Cloud Security with AWS IAM

Project Overview

This project demonstrates the implementation of AWS Identity and Access Management (IAM) to control access to cloud resources. I successfully configured EC2 instances with tag-based access control, created custom IAM policies, established user groups, and tested security permissions to ensure proper access restrictions.

Project Duration: Approximately 40 minutes
Difficulty Level: Easy
AWS Region Used: eu-west-3 (Paris)

Table of Contents

What I Built

A complete IAM security infrastructure demonstrating:

- Two EC2 instances with environment-based tagging (production and development)
- Custom IAM policy with conditional access controls
- IAM user group for managing intern permissions
- IAM user with restricted access based on resource tags
- AWS account alias for simplified login

This project simulates a real-world scenario where an intern needs access to development resources while being restricted from production environments.

Technologies & Concepts

AWS Services Used

- **Amazon EC2 (Elastic Compute Cloud)** - Virtual servers for hosting applications
- **AWS IAM (Identity and Access Management)** - Authentication and authorization service
- **Resource Tagging** - Metadata labels for organizing and controlling access to resources

Key Concepts Learned

1. **IAM Policies** - JSON-based permission documents that define what actions are allowed or denied
2. **IAM Users** - Individual identities representing people or applications

3. **IAM User Groups** - Collections of users with shared permissions
4. **Tag-Based Access Control** - Using resource tags to conditionally grant or deny permissions
5. **Account Aliases** - Custom, memorable names for AWS account sign-in URLs
6. **Policy Effects** - Understanding "Allow" and "Deny" statements (Deny always takes precedence)
7. **Conditions** - Adding logic to policies to make permissions context-aware

---

Step-by-Step Implementation

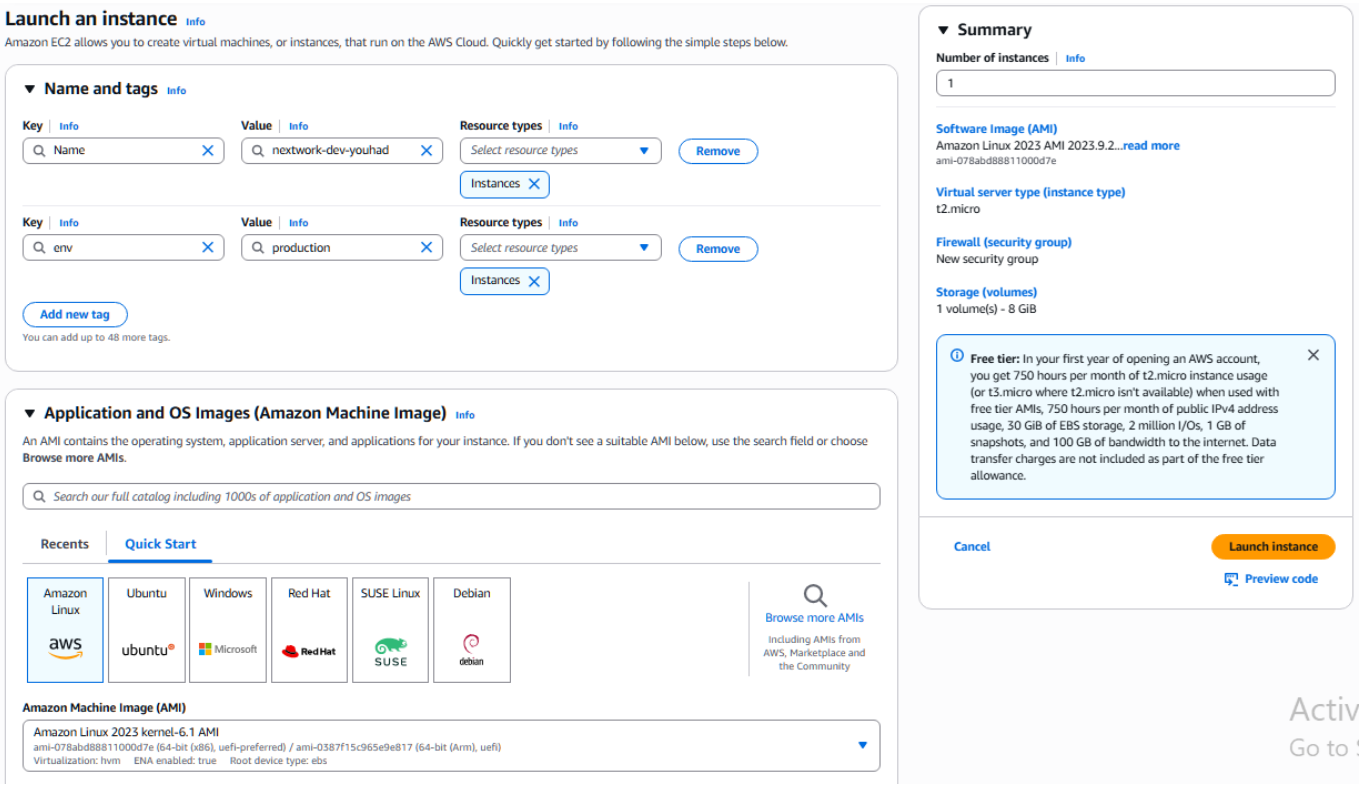
Step 1: Launch EC2 Instances

What I did:

I launched two Amazon EC2 instances to simulate production and development environments for the NextWork company.

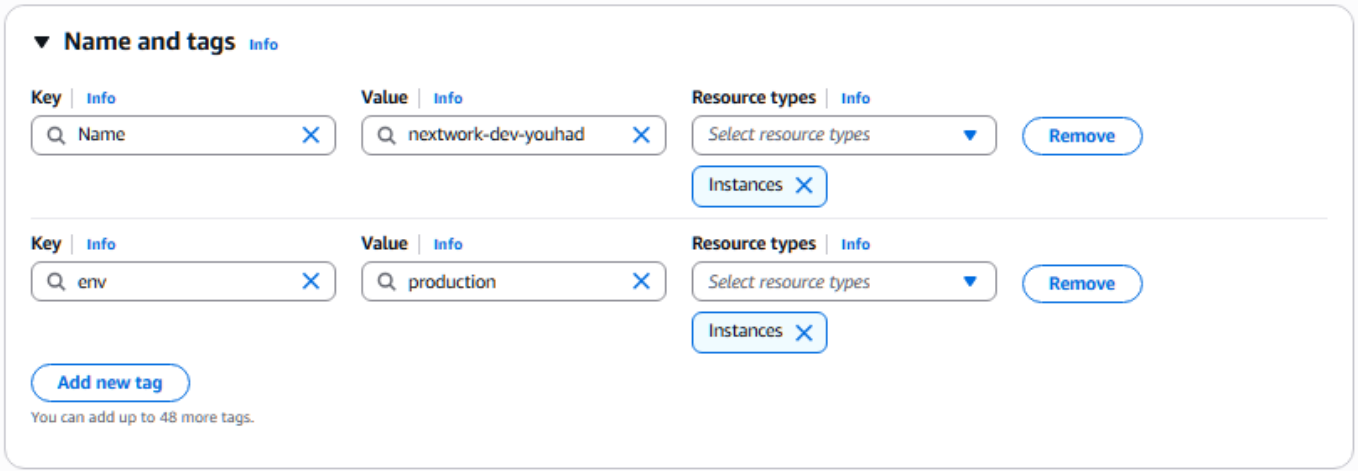
Instance Configuration:

- **Instance Type:** t2.micro (Free tier eligible)
- **AMI:** Amazon Linux 2023 (64-bit x86)
- **Key Pair:** Proceeded without a key pair (console access only)
- **Network Settings:** Default VPC with auto-assign public IP enabled
- **Security Group:** New security group with SSH access from anywhere
- **Storage:** 8 GiB gp3 volume

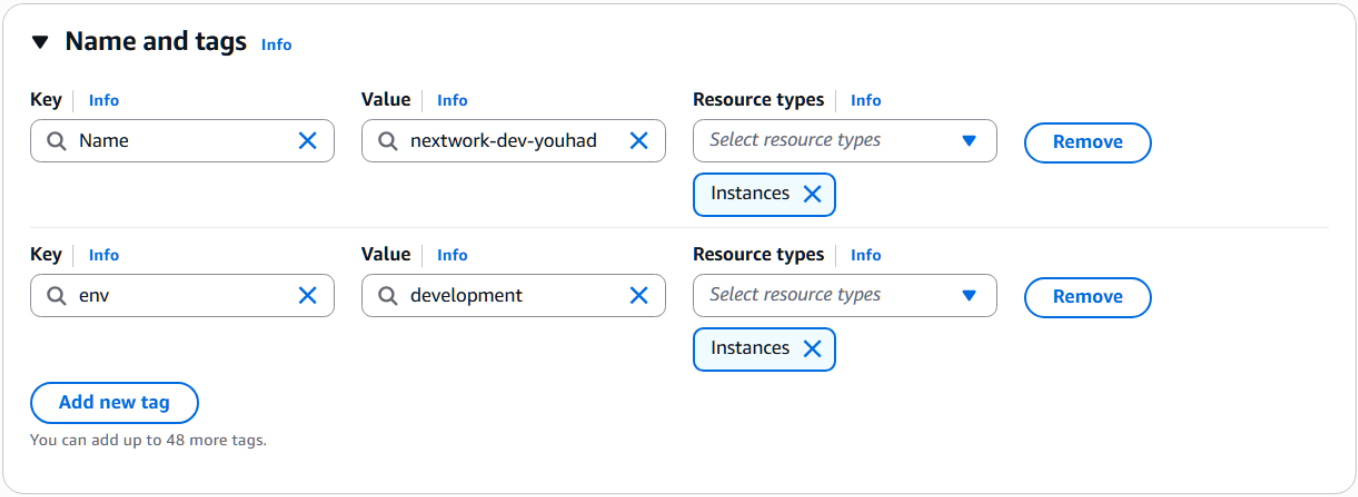


Tags Applied:

1. **Production Instance:**

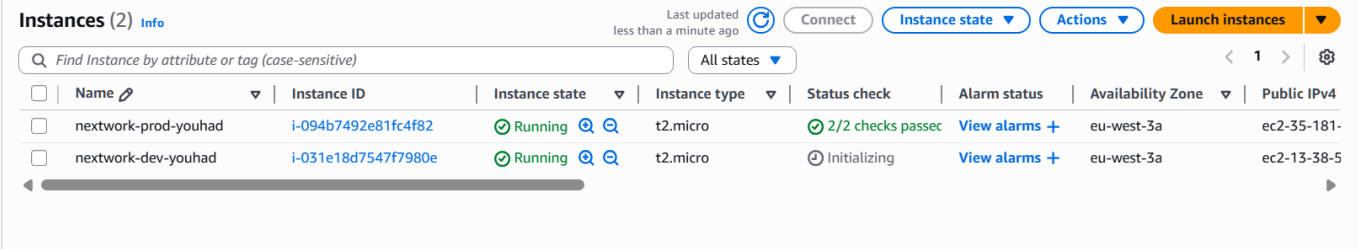


1. **Development Instance:**



Why tags matter:

Tags are key-value pairs that help organize AWS resources. More importantly, they can be used in IAM policies to create conditional access controls. In this project, the `env` tag determines whether a user can perform actions on an instance.

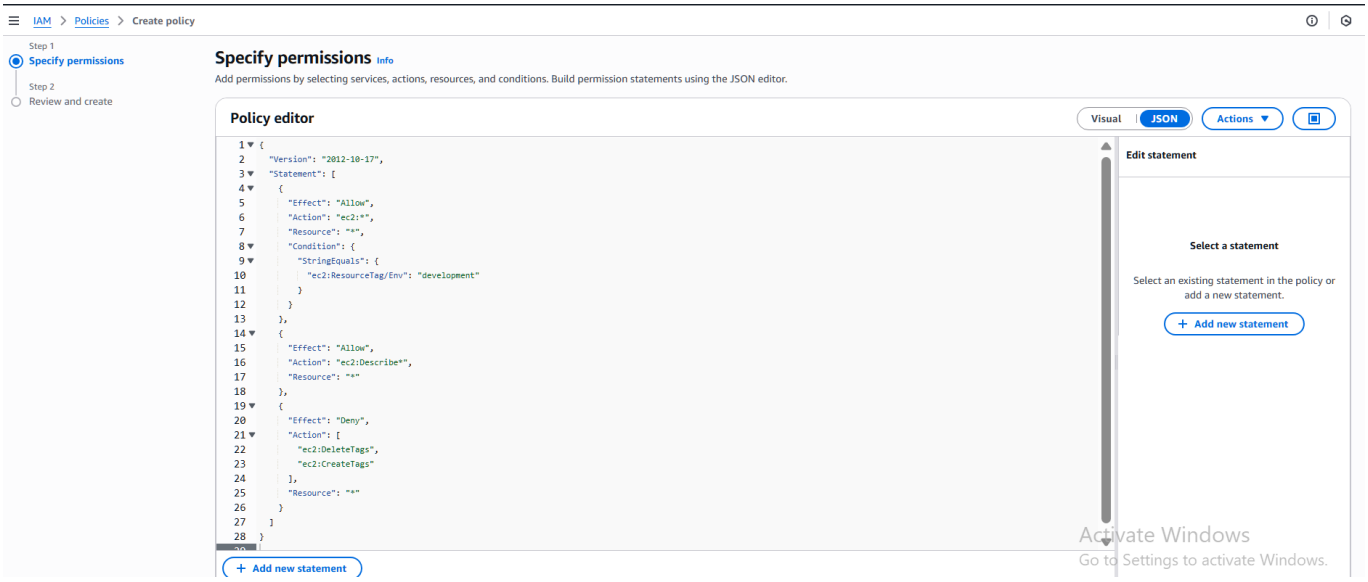


Result: Both instances successfully launched and are running in the eu-west-3 region.

Step 2: Create an IAM Policy

What I did:

I created a custom IAM policy called `NextWorkDevEnvironmentPolicy` that grants specific permissions based on resource tags.



Breaking Down the Policy:

Statement 1 - Conditional Full Access:

```
{
  "Effect": "Allow",
  "Action": "ec2:*",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "ec2:ResourceTag/Env": "development"
    }
  }
}
```

- Allows **all EC2 actions** (`ec2:*`)
- Only on resources tagged with `Env=development`
- This means users can start, stop, modify, and manage development instances

Statement 2 - Read-Only Access:

```
{
  "Effect": "Allow",
  "Action": "ec2:Describe*",
  "Resource": "*"
}
```

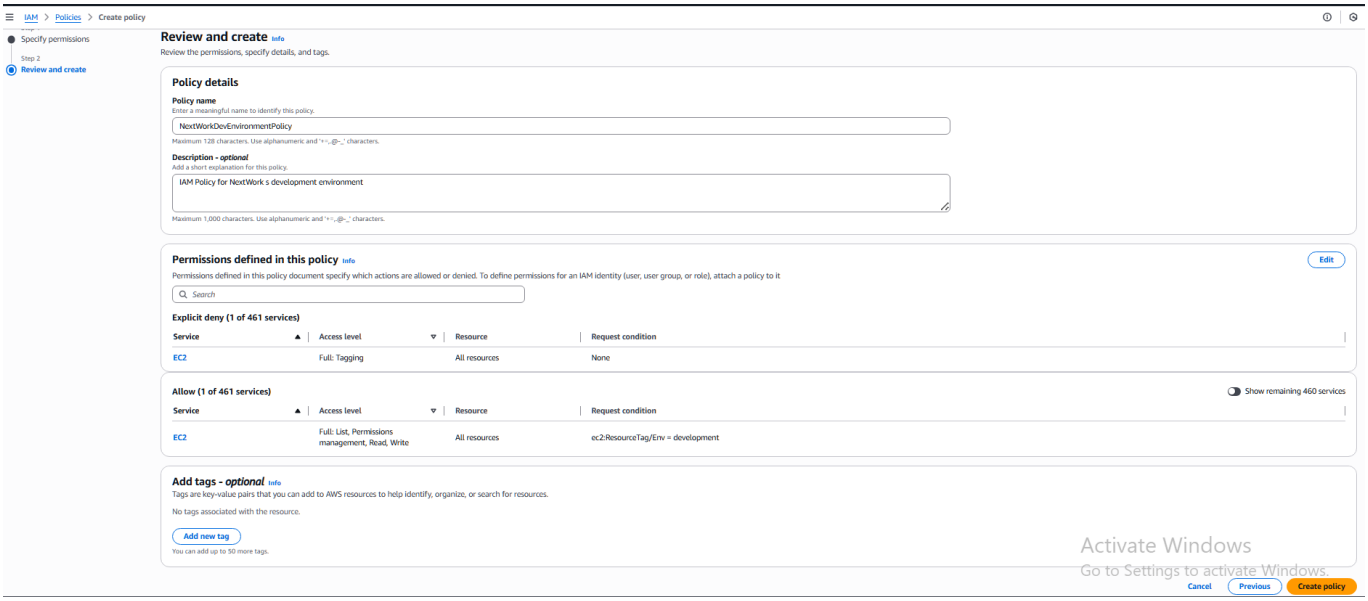
- Allows all **Describe** actions (read-only operations)
- On **all resources** (no conditions)
- Users can view all instances but can only modify development ones

Statement 3 - Explicit Deny:

```
{
  "Effect": "Deny",
  "Action": ["ec2:DeleteTags", "ec2:CreateTags"],
  "Resource": "*"
}
```

- **Denies** tag modification actions
- On **all resources**
- Prevents users from changing tags to bypass access controls
- **Remember:** Deny always overrides Allow

Policy Details:



Key Understanding:

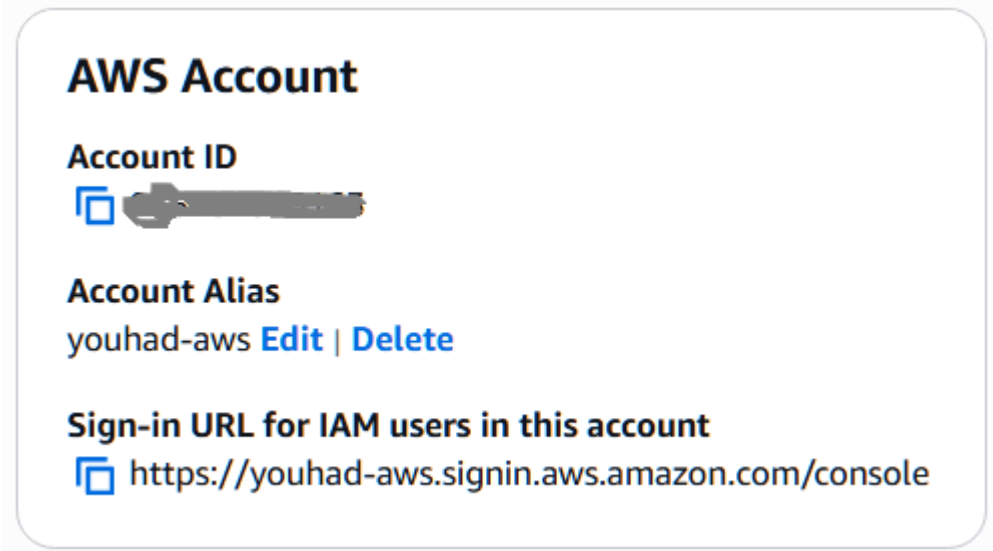
This policy demonstrates the **principle of least privilege** - users get exactly the permissions they need, nothing more. Interns can fully manage development resources but only view production resources, and they cannot manipulate tags to escalate their privileges.

Step 3: Create an AWS Account Alias

What I did:

I created a custom account alias to make the IAM sign-in URL more user-friendly.

Account Details:



- **Account ID:** `[Redacted for security]`
- **Account Alias:** `youhad-aws`
- **Sign-in URL for IAM users:** `https://youhad-aws.signin.aws.amazon.com/console`

What is an Account Alias?

An account alias is a custom, memorable name that replaces your 12-digit AWS account ID in the sign-in URL. Instead of sharing a complex URL like:

```
https://123456789012.signin.aws.amazon.com/console
```

Users can access:

```
https://youhad-aws.signin.aws.amazon.com/console
```

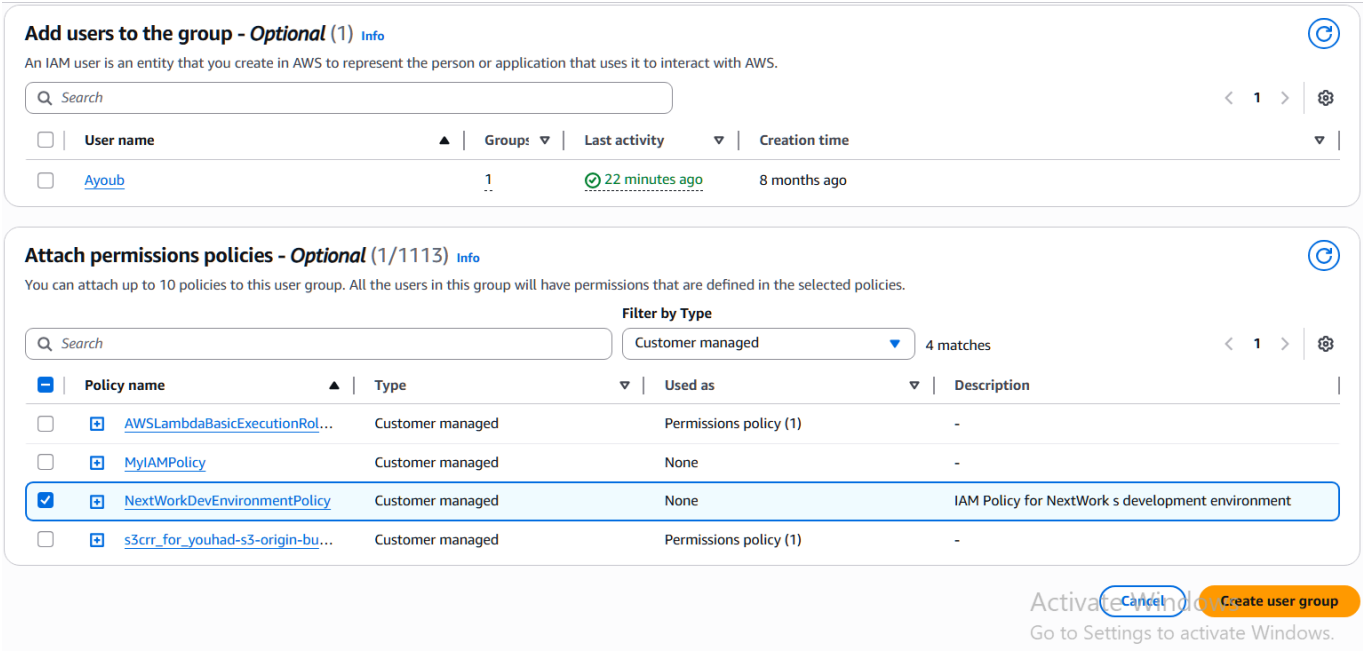
Benefits:

- Easier to remember and share
- More professional appearance
- Reduces login errors
- One alias per AWS account (must be globally unique)

Step 4: Create IAM User Group

What I did:

I created an IAM user group called `nextwork-dev-group` to manage permissions for all NextWork interns collectively.



User Group Configuration:

- **Group Name:** `nextwork-dev-group`
- **Attached Policy:** `NextWorkDevEnvironmentPolicy`
- **Users:** Initially 0 (user will be added in next step)

What are IAM User Groups?

IAM user groups are collections of IAM users that share the same set of permissions. Instead of attaching policies to individual users, you attach them to groups, making permission management much more efficient.

Why use groups?

Imagine managing 50 interns individually - if you need to update permissions, you'd have to modify 50 separate users. With groups:

1. Create one group
2. Attach policies to the group
3. Add all interns to the group
4. Any policy changes automatically apply to all members

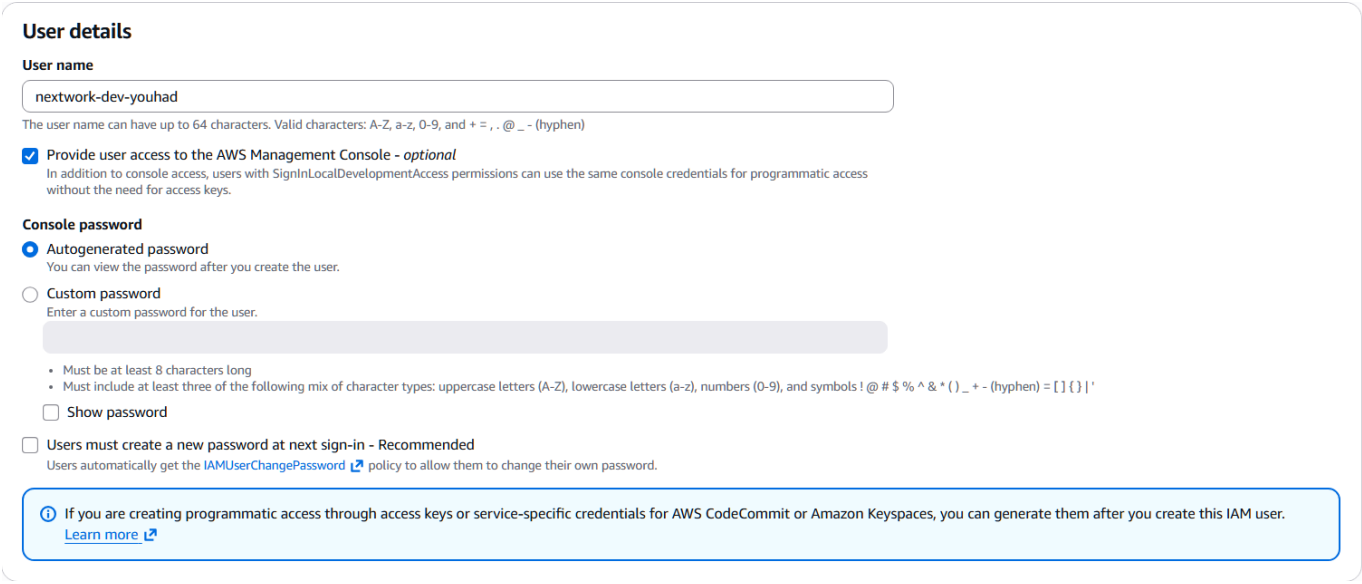
Best Practice: Always use groups for permission management, even for a single user. This makes future scaling easier and follows AWS security best practices.

Step 5: Create IAM User

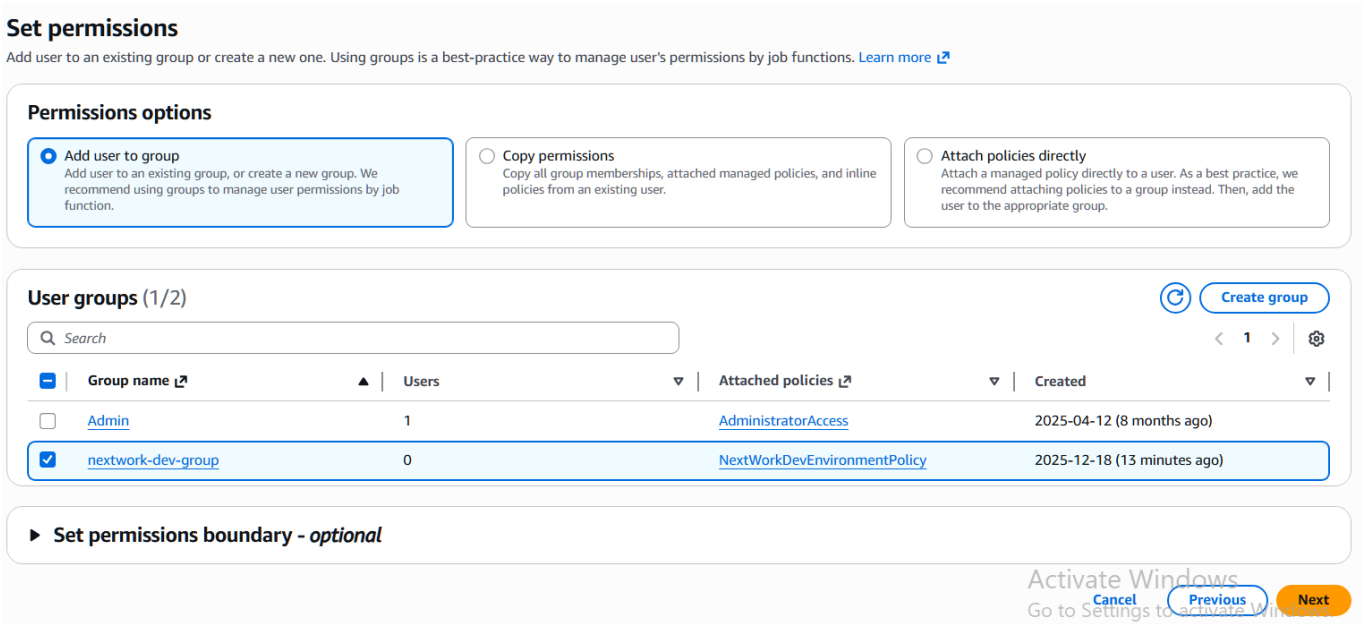
What I did:

I created an IAM user for a new NextWork intern with console access and added them to the development user group.

User Configuration:



Permissions Setup:

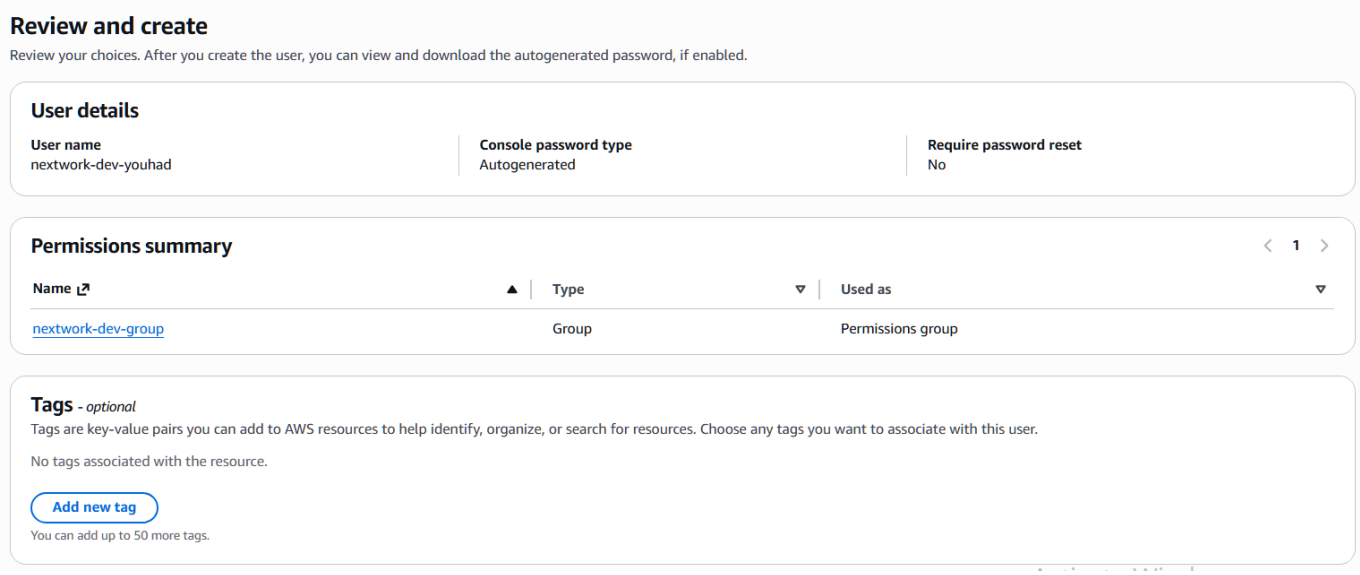


Why add to a group?

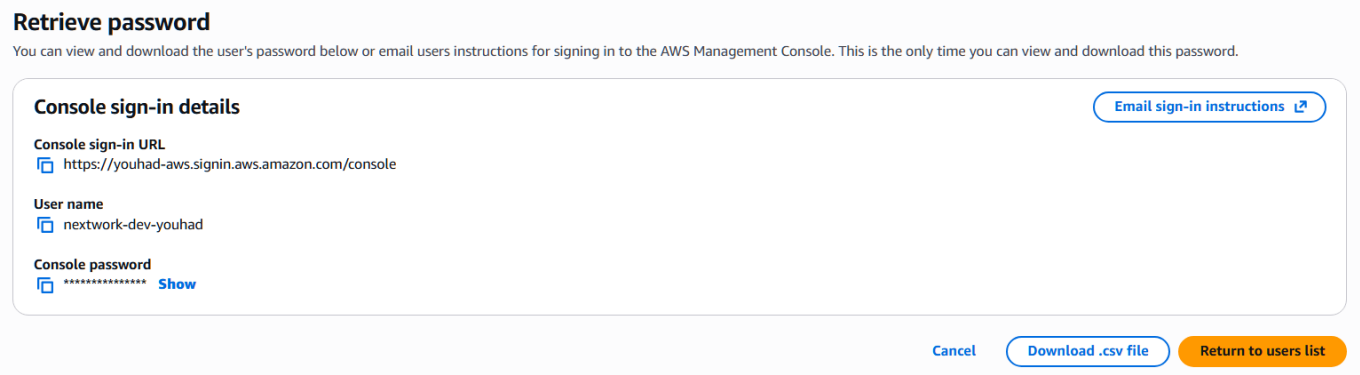
Rather than attaching the policy directly to the user, adding them to a group is a best practice because:

- Easier to manage permissions for multiple users
- Consistent permissions across team members
- Simpler to audit and modify access
- Follows the principle of role-based access control (RBAC)

Review Summary:



Console Sign-in Details:



Important Security Note:

This is the **only time** you can view or download the autogenerated password. In production environments, you should:

- Download the CSV file with credentials
- Send credentials through a secure channel (not email)
- Require password reset on first login
- Enable multi-factor authentication (MFA)

Result: The IAM user is now created and ready to test our policy restrictions!

Step 6: Testing IAM Policy - Failed Stop Production Instance

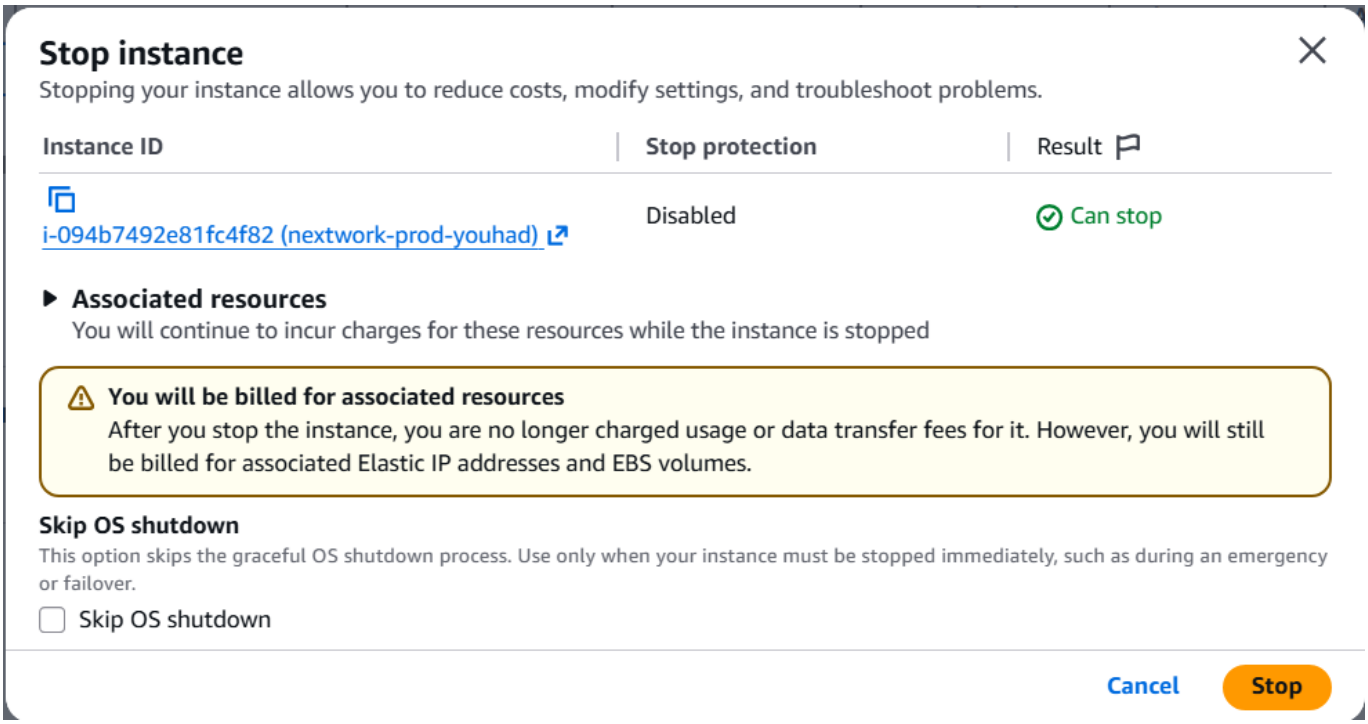
What I did:

I logged in as the IAM user `nextwork-dev-youhad` and attempted to stop the production instance to verify that the policy correctly denies access.

Test Scenario:

- **Instance:** `nextwork-prod-youhad` (tagged with `env=production`)
- **Action:** Stop instance

- **Instance ID:** `i-094b7492e81fc4f82`



Result: Access Denied!

Why did this fail?

Looking back at our IAM policy, the first statement only allows EC2 actions when:

```
"Condition": {
  "StringEquals": {
    "ec2:ResourceTag/Env": "development"
  }
}
```

Since the production instance has `Env=production`, the condition doesn't match, and the action is denied. The policy is working correctly!

This is exactly what we wanted - interns cannot modify production resources, preventing accidental or unauthorized changes to critical infrastructure.

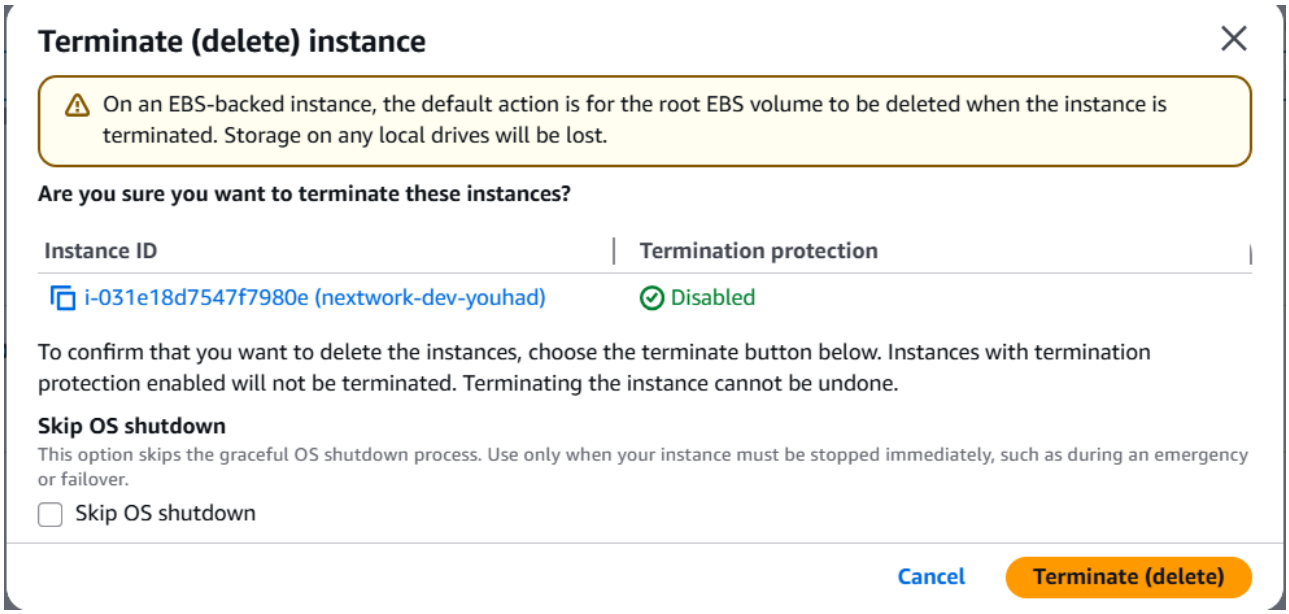
Step 7: Testing IAM Policy - Successfully Stop Development Instance

What I did:

Still logged in as the IAM user, I attempted to stop the development instance to verify the policy grants proper access.

Test Scenario:

- **Instance:** `nextwork-dev-youhad` (tagged with `env=development`)
- **Action:** Stop instance
- **Instance ID:** `i-031e18d7547f7980e`



Result: Success! ☑

The instance state changed from "Running" to "Stopping", proving that the IAM user has permission to manage development resources.

Why did this succeed?

The policy's first statement grants full EC2 permissions (`ec2:*`) when the resource tag matches `Env=development`. Since this instance has the correct tag, the user can:

- Stop the instance
- Start the instance
- Reboot the instance
- Modify instance settings
- Perform any other EC2 action

Key Takeaway:

Tag-based access control allows for flexible, scalable security policies. You can have hundreds of resources, and users automatically get the right permissions based on tags - no need to list specific instance IDs in policies.

Step 8: Cleanup - Terminate Instances

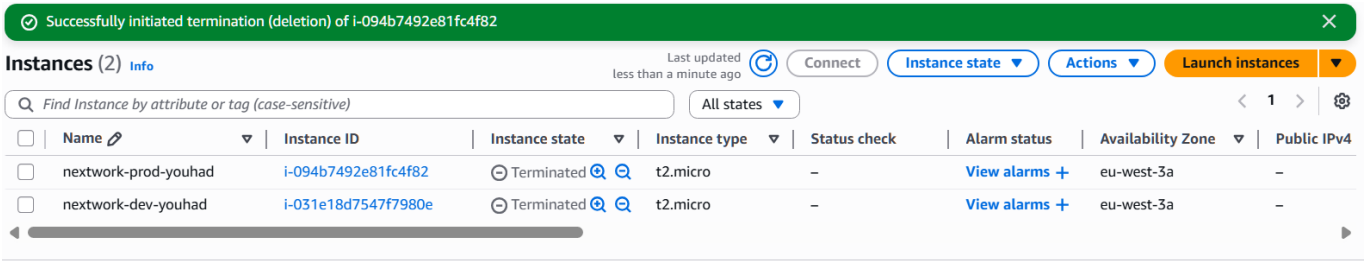
What I did:

I switched back to my admin AWS account (with full administrator access) to terminate the production and devolepement instances.

Why switch accounts?

The IAM user `nextwork-dev-youhad` doesn't have permission to terminate production instances due to our policy restrictions. This demonstrates another layer of security - only administrators with proper credentials can manage production resources.

Final Instance Status:

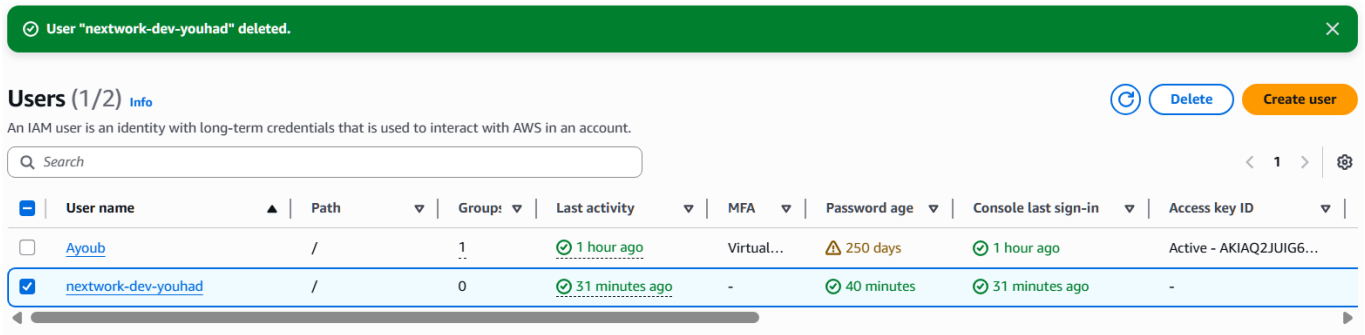


Both instances now show "Terminated" status, and AWS will automatically remove them from the console view after a short period.

Step 10: Cleanup - Delete IAM User

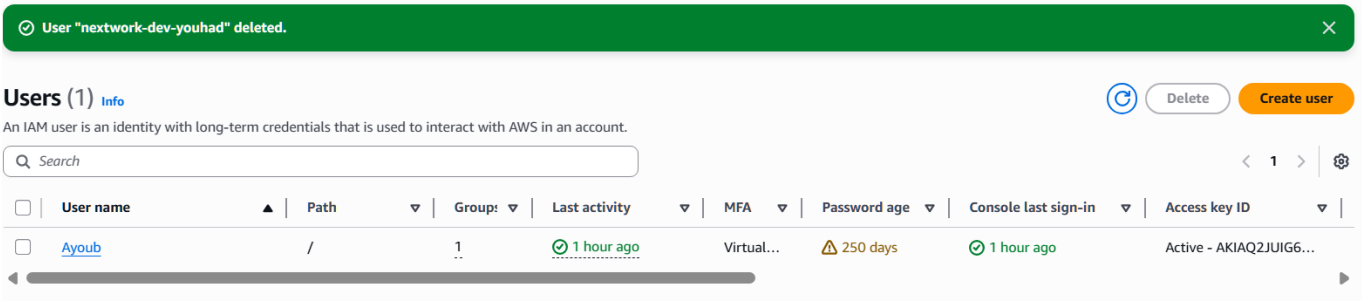
What I did:

I deleted the IAM user `nextwork-dev-youhad` since the project testing is complete.



User Deletion Confirmation:

- **User Name:** `nextwork-dev-youhad`
- **Groups:** 0 (automatically removed from groups upon deletion)
- **Last Activity:** 31 minutes ago
- **Password Age:** 40 minutes



What Happens When You Delete a User?

- All associated credentials (password, access keys) are permanently revoked
- User is automatically removed from all groups
- Any policies directly attached to the user are detached
- The user's sign-in URL becomes invalid
- Action cannot be undone (though you can create a new user with the same name)

Security Best Practice:

Delete IAM users when:

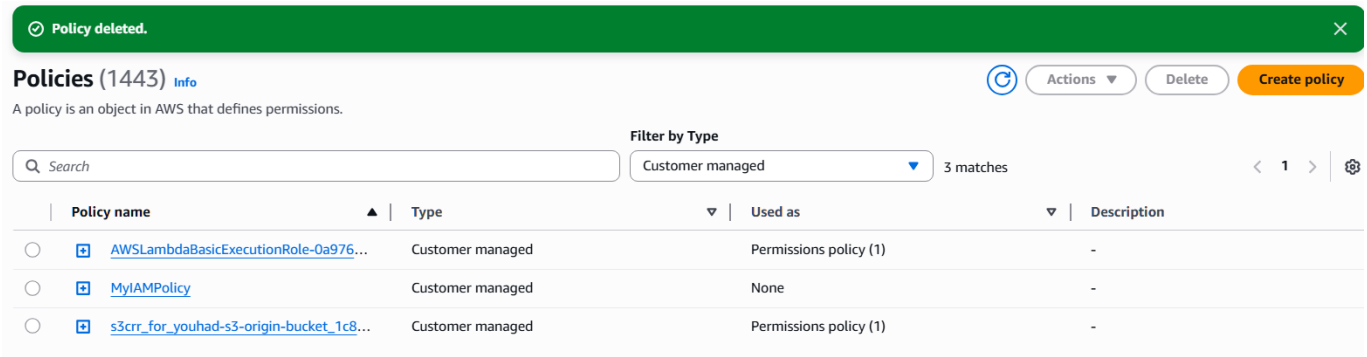
- Employees leave the company
- Test users are no longer needed
- Credentials may have been compromised
- Following the principle of least privilege and clean account hygiene

---

Step 11: Cleanup - Delete IAM Policy

What I did:

I deleted the custom IAM policy `NextWorkDevEnvironmentPolicy` to complete the cleanup process.



Remaining Policies: After deletion, only standard AWS-managed policies and other custom policies remain:

- `AWSLambdaBasicExecutionRole` (AWS managed)
- `MyIAMPolicy` (Custom managed)
- `s3crr_for_youhad-s3-origin-bucket` (Custom managed)

Important Notes About Deleting Policies:

- You can only delete customer-managed policies (not AWS-managed policies)

- Policy must not be attached to any users, groups, or roles
- Once deleted, the policy cannot be recovered
- All versions of the policy are permanently deleted

Why Clean Up Policies?

- **Security:** Unused policies can be accidentally attached, granting unintended permissions
- **Organization:** Keep your IAM console clean and manageable
- **Compliance:** Audit logs are cleaner when unused policies are removed
- **Best Practice:** Following the principle of least privilege includes removing unnecessary permission definitions

---

Key Learnings

1. IAM Policy Structure

IAM policies use JSON format with these key components:

- **Version:** Policy language version (always use "2012-10-17")
- **Statement:** Array of permission statements
- **Effect:** "Allow" or "Deny" (Deny always wins)
- **Action:** AWS service actions (e.g., `ec2:StopInstances`)
- **Resource:** AWS resources the policy applies to
- **Condition:** Optional logic for conditional permissions

2. Tag-Based Access Control

Tags are powerful tools for:

- Organizing resources
- Cost allocation and tracking
- Conditional access control in IAM policies
- Automation and scripting

Best practices:

- Use consistent naming conventions (e.g., `Env` vs `Environment`)
- Apply tags consistently across all resources
- Protect tags from unauthorized modification (like we did with the Deny statement)

3. Principle of Least Privilege

Grant users only the permissions they need to do their job:

- Development interns need full access to dev resources
- They need read-only access to view production (for learning/context)
- They should not modify production or change tags

4. IAM Best Practices Demonstrated

- [x] Use groups instead of attaching policies directly to users
- [x] Use account aliases for user-friendly sign-in URLs
- [x] Test policies thoroughly before production use
- [x] Use explicit deny statements to enforce critical restrictions
- [x] Clean up unused users, groups, and policies
- [x] Use conditions in policies for context-aware permissions

5. Security Insights

- **Defense in Depth:** Multiple layers of security (tags + policies + groups)
- **Fail-Safe Defaults:** Resources are secure by default, permissions must be explicitly granted
- **Audit Trail:** All IAM actions are logged in CloudTrail
- **Separation of Duties:** Different users have different permission levels based on roles

---

Conclusion

This project successfully demonstrated how to implement secure access control in AWS using IAM. I learned how to:

- Launch and tag EC2 instances for environment-based access control
- Write custom JSON IAM policies with conditions
- Create IAM users and user groups following best practices
- Implement tag-based conditional access control
- Test security policies to verify they work as intended
- Clean up resources properly to maintain account hygiene

Real-World Applications:

This type of IAM configuration is used by companies of all sizes to:

- Safely onboard interns and contractors with limited access
- Separate development, staging, and production environments
- Prevent accidental deletion or modification of critical resources
- Enable self-service infrastructure management within guardrails
- Maintain security compliance and audit requirements

Time Investment: ~40 minutes
Cost: $0 (Free tier eligible)
Security Value: Priceless 🔒

Additional Resources

- AWS IAM Documentation
- IAM Best Practices
- IAM Policy Simulator
- AWS Tagging Strategies

Project Completed: December 2025
Author: YOUHAD AYOUB
Region: eu-west-3 (Paris)
NextWork Challenge: AWS Beginners Challenge - Project 2

This project was completed as part of the NextWork AWS Beginners Challenge. Special thanks to the NextWork community for their excellent guidance and project structure!