nextwork.org

Cloud Security with AWS IAM

Blackout

Policy editor

```
1  {
2    "Version": "2012-10-17",
3    "Statement": [
4      {
5        "Effect": "Allow",
6        "Action": "ec2:*",
7        "Resource": "*",
8        "Condition": {
9          "StringEquals": {
10           "ec2:ResourceTag/Env": "development"
11         }
12       }
13     },
14     {
15       "Effect": "Allow",
16       "Action": "ec2:Describe*",
17       "Resource": "*"
18     },
19     {
20       "Effect": "Deny",
21       "Action": [
22         "ec2:DeleteTags",
23         "ec2:CreateTags"
24       ],
25       "Resource": "*"
26     }
27   ]
28 }
```

Introducing Today's Project!

In this project, I will demonstrate how to use AWS IAM to manage authentication and authorisation. By launching an EC2 instance and applying IAM policies and groups, I aim to strengthen my understanding of cloud security best practices.

Tools and concepts

Services I used were AWS IAM for users and groups and EC2 for instances. Key concepts I learnt include tagging EC2 instances, creating JSON policies, managing IAM users and groups, and controlling access based on policies and tags.

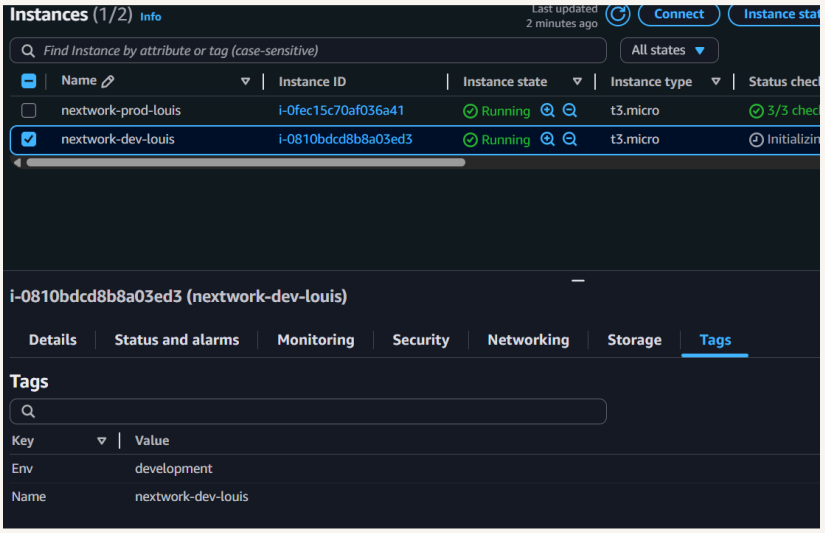
Project reflection

This project took me approximately one hour. The most challenging part was writing the JSON policy to enforce precise permissions. It was most rewarding to ensure security, allowing access to development resources while protecting production.

Tags

Tags in AWS are key-value pairs that help you organize and manage your resources by assigning custom metadata, such as project name or environment. They make it easier to search, filter, and apply policies across resources.

The tag I've used on my EC2 instances is called Name and Env. The values I've assigned for my instances are nextwork-dev-louis with dev and nextwork-prod-louis with prod.



IAM Policies

IAM policies define permissions in AWS by specifying which actions are allowed or denied on specific resources, helping control secure access to cloud services.

The policy I set up

For this project, I've set up a policy using JSON because it allows precise control over permissions and makes it easier to customize access to specific resources.

I've created a policy that allows full EC2 access for resources tagged Env=development, read-only access to all EC2 resources, and explicitly denies creating or deleting tags to protect environment integrity.

When creating a JSON policy, you have to define its Effect, Action and Resource.

Effect specifies whether the policy allows or denies an action. Action defines which API operations are permitted or blocked. Resource identifies the specific AWS resources the policy applies to, like an EC2 instance or S3 bucket.

My JSON Policy

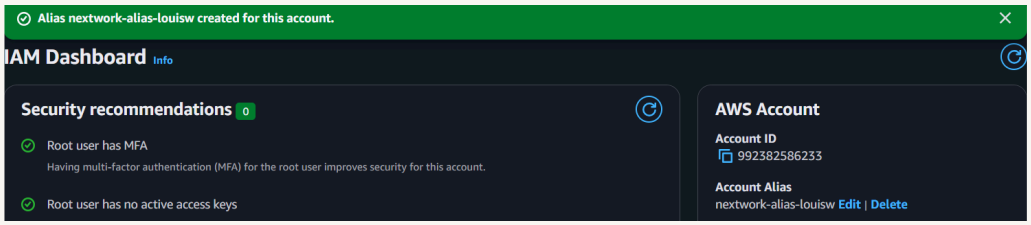
Policy editor

```
1  {
2    "Version": "2012-10-17",
3    "Statement": [
4      {
5        "Effect": "Allow",
6        "Action": "ec2:*",
7        "Resource": "*",
8        "Condition": {
9          "StringEquals": {
10           "ec2:ResourceTag/Env": "development"
11         }
12       }
13     },
14     {
15       "Effect": "Allow",
16       "Action": "ec2:Describe*",
17       "Resource": "*"
18     },
19     {
20       "Effect": "Deny",
21       "Action": [
22         "ec2:DeleteTags",
23         "ec2:CreateTags"
24       ],
25       "Resource": "*"
26     }
27   ]
28 }
```

Account Alias

An Account Alias is a custom, easy-to-remember name for your AWS account that replaces the long numeric account ID in the login URL, making it simpler for users to access the AWS Management Console.

Creating an account alias took me just a few seconds. Now, my new AWS console sign-in URL is https://louisw.signin.aws.amazon.com/console



IAM Users and User Groups

Users

IAM users are individual identities within an AWS account that represent people or applications. Each user has unique credentials and can be assigned permissions to access AWS resources securely.

User Groups

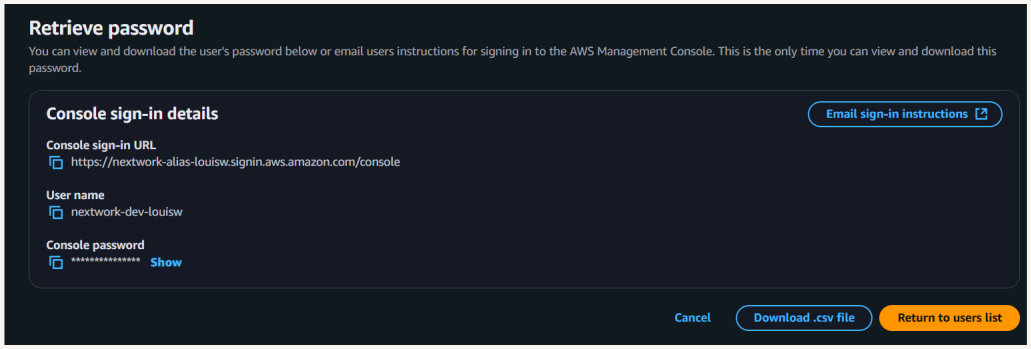
IAM user groups are collections of IAM users that let you manage permissions collectively. Instead of assigning policies to each user, you attach policies to the group, and all members inherit those permissions.

I attached the policy I created to this user group, which means all users in the group automatically inherit the policy's permissions, allowing them to access EC2 resources as defined without assigning permissions individually.

Logging in as an IAM User

The first way is to download or view the user's temporary password and give it to them directly. The second way is to email instructions with the AWS console login URL, username, and temporary password so they can log in.

Once I logged in as my new IAM user, I noticed some dashboard panels showed Access Denied. This was because my permissions are limited by the attached policy, restricting access to only the resources and actions allowed.



Testing IAM Policies

I tested my JSON IAM policy by starting and stopping the development EC2 instance, which was allowed, and attempting to stop the production instance, which was denied due to the policy restrictions.

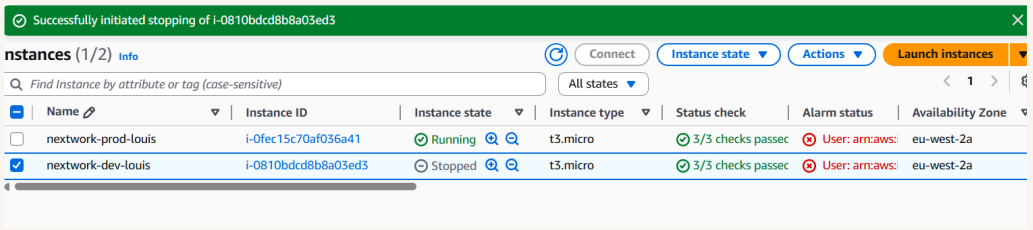
Stopping the production instance

When I tried to stop the production instance, I received an Access Denied error. This was because the policy only allows actions on development instances and explicitly restricts modifying production resources.

Testing IAM Policies

Stopping the development instance

Next, when I tried to stop the development instance, it stopped successfully. This was because the IAM policy allows all actions on EC2 resources tagged Env=development.



nextwork.org

The place to learn & showcase your skills

Check out nextwork.org for more projects