Project Title:

IAM ROLES AND SECURE ACCESS AUTOMATION ON AWS

Students Name: Onyinye Susan Nwosu

Institution: TechCrush HQ

Course: AWS Cloud Computing Capstone Project 4

Instructors: Temiloluwa Komolafe

Submission Date: November, 2025

INTRODUCTION

This project titled “IAM and Subnet Configuration Using AWS CLI” focuses on building automation and infrastructure management skills by handling all AWS resource creation through the command line interface rather than the AWS Management Console.

- The project involved creating IAM users and groups
- Creating Virtual Private Cloud(VPC) and subnet Configuration

Applying least privilege access and testing policies to confirm proper access control between two separate admin users — TestDBUser and TestWebUser.

This project was implemented entirely through AWS CLI to manage IAM resources and subnet configurations without relying on the graphical console.

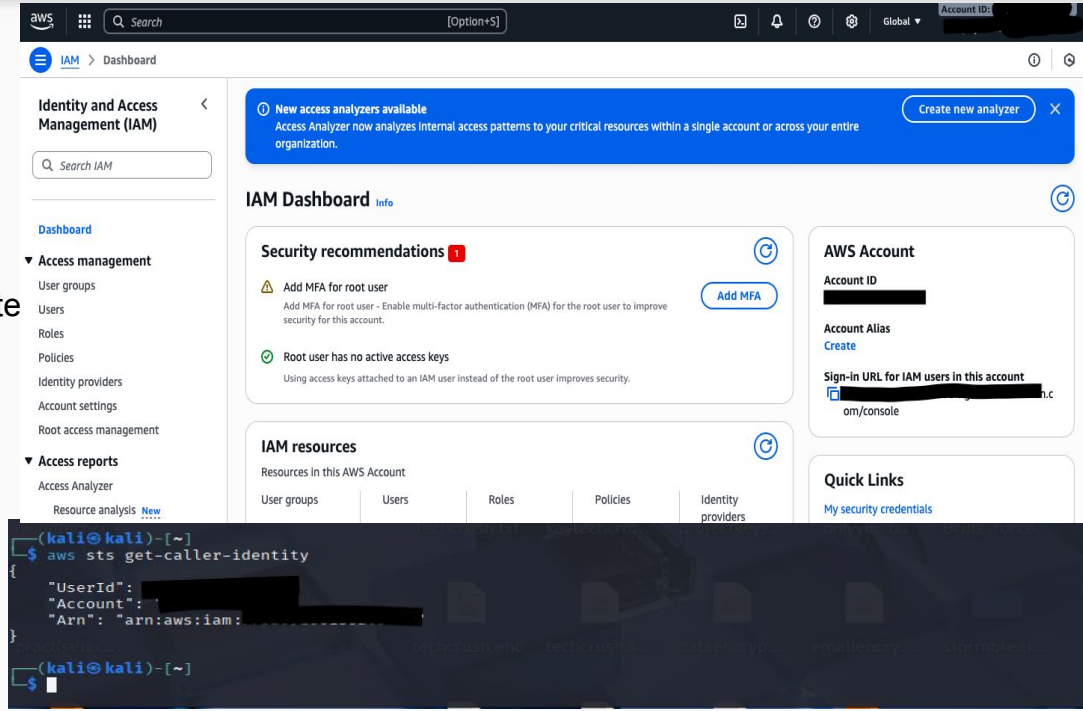
The idea was to simulate a real-world organization where two administrative teams — Database Administrators (DBAdmins) and Web Administrators (WebAdmins) — need different permissions to perform their roles.

Each group was assigned custom IAM policies restricting their actions to “read-only” access. By the end, all configurations were validated to ensure that each group only had access to the resources they were meant to manage.

Project Objectives

The objectives of this project were:
1. To configure AWS CLI with proper authentication for automation.
2. To create a VPC and two subnets (WebSubnet and DBSubnet).
3. To create IAM groups — DBAdmins and WebAdmins.
4. To create IAM users and assign them to their appropriate groups.
5. To develop and attach custom read-only JSON IAM policies.
6. To validate that permissions applied correctly through AWS CLI profiles.
7. To document the process, challenges, and solutions.

These objectives ensured a complete workflow from environment setup to security verification, covering key areas of cloud administration and identity management.



TOOLS AND ENVIRONMENT SET UP

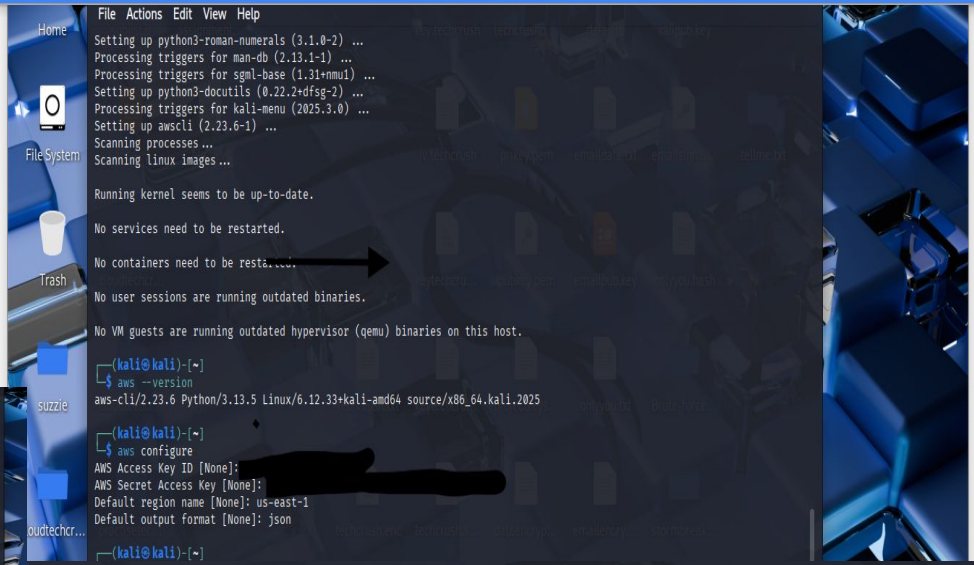
Tools and Services Used:

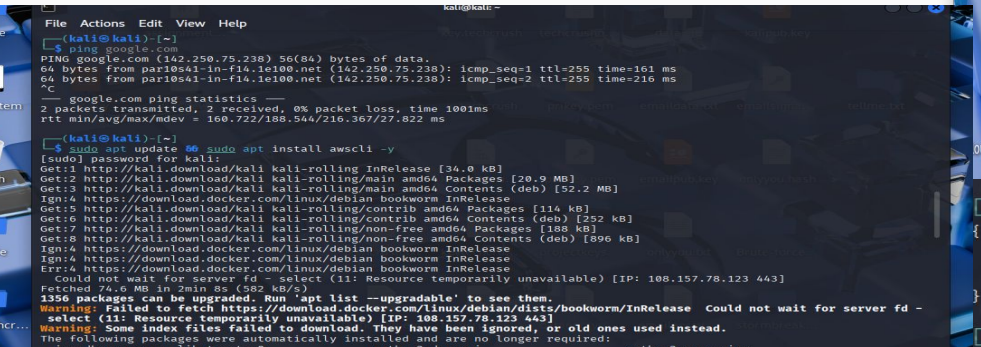
- AWS CLI: For managing IAM and VPC resources.
- AWS IAM: For creating users, groups, and custom policies.
- Amazon VPC: For subnet and network management.
- Linux (Kali): Used as the operating environment for CLI commands.
- JSON Policy Files: For defining permissions.

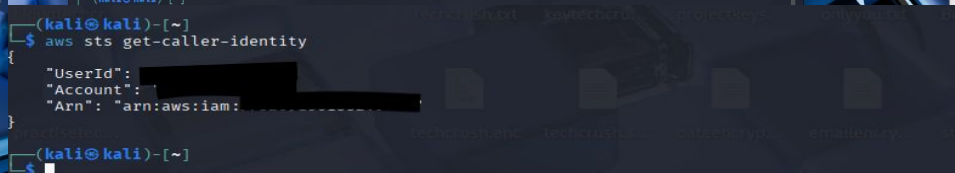
The setup was done in us-east-1 region, and the CLI was configured with named profiles to switch between users easily. This approach made automation repeatable and reduced dependency on the web console.

AWS CLI Installation and Configuration

AWS CLI was installed and configured using Access Keys generated from the IAM console.
After configuration, the following command was used to verify connection:
aws sts get-caller-identity
The command successfully returned the account ID and ARN, confirming t hat the CLI was authenticated.
This setup step was important because every other operation in the project depended on a working and secure CLI session.
During setup, a typing mistake in the key caused a failed configuration, which was fixed by regenerating new keys and reconfiguring.







VPC Creation and Verification

A new VPC named IAM-Automation-VPC was created with the CIDR block 10.0.0.0/16 to isolate the project's network.

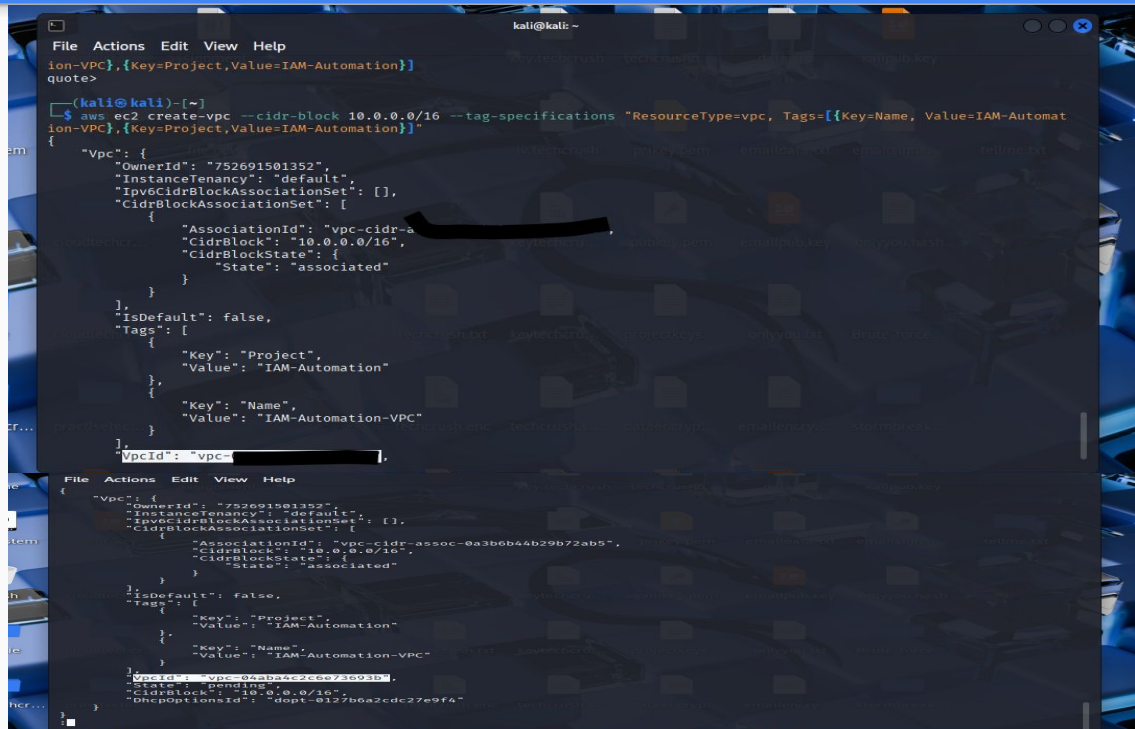
Using a dedicated VPC ensured that subnets could be configured cleanly without overlapping existing AWS r esources.

Verification was done using:

aws ec2 describe-vpcs

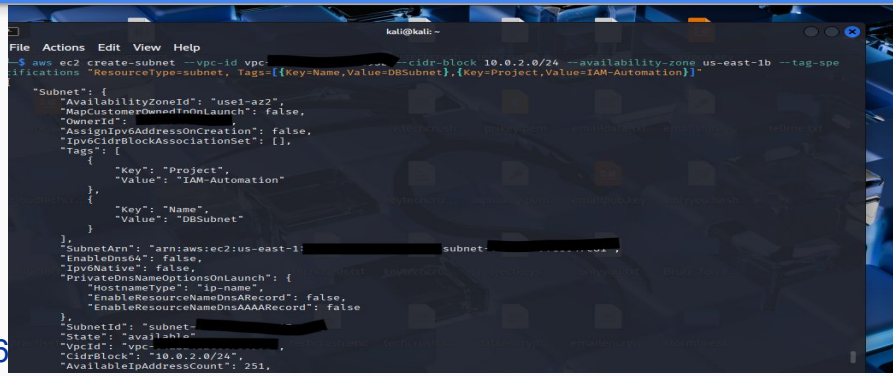
The output displayed the correct VPC ID and CIDR, confirming the setup.

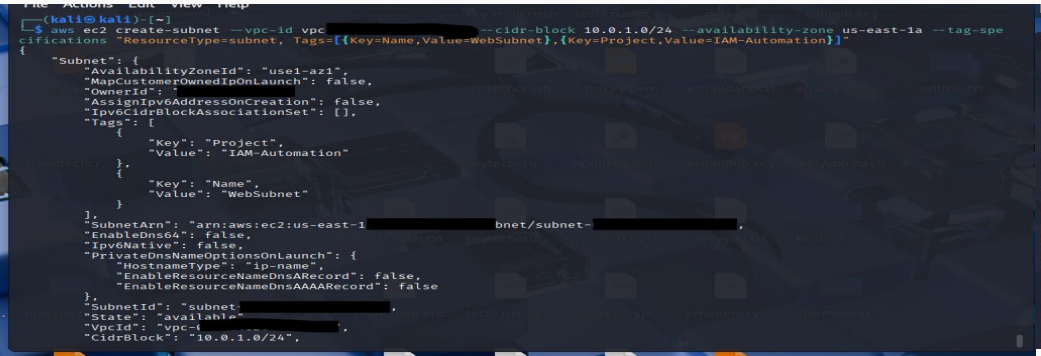
Earlier, a wrong VPC ID caused subnet errors, but this was fixed by verifying all available VPCs and consistently using the correct one.

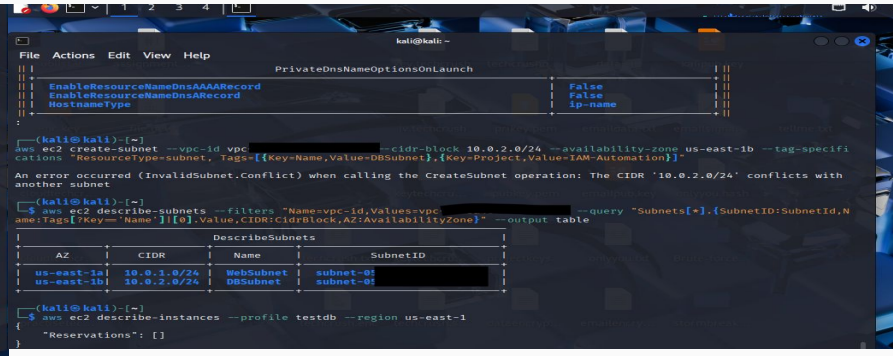


Subnet Creation

Two subnets were created under the project VPC:
WebSubnet: 10.0.1.0/24 in Availability Zone us-east-1a
DBSubnet: 10.0.2.0/24 in Availability Zone us-east-1b
Each subnet was tagged properly for easy identification.
Several errors occurred initially — such as invalid CIDR overlaps and command syntax issues when pasting multi-line CLI commands.
The issue was solved by switching to single-line commands and choosing unique CIDR ranges that matched the main VPC's 10.0.0.0/16

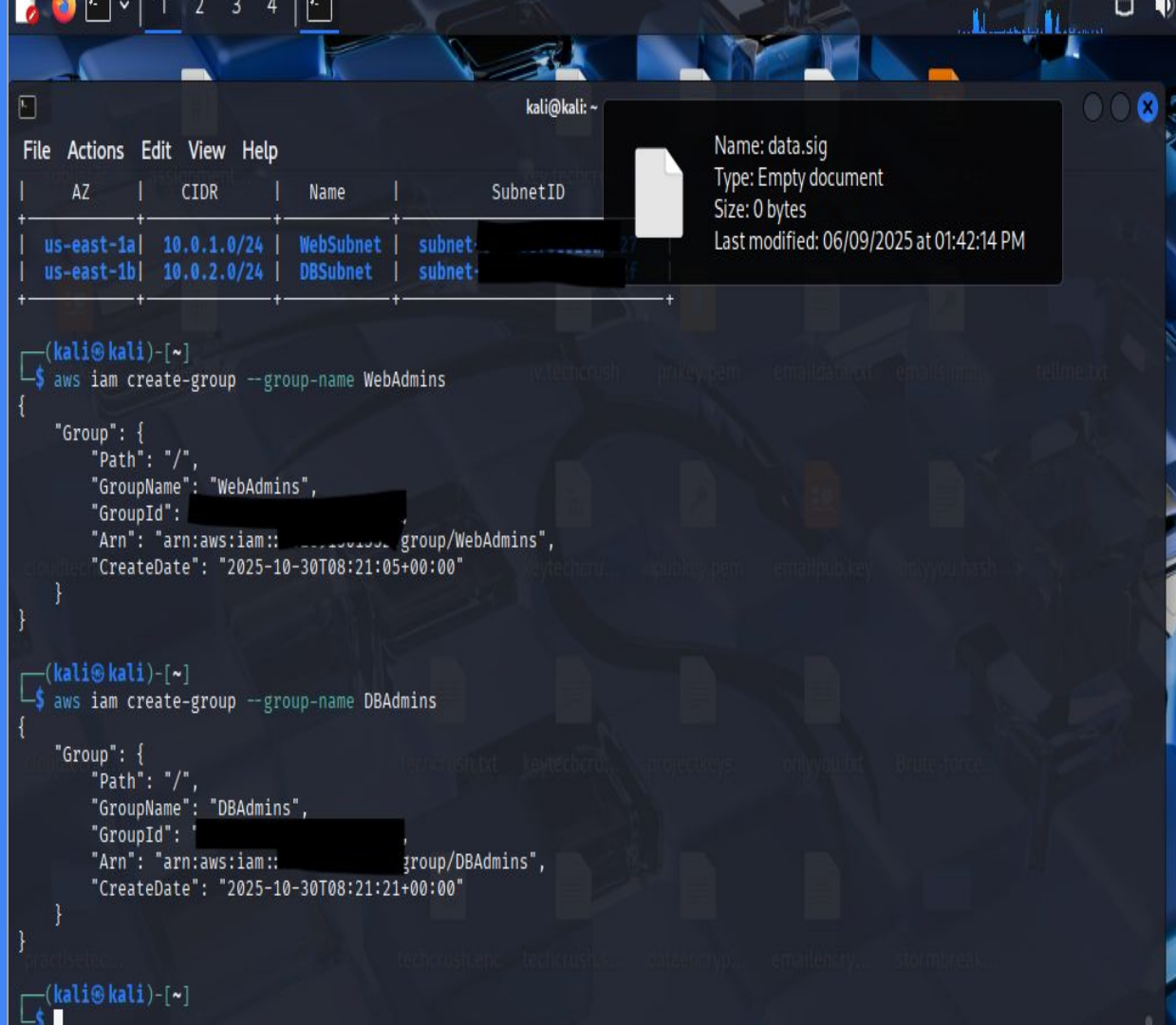






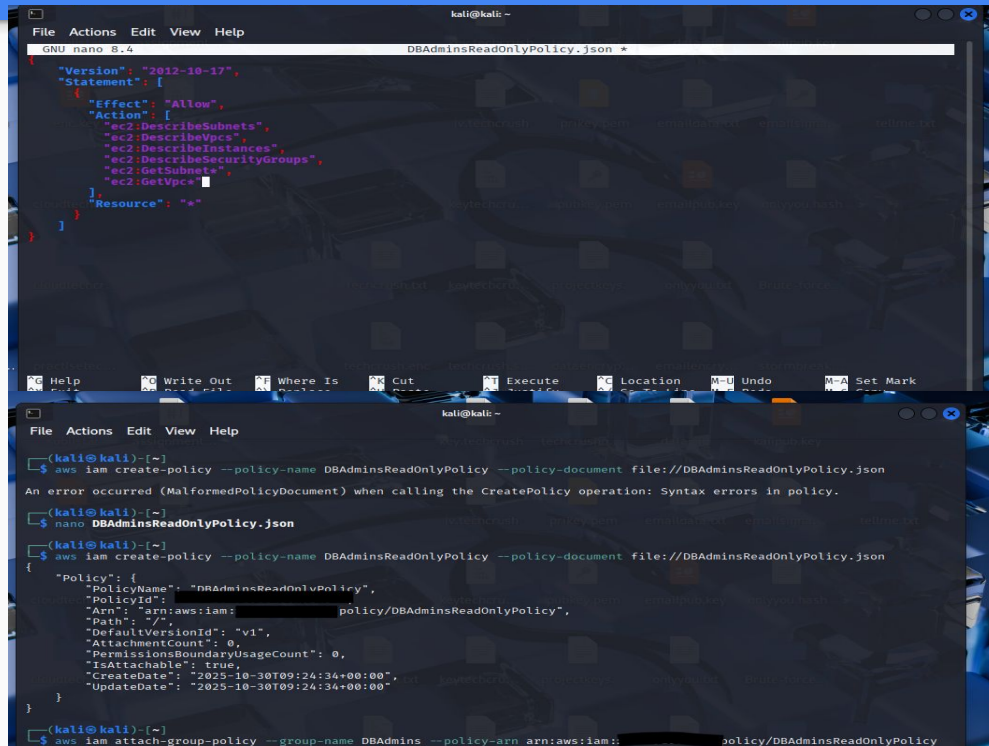
IAM Group Creation

Two groups were created:
DBAdmins – for database administrators
WebAdmins – for web administrators
The idea was to use role-based access control, so each team had permissions related only to their specific environment.
The CLI confirmed successful group creation, and both were visible in the IAM console.

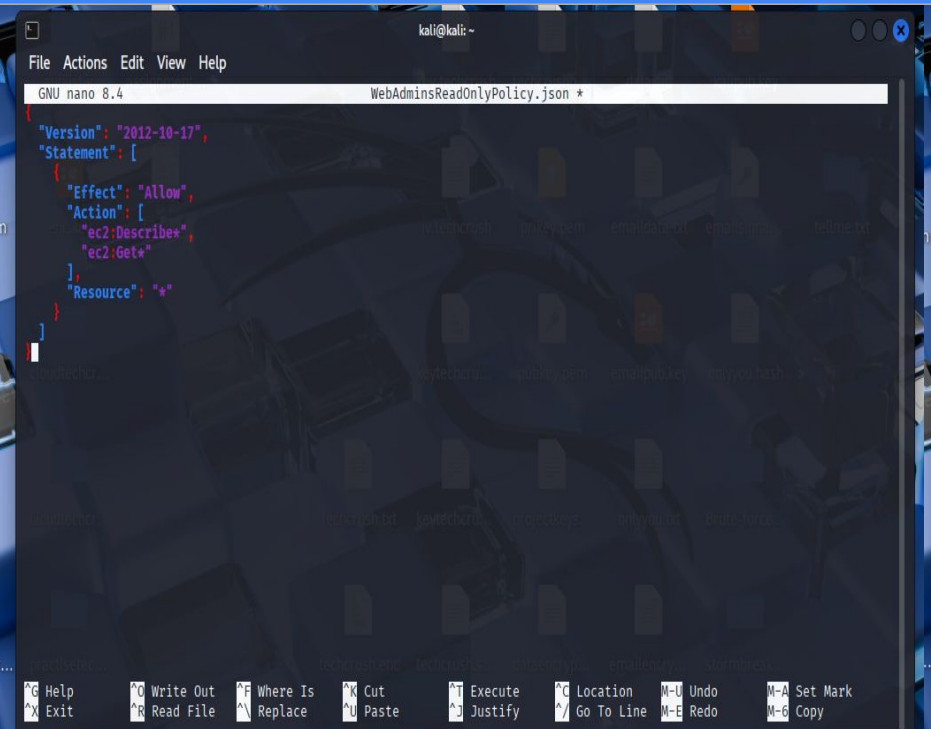


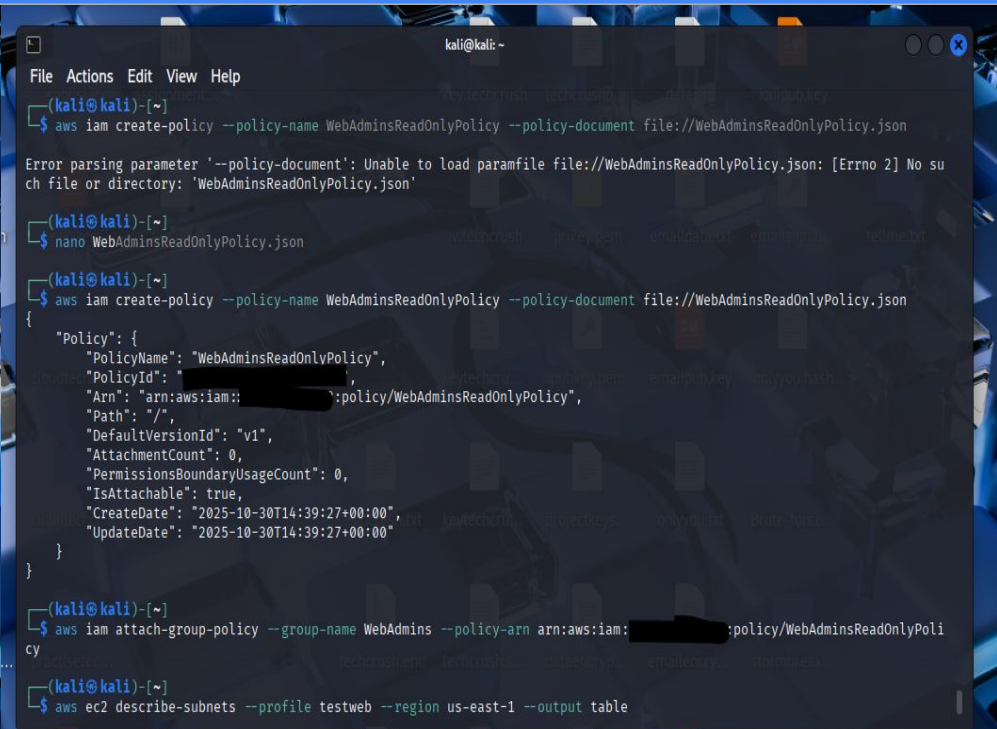
IAM Policies Creation

Two custom read-only policies were written in JSON format and attached to each group.
DBAdminsReadOnlyPolicy.json – Allowed describing subnets, VPCs, instances, and security groups.
WebAdminsReadOnlyPolicy.json – Allowed general describe and get actions for EC2 and VPC resources.
These policies enforced the principle of least privilege, ensuring no user could modify or delete AWS resources.
An initial attempt failed because the JSON file was attached incorrectly. This was corrected by creating the policy resource first, retrieving its ARN, and then attaching it properly.



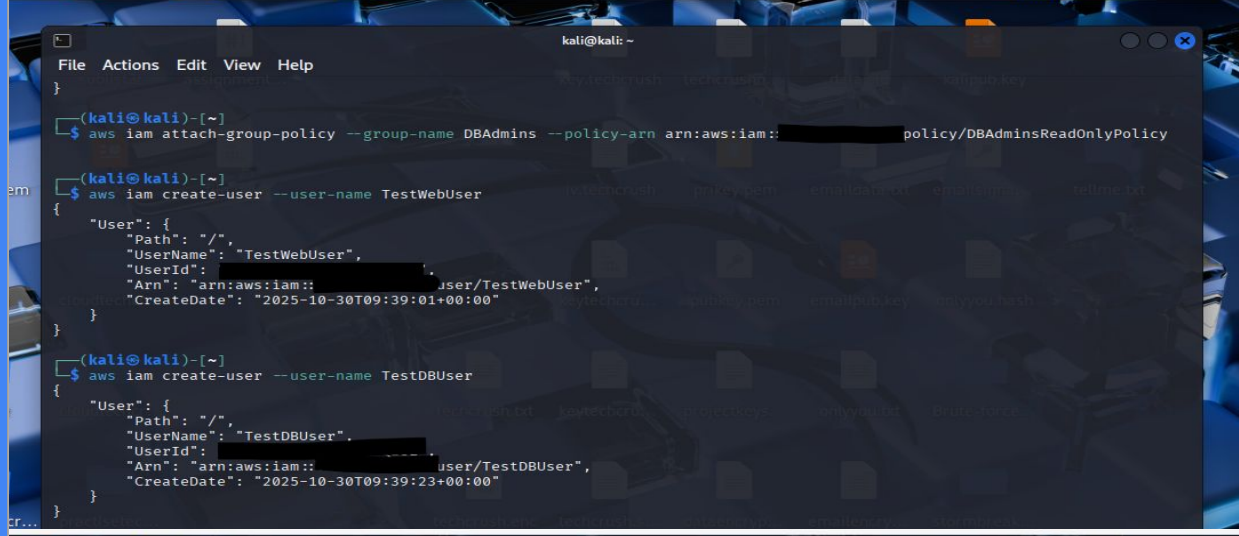
IAM Policies Creation (WebAdmin)

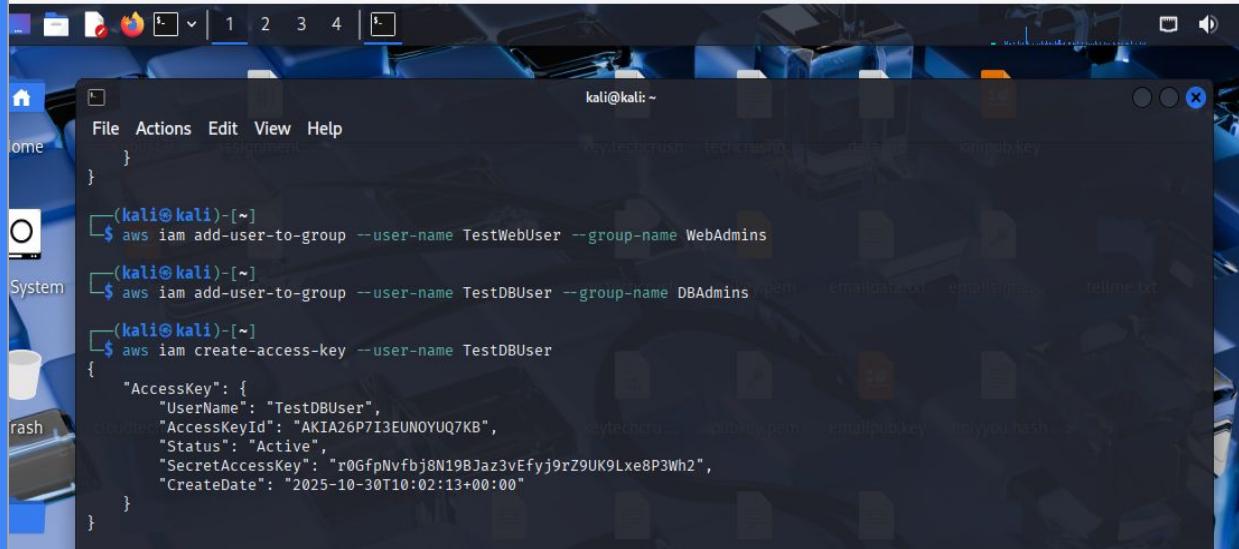




User Creation and Assignment

Two IAM users were created:
TestDBUser (for DBAdmins)
TestWebUser (for WebAdmins)
Each user was assigned to their respective group using AWS CLI.
This simulated how access control works in an organization where users are grouped by responsibility.
The creation process completed successfully, and user ARNs were recorded.





Access Key Generation and Profile Configuration

Each test user was given their own Access Key ID and Secret Access Key. Using those credentials, two named profiles were configured in the CLI:

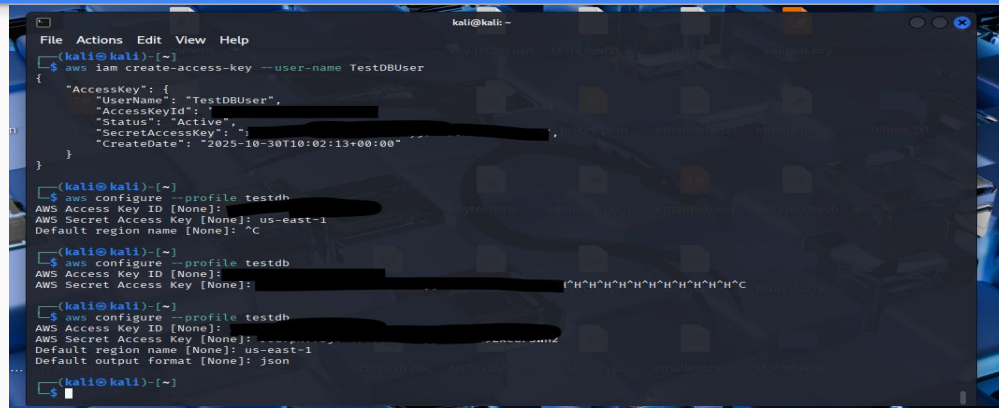
testdb

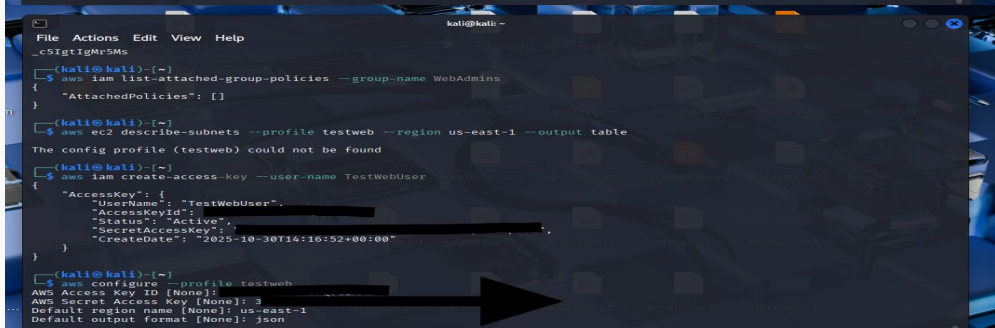
testweb

This setup allowed me to test each user's permissions independently without switching accounts.

A wrong access key was created initially but later deleted, and correct keys were generated again.

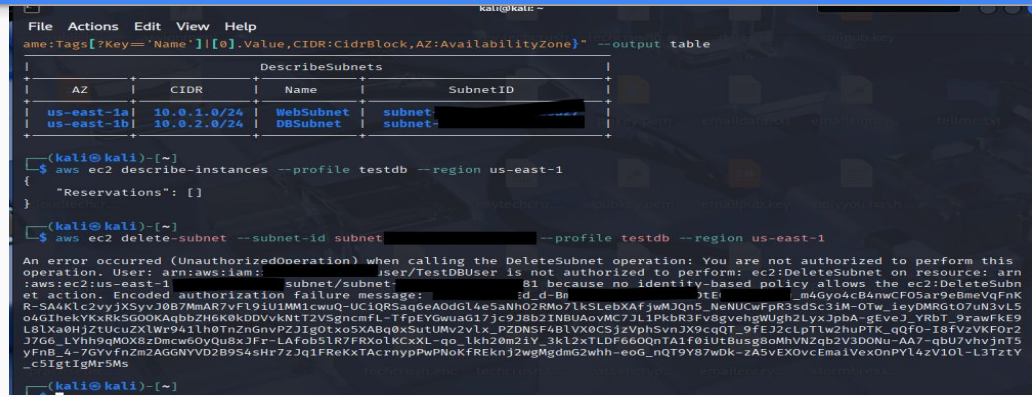
TestDBUser & TestWebUser Access key & attached

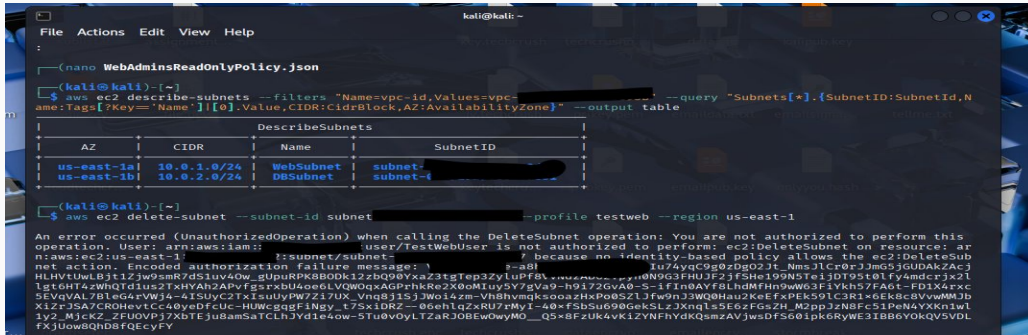




Permission Verification

For verification, both profiles were tested using AWS CLI commands.
For TestDBUser (DBAdmins):
Successfully described subnets and VPCs.
Delete subnet command returned AccessDenied confirming read-only enforcement.
For TestWebUser (WebAdmins):
Initially failed due to incorrect policy attachment but worked after fixing.
Could describe resources but was denied delete operations.
This confirmed that IAM group policies were properly enforced and no user could exceed their privileges.





Challenges and Solutions

Throughout implementation, several issues were faced:

1. CLI Syntax Errors: Resolved by using proper quotation marks and single-line format.
2. CIDR Conflicts: Solved by verifying VPC CIDR and assigning unique subnet ranges.
3. Wrong VPC ID: Identified correct VPC through describe-vpcs.
4. Policy JSON Errors: Fixed by reformatting JSON and verifying with aws iam create-policy.
5. Access Key Mistakes: Deleted unused keys and reconfigured profiles.

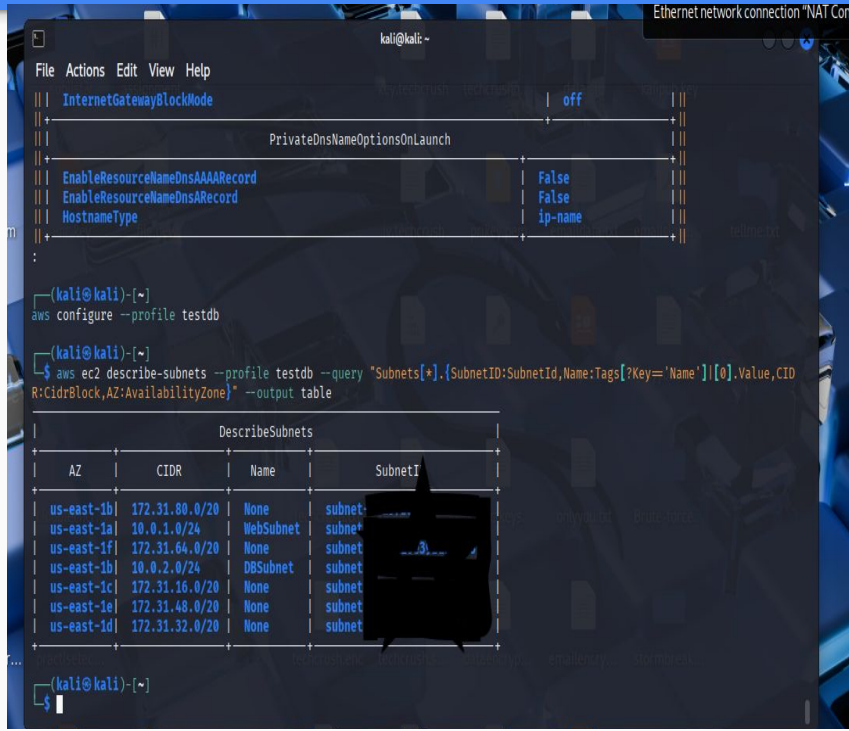
After troubleshooting, every step executed successfully and matched the intended outcome.

Final Environment State

Final environment included:

1. One working VPC (IAM-Automation-VPC)
2. Two Subnets (WebSubnet and DBSubnet)
3. Two IAM Groups (DBAdmins, WebAdmins)
4. Two IAM Users (TestDBUser, TestWebUser)
5. Two Policies (DBAdminsReadOnlyPolicy and
6. WebAdminsReadOnlyPolicy)

All permissions were verified successfully both test profiles had read-only access and could not perform any delete or modify operations.



Conclusion and Recommendations

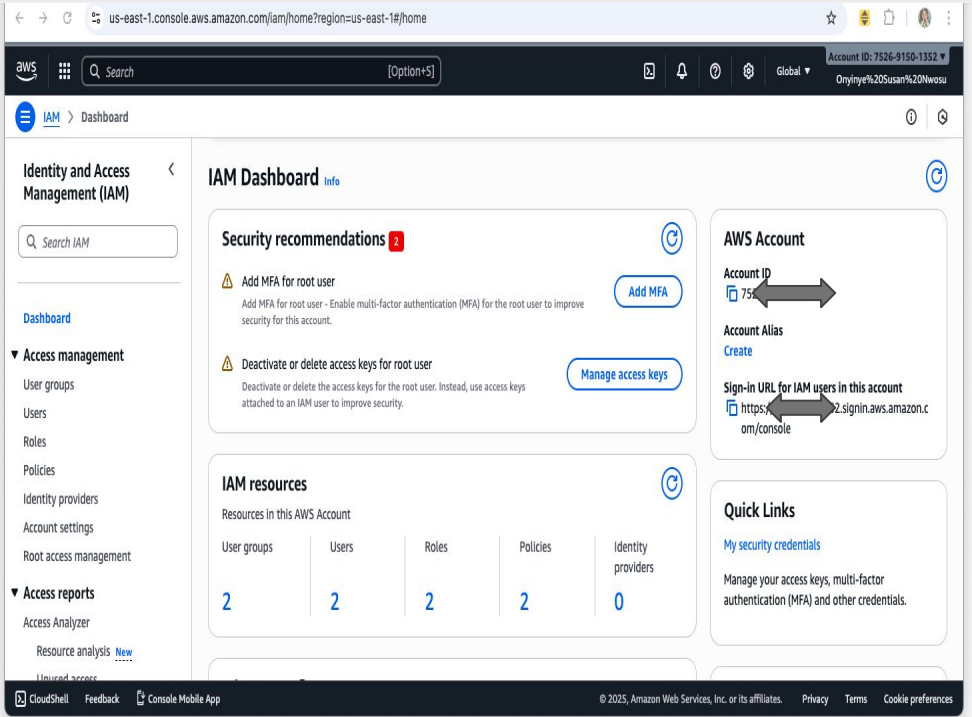
This project was successfully implemented using AWS CLI for automation.It demonstrated practical skills in IAM configuration, VPC management, and enforcing least privilege through custom Policies.

By working directly in the CLI, I learned how to control cloud infrastructure efficiently and securely.

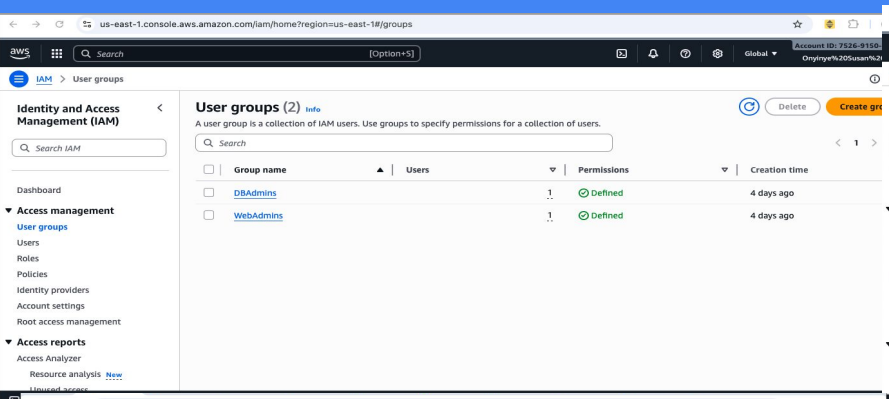
Recommendations:

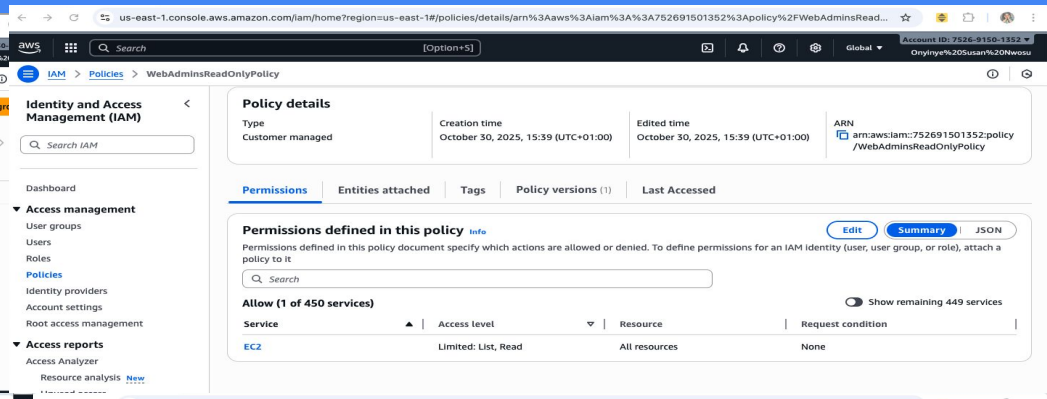
1. Always document VPC and subnet CIDRs to avoid overlap.
2. Store JSON policy files in version control for reuse.
3. Include cleanup scripts for scalability.
4. Use separate CLI profiles for dev, test, and production environments.

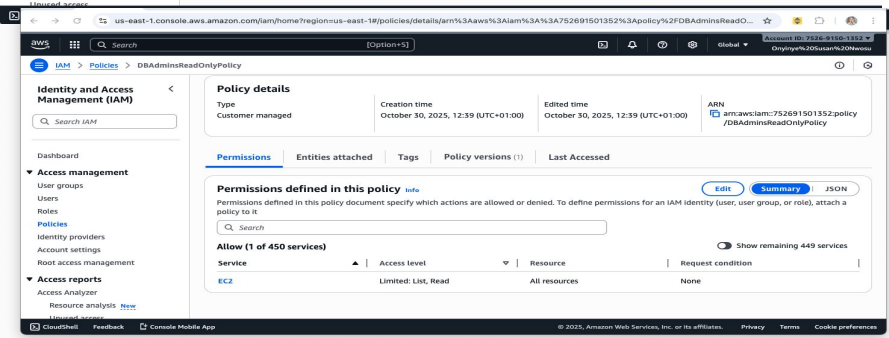
...

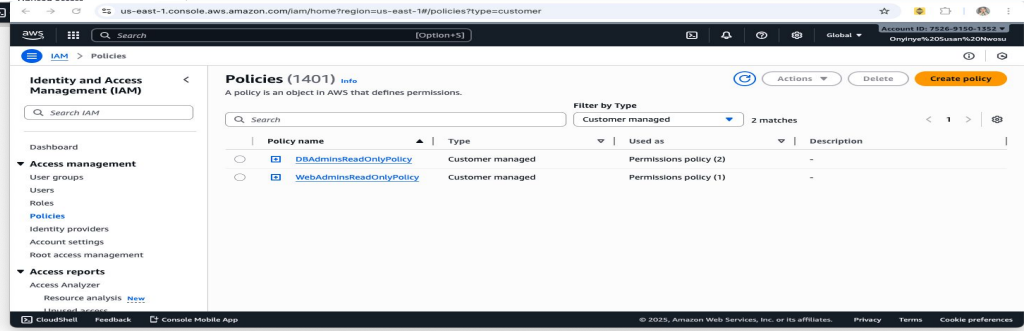


Conclusion and Recommendations

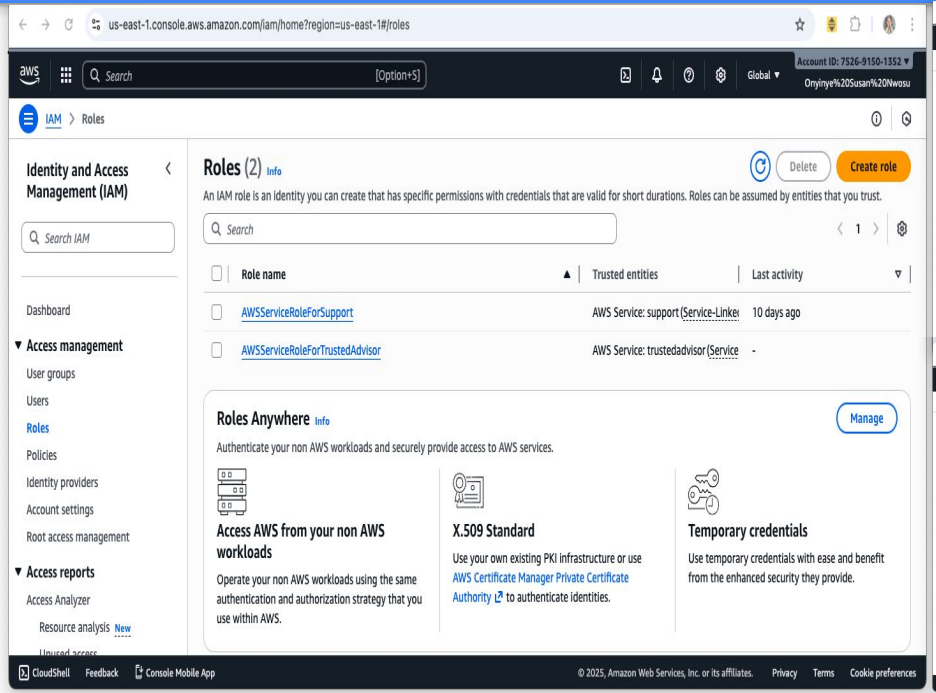


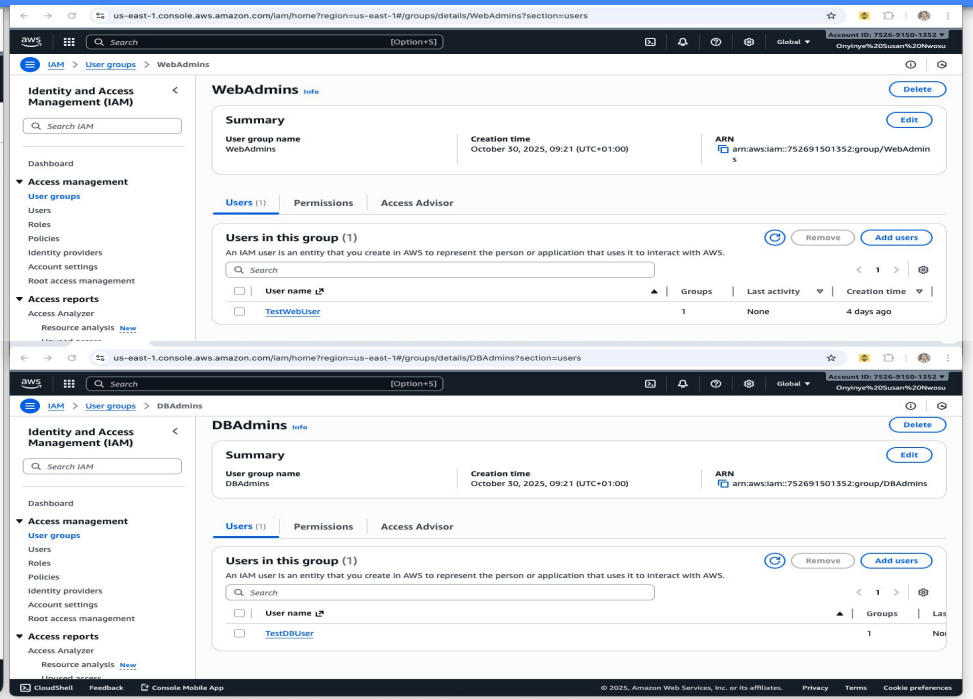






Conclusion and Recommendations





References

1. AWS IAM and VPC Documentation (2024)
2. AWS CLI User Guide
3. NIST (National Institute of Standards and Technology). Special Publication 800-53: Security and Privacy Controls for Federal Information Systems and Organizations. Retrieved from: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r5.pdf
4. TechCrush Cloud Academy Course Material
5. Kavis, M. (2023). Architecting the Cloud: Design Decisions for Cloud Computing Service Models (SaaS, PaaS, and IaaS). John Wiley & Sons.
6. Microsoft Learn (2024). Comparison of IAM Between AWS and Azure. Retrieved from: https://learn.microsoft.com/en-us/azure/active-directory/fundamentals/
7. CompTIA (2024). Security+ Certification Exam Objectives (SY0-701). Retrieved from: https://www.comptia.org/certifications/security