Lab 3

Prep

- ☒ Gitea set up
- ☒ MFA set up
- ☒ Add git ignore
- ☒ Secrets/Token Management
  - ☒ Consider secret-scanning
    - ☒ Added git-leaks on pre-commit hook
- ☒ Create & Connect to a Git repository
  - ☒ https://code.wizards.cafe
- ☒ Modify and make a second commit

Figure 1: image of terminal

- ☒ Test to see if gitea actions works
- ☒ Have an existing s3 bucket

Resources

- ☒ Capital One Data Breach
- ☒ Grant IAM User Access to Only One S3 Bucket
- ☐ IAM Bucket Policies
- ☐ Dumping S3 Buckets!

Lab

Tasks

- ☒ **4.1. Create a Custom IAM Policy**
- ☒ **4.2 Create an IAM Role for EC2**
- ☒ **4.3. Attach the Role to your EC2 Instance**
- ☒ **4.4 Verify is3 access from the EC2 Instance**
  - HTTPS outbound was not set up
    - I did not check outbound rules (even when the lab explicitly called this out) because it mentioned lab 2, so my assumption was that it had already been set up (it was not). So I had to go back and fix the missing outbound connection.

Stretch Goals

- ☒ **Deep Dive into Bucket Policies**
  - ☒ **1. Create a bucket policy** that blocks all public access but allows your IAM role
    - Implmented using: guide
- ☒ **2. Experiment** with requiring MFA or VPC conditions.
  - ☒ MFA conditions

Last activity
-

Maximum session duration
1 hour

Permissions | Trust relationships | Tags | Last Accessed | Revoke sessions

Trusted entities

Entities that can assume this role under specified conditions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "ec2.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Figure 2: trust relationships

Permissions policies (1) Info

You can attach up to 10 managed policies.

Simulate

Search

Filter by Type: All types

Policy name	Type	Att
EC2_S3_ListAndGet_daphodell	Customer managed	1

Figure 3: permissions

```
^C
[ec2-user(                    ~]$ aws s3 ls s3://witch-lab-3
2025-06-1                     1694 green.png
[ec2-user(                    ~]$ aws s3 ls s3://witch-lab-3
2025-06-1                     1694 green.png
[ec2-user(                    ~]$
```

Figure 4: screenshot of listing s3 contents

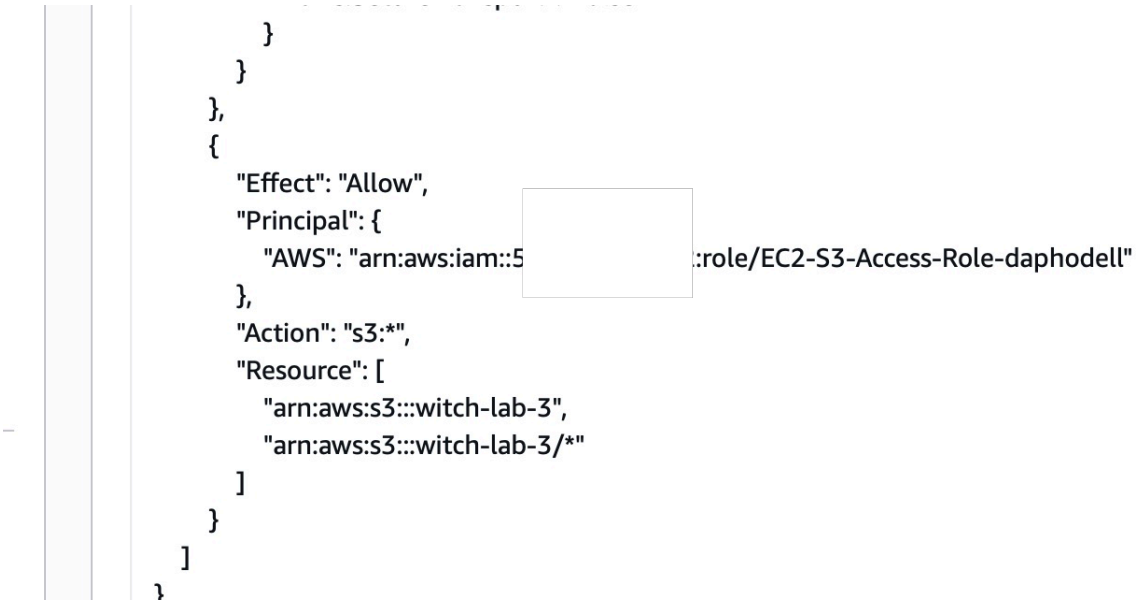

Figure 5: restrict to role

- MFA did not work out of the box after setting it in the s3 bucket policy. The ways I found you can configure MFA:
- stackoverflow
- official guide
- ☒ via cli roles - I set up a new set of role-trust relationships.
  - Update s3 Role:
    - Update action: sts:assumerole
    - Update principle (Since you cannot not target groups, I tried setting it first for my user, and then to the root account for scalability)
    - Add condition (MFA bool must be true)
- Commands referenced: I set up a script that looks like this

```
MFA_TOKEN=$1

if [ -z "$1" ]; then
  echo "Error: Run with MFA token!"
  exit 1
fi

if [ -z $BW_AWS_ACCOUNT_SECRET_ID ]; then
  echo "env var BW_AWS_ACCOUNT_SECRET_ID must be set!"
  exit 1
fi

AWS_SECRETS=$(bw get item $BW_AWS_ACCOUNT_SECRET_ID)

export AWS_ACCESS_KEY_ID=$(echo "$AWS_SECRETS" | jq -r '.fields[0].value')
export AWS_SECRET_ACCESS_KEY=$(echo "$AWS_SECRETS" | jq '.fields[1].value' | tr -d '"')

SESSION_OUTPUT=$(aws sts assume-role --role-arn $S3_ROLE --role-session-name $SESSION_TYPE --serial-number $MF
#echo $SESSION_OUTPUT
export AWS_SESSION_TOKEN=$(echo "$SESSION_OUTPUT" | jq '.Credentials.SessionToken' | tr -d '"')
export AWS_ACCESS_KEY_ID=$(echo "$SESSION_OUTPUT" | jq '.Credentials.AccessKeyId' | tr -d '"')
export AWS_SECRET_ACCESS_KEY=$(echo "$SESSION_OUTPUT" | jq '.Credentials.SecretAccessKey' | tr -d '"')
#echo $AWS_SESSION_TOKEN
#echo $AWS_ACCESS_KEY_ID
#echo $AWS_SECRET_ACCESS_KEY
aws s3 ls s3://witch-lab-3
```

- configuration via ~/.aws/credentials
- 1Password CLI with AWS Plugin
  - I use bitwarden, which also has an AWS Plugin
  - I've seen a lot more recommendations (TBH it's more like 2 vs 0) for 1password for password credential setup. Wonder why?

☒ **3. Host a static site**

- ☒ Enable a static website hosting (`index.html`)
- ☒ Configure route 53 alias or CNAME for `resume.<yourdomain>` to the bucket endpoint.
- ☒ Deploy CloudFront with ACM certificate for HTTPS
  - see: resume

☒ **b. Pre-signed URLs** (see: presigned url screenshot) `aws s3 presign s3://<YOUR_BUCKET_NAME>/resume.pdf --expires-in 3600`

- Cloudflare Edge Certificate -> Cloudfront -> S3 Bucket
- In this step, I disabled "static website hosting" on the s3 bucket
- This was actually maddening to set up. For reasons I can't understand even after Google Searching and ChatGPTing, my s3 bucket is under us-east-2 and Cloudfront kept redirecting me to the us-east-1 for some reason. I don't like switching up regions under AWS because this way it's easy to forget what region you created a specific service in because they're hidden depending on what region is active at the moment.

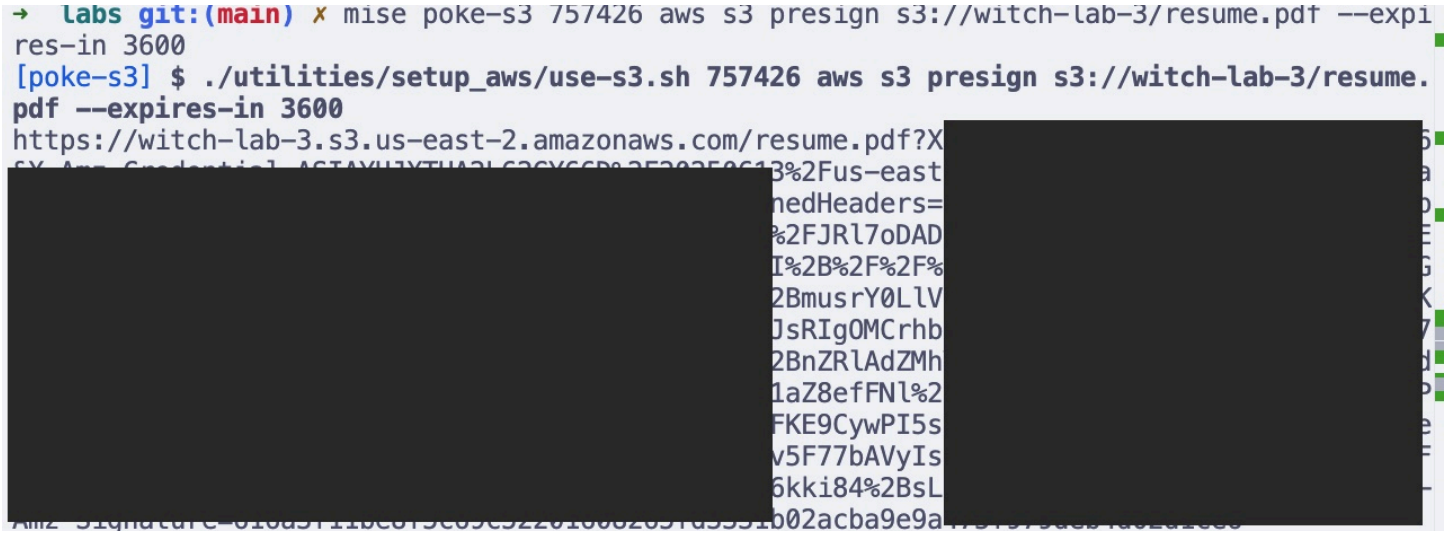

Figure 6: presigned url

☒ Import resources into terraform Ran some commands:

```
terraform import aws_iam_policy.assume_role_s3_policy $TF_VAR_ASSUME_ROLE_POLICY
...
terraform plan
terraform apply
```

Further Exploration

☒ Snapshots & AMIs

- ☒ Create an EBS snapshot of `/dev/xvda`
- ☒ Register/create an AMI from that snapshot
  - I think with terraform we can combine the two steps and use `aws_ami_from_instance` to get the end result (create an AMI from snapshot) for free. Otherwise I think you would want to do aws_ebs_snapshot and then aws_ami

```
resource "aws_ami_from_instance" "ami_snapshot" {
  name = "ami-snapshot-${formatdate("YYYY-MM-DD", timestamp())}"
  source_instance_id = aws_instance.my_first_linux.id
  snapshot_without_reboot = true

  tags = {
    Name = "labs"
```

```
  }
}
```

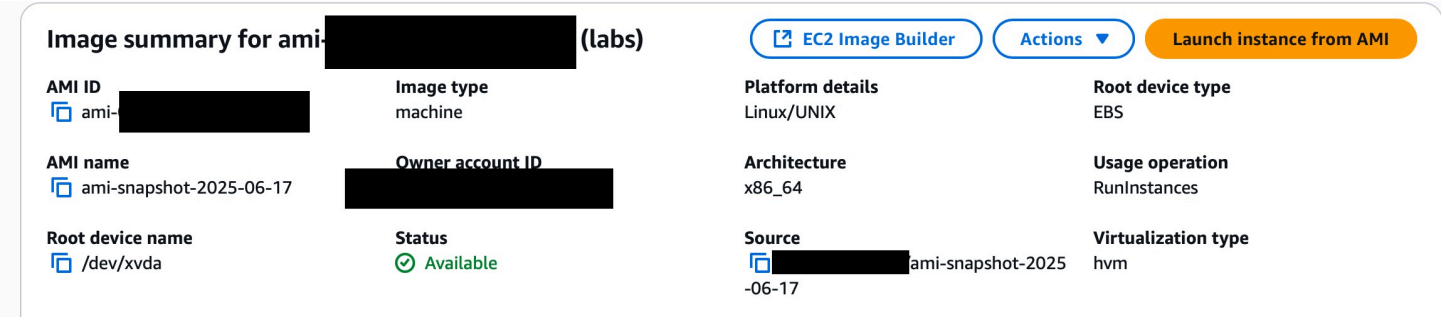

Figure 7: ami creation

☒ How do you "version" a server with snapshots? Why is this useful?

- **Cattle, not pets**

  This is useful for following the concept for treating your servers as "cattle, not pets". Being able to keep versioned snapshots of your machines means there's nothing special about your currently running server. If it goes down (or you need to shoot it down), you can restore it on another machine from an older snapshot.

  Or, as another example, suppose you were tasked with installing a whole suite of tools on an ec2 instance (ex: fail2ban, ClamAV, etc.). Then your boss tells you the company needs the same setup for another 49 instances.

  Having 1 AMI that contains all of those tools can save you from re-running the same commands 50x times.

☒ Launch a new instance from your AMI

Terraform is great. Terraform is life. This was really simple to do.

```
# Launch new instance from AMI
resource "aws_instance" "my_second_linux" {
    instance_type = "t2.micro"
    ami = aws_ami_from_instance.ami_snapshot.arn
    security_groups = ["ssh-access-witch"]

    tags = {
        Name = "labs"
    }
}
```

☒ Convert to terraform

- Terraform files can be found here

Reflection

- What I built
  - A secured s3 bucket for secure content that can only be accessed via multi-factor authentication Good for storing particularly sensitive information.
  - A minimal HTML website served from an S3 bucket
- Challenges
  - The stretch goal for setting up s3 + mfa was a bit of a pain:
    - Groups cannot be used as the principal in a trust relationship so to get things working I added the trust relationship to my user's ARN instead. I prodded ChatGPT on a more practical way to do this (this wouldn't scale with 100s of users, onboarding/offboarding etc.) and had to go back and fix how the policies worked.
  - Issues between setting up Cloudflare -> CloudFront -> s3 bucket
    - I think adding an extra service (Cloudflare, where I host my domain) added a little bit of complexity, though my main issue was figuring out how to set up the ACM cert -> CloudFront distribution -> S3. Most of the instructions I was able to parse through with ChatGPT – I have to say I had a much better reading through

those instructions than with the official AWS docs, which led me through nested links (understandably, because there seem to be multiple ways of doing everything).

- Security concerns

  Scale and security at scale

  I started out this lab doing "click-ops", and I noticed while testing connections that there was just a lot of trial and error in setting up permissions.

  My process seemed to be: OK, this seems pretty straightforward, let's just add the policy.

  But after adding the policy it looked like there was a cascade of errors where I forgot to add additional permissions or trust relationships that weren't immediately obvious until I actually went through the error logs one by one.

  Once everything got set up via click-ops and imported to terraform though, repeating the same steps via Terraform was *very easy*.

  I think putting everything down into code really helps to self-document the steps it takes to get a fully functioning setup.

Terms

Identity Access Management

```
graph LR
  IAMPolicy -- attaches to --> IAMIdentity
  ExplainIAMIdentity[users, groups of users, roles, AWS resources]:::aside
  ExplainIAMIdentity -.-> IAMIdentity
classDef aside stroke-dasharray: 5 5, stroke-width:2px;
```

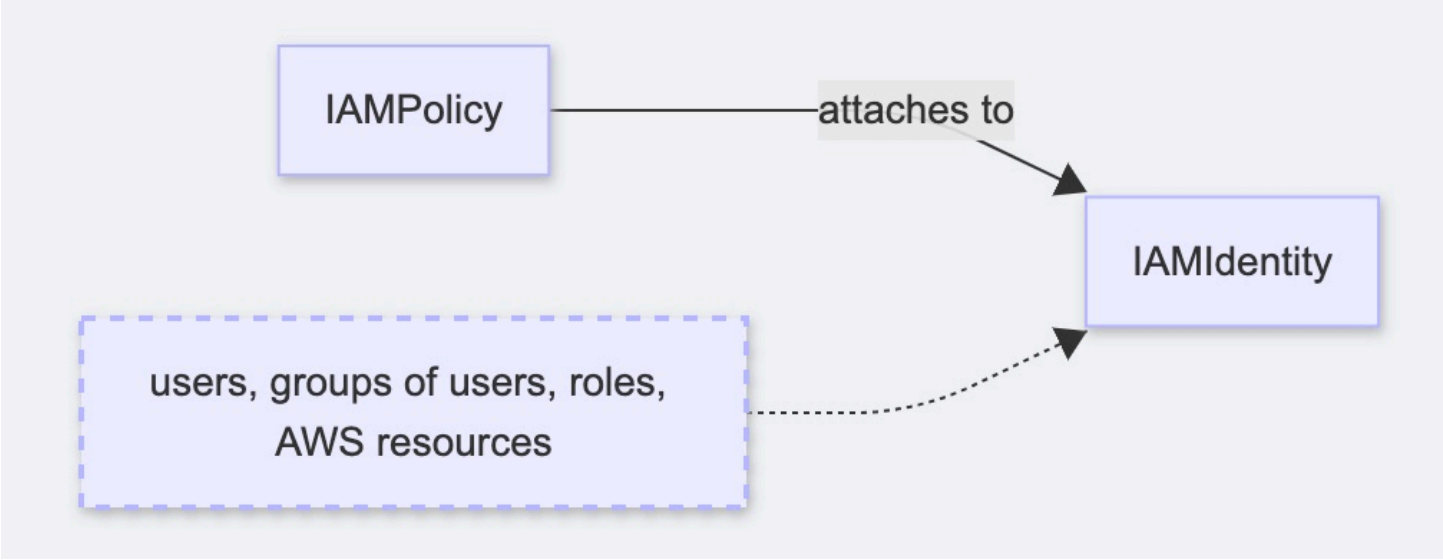

Figure 8: Identity Access Management