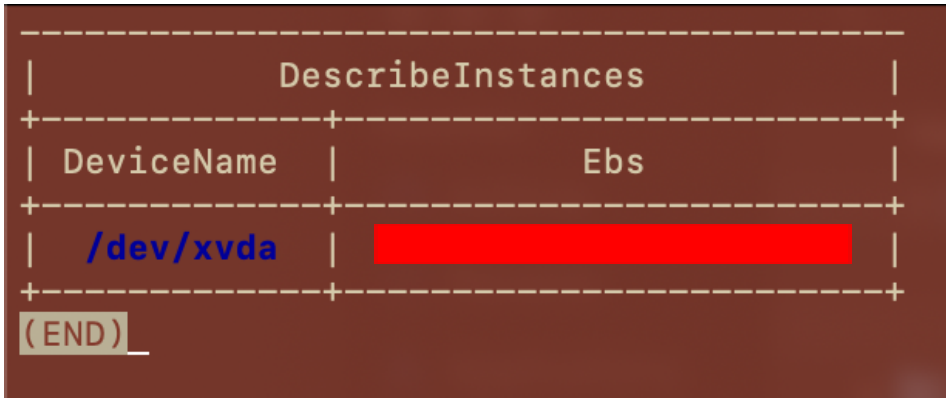
Steps

Create S3 bucket (static file storage)
Enable static website hosting (serve via HTTP)
Add public read policy (allow browsers to fetch files)
Route 53 CNAME or Alias (connect DNS name to S3)
CloudFront + ACM (Enable HTTPS with SSL)

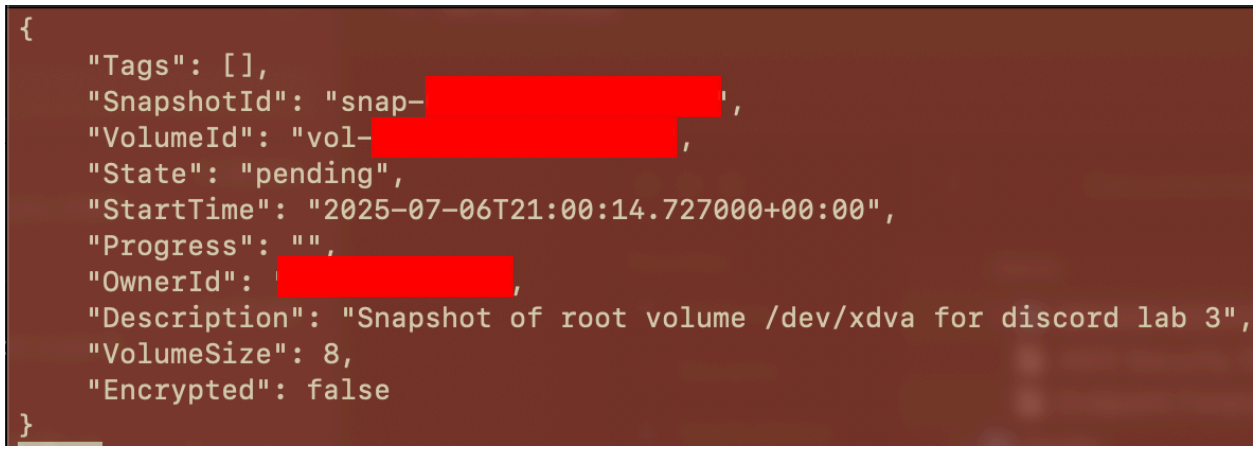
c. Further EC2 Exploration on Free Tier

1. **Snapshots & AMIs**
    - Create an EBS snapshot of `/dev/xvda`.
        - /dev/xvda is the device name assigned to the root EBS (Elastic Block Storage, the storage device) volume of the EC2 instance
            1. Can think of it as main hard drive for EC2 (like C:\ for Windows)
            2. Where OS is installed, typically maps to / . usually the first block device attached when launching EC2 instance
        - Then create snapshot from AWS console or CLI
            1. From CLI:
                a. Get Volume ID (describe-instances with query containing Ebs.VolumeId)

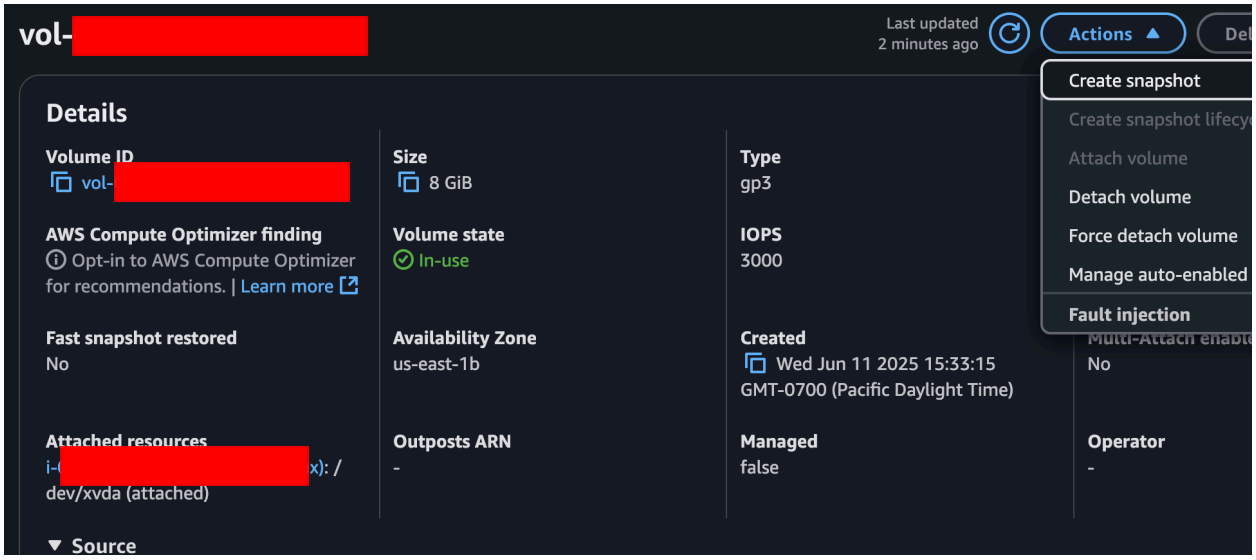
```
(base) ➜  ~ aws ec2 describe-instances --instance-id i-[redacted] --query "Reservations[*]
.Instances[*].BlockDeviceMappings[*].{DeviceName:DeviceName,Ebs:Ebs.VolumeId}" --output table
```



b. Create snapshot (using create-snapshot of volume ID found above)



2. From AWS Console:



- Register or create an AMI from that snapshot.
  - Some sources online recommend using create-image command (rather than create-snapshot -> register-image, as snapshots alone don't contain **launch config**, **boot info**, or **kernel info**). But see below for my running of register-image:

```
(base) ➜  ~ aws ec2 create-snapshot --volume-id vol-[redacted] --description "Snapshot of
root volume /dev/xdva for discord lab 3"
```

Register-image (using snapshot ID of snapshot generated above)

```
(base) ➜  ~ aws ec2 register-image \
  --name "Lab3-AMIFromSnapshot" \
  --architecture x86_64 \
  --root-device-name /dev/xvda \
  --block-device-mappings "[{\"DeviceName\": \"/dev/xvda\",\"Ebs\":{\"SnapshotId\":\"[redacted]
  --virtualization-type hvm
```

```
{
    "ImageId": "ami-[redacted]
}
```

To query for AMI ID (use describe-images to output a table containing image ID):

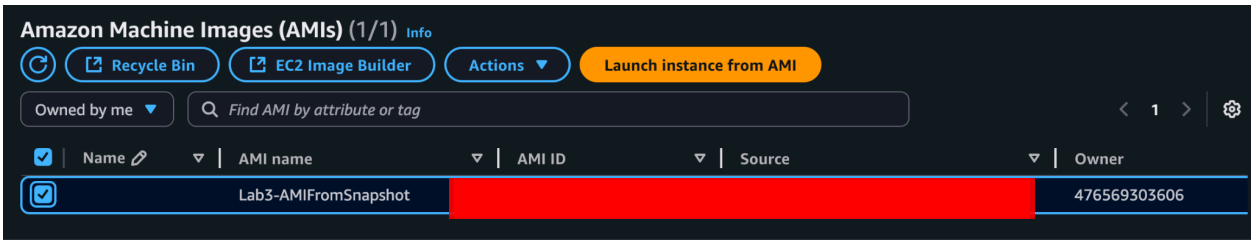
```
(base) ➜  ~ aws ec2 describe-images \
  --owners self \
  --query "Images[*].{ID:ImageId,Name:Name}" \
  --output table
```

- Lab question: Understand how you can "version" a server with snapshots. Why is this useful?
  - It's useful to 'version' a server using snapshots similar to how it's useful to version your codebase using a code repository. Before carrying out a change, you can take a snapshot of your server for contingency/rollback purposes if something goes astray.

- Launch a new instance from your AMI
  - From CLI:
    1. Utilize run-instances command (passing in the AMI ID into the image-id parameter, security-group-ids, subnet-id, etc)

```
(base) ➜  ~ aws ec2 run-instances \
  --image-id [redacted]
  --count 1 \
  --instance-type t3.micro \
  --key-name my-key-pair \
  --security-group-ids sg-xxxxxxxxxxxx \
  --subnet-id subnet-xxxxxxxxxxxx \
  --tag-specifications 'ResourceType=instance,Tags=[{Key=Name,Value=My
```

  - From AWS Console



2. **Linux & Security Tooling**

   TODO: Did not get to this

   - Use `ss -tulpn`, `lsof`, and `auditctl` to inspect services and audit.
   - Install and run:
     - `nmap localhost`
     - `tcpdump -c 20 -ni eth0`
     - `lynis audit system`
     - `fail2ban-client status`
     - OSSEC/Wazuh or ClamAV.

3. **Scripting & Automation**

   TODO: Did not get to this

   - Bash: report world-writable files (research any commands you don't know!):

     ```
     find / -perm -002 -type f > ww-files.txt
     ```

   - Python with `boto3`: list snapshots, start/stop instances.

8. Reflection

Given my general unfamiliarity with AWS, I had to repeatedly backtrack and review the steps I took earlier in the lab to find incorrect configurations. For example, when testing out policy changes using simulate-principal-policy (to test ability to get-object from my s3), I ran into an implicit deny message. I then figured out I incorrectly specified "arn:aws:s3:::<my bucket name>" as the resource for the getobject action, instead of "arn:aws:s3:::<my bucket name>/* ". After making the change, I was able to successfully run getObject using simulate-principal-policy.

The lab was definitely illuminating in terms of the number of potential security issues that can arise when provisioning/maintaining a larger environment, as security/policy creation took a lot of thought even when working with only 2 components. I faced a few policy design decisions that took me a while to weigh the pros and cons of (e.g. whether to add the IP or VPC condition as part of an 'Allow' action if the source value matched or as part of a 'Deny' if the source value did NOT match). Or as another example, what to define the Principal value as for a statement (either as * or as the role assigned to the EC2 to balance security vs availability.

As part of the lab, a key realization I made and (hope to drill into my brain haha) is that each statement in a bucket policy is evaluated independently, and that permissions are cumulative (unless there's an explicit Deny).