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

Table of Contents

1. Introduction

This project focuses on automating and managing AWS Identity and Access Management (IAM) roles and subnet configurations using the AWS Command Line Interface (CLI). The goal was to implement secure user and group access control while managing subnets within a Virtual Private Cloud (VPC) environment.

The project simulates a real-world enterprise setup where different administrative groups, Database Administrators (DBAdmins) and Web Administrators (WebAdmins) require distinct permissions and network resources. Each group was given custom IAM policies to control their level of access, ensuring a separation of duties and enhanced security.

By completing this project, I learned how to:

- Use the AWS CLI for creating and managing IAM users, groups, and policies.
- Configure subnets within a specific VPC and assign tags for project identification.
- Apply security best practices by implementing least privilege access.
- Troubleshoot permission and resource errors during the setup process.

This hands-on project bridges theoretical knowledge of AWS IAM and VPC networking with practical, command-line-based execution—essential skills for any cloud or cybersecurity professional.

2. Project Objectives

The objectives of this project were to:

i. Create two IAM groups: DBAdmins and WebAdmins to represent different administrative roles.

ii. Create and configure IAM users (TestDBUser and TestWebUser) and assign them to their appropriate groups.

iii. Create custom IAM policies granting read-only access to specific AWS resources for each group.

Iv. Create and configure two subnets (WebSubnet and DBSubnet) within a chosen VPC using AWS CLI.

v. Verify the network and IAM configurations through the test users and their respective AWS profiles.

vi. Identify and troubleshoot any command syntax or permission errors encountered during execution.

vii. Document every process, including challenges, corrections, and final outcomes.

These objectives helped ensure a structured approach to learning and applying both IAM access control and VPC subnet management using real AWS commands.

3. Tools and Environment Setup

Tool/Service	Description	Purpose in the Project
AWS CLI	Command-line tool for managing AWS services.	Used to create, configure, and verify IAM and subnet resources.
AWS Management Console	Web interface for AWS services	Used occasionally for verification and monitoring.
IAM (Identity and Access Management)	AWS service for access control.	Used for creating users, groups, and custom policies.
Amazon VPC	Service for creating isolated virtual networks.	Used to create and manage subnets (WebSubnet and DBSubnet).
JSON Files	Used for defining IAM policy documents.	Contained access permissions for both groups.
AWS Profiles	Named CLI configurations.	Used to differentiate between user sessions (testdb, testweb).

Configuration Item	Details
Operating System	Linux (Kali)
AWS Region	Us-east-1
VPC ID Used	vpc-04aba4c2c6e73693b
CIDR Blocks	172.31.0.0/16, 10.0.0.0/16
Subnets	WebSubnet, DBSubnet
Availability Zones	Us-east-1a, us-east-1b
IAM Users	TestDBUser, TestWebUser
IAM Groups	DBAdmins, WebAdmins
Policy Files	DBAdminsReadOnlyPolicy.json, WebAdminsReadOnlyPolicy.json

Expected Outcomes

By the end of the project, the following outcomes were expected:

- Two properly configured IAM groups with attached custom policies.
- Two IAM users, each restricted to specific administrative tasks within AWS.
- Two subnets successfully created within the chosen VPC, tagged correctly.
- Validation of user permissions through AWS CLI commands.
- Full documentation of all steps, including troubleshooting details, lessons learned, and outcomes.

4. Methodology / Step-by-Step Implementation

4.1 AWS CLI installation & initial configuration

What I did (summary)

- Installed AWS CLI on the Kali VM and verified the installation.
- Configured AWS CLI with programmatic credentials (Access Key ID and Secret Access Key) using a named profile for testing.
- Confirmed the CLI connection to AWS using the security token service to get caller identity.

Why

I needed a working, authenticated CLI environment to run all automation steps consistently from the terminal.

What happened / outputs to record

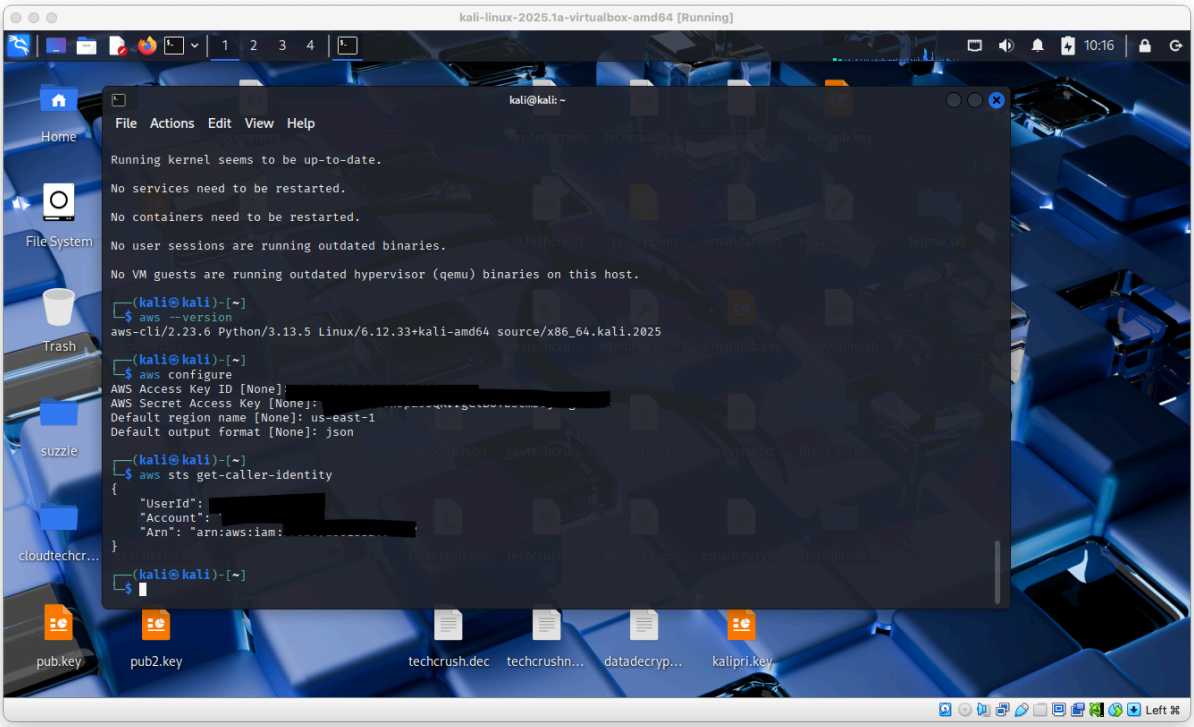
- aws --version confirmed installation.

- aws configure (using the saved Access Key ID / Secret Access Key) created the testdb profile.
- aws sts get-caller-identity returned the account ID and ARN — proving CLI authentication.

Challenges & fixes

- First credential entry failed (typing mistake); solution: regenerated keys in the AWS console and reconfigured the CLI with the correct keys.

terminal showing aws --version & output showing configured profiles & aws sts get-caller-identity.



4.2 VPC creation and verification

What i did (summary)

- Created a dedicated VPC for the project (named IAM-Automation-VPC) and recorded its VPC ID.
- Verified available VPCs to ensure we targeted the correct one throughout the project.

Why

A dedicated VPC isolates the project network and ensures clean addressing for project subnets.

What happened / outputs to record

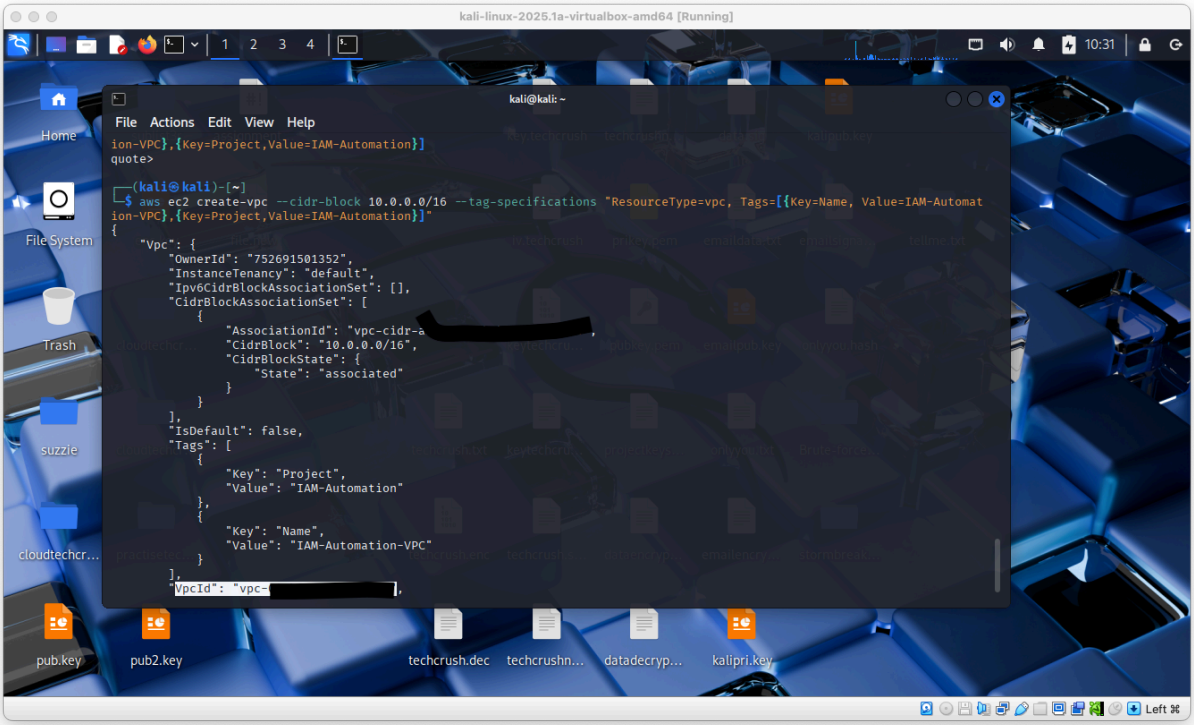
- VPC created with CIDR 10.0.0.0/16.

- The VPC ID (the working, final one) was identified and used for all further network steps.

Challenges & fixes

- During earlier steps we referenced the wrong VPC ID and flung into CIDR/subnet errors. Solution: described VPCs and picked the correct VPC ID (the one with 10.0.0.0/16) and used that consistently thereafter.
- Note: At one point the account showed a second VPC (default or previous), but we used only the final working VPC in the report.

AWS Console (or CLI output) showing the VPC name, ID and 10.0.0.0/16 CIDR & terminal output confirming VPC creation and returned VpcId.



4.3 Subnet creation (WebSubnet and DBSubnet)

What I did (summary)

- Created two subnets inside the project VPC:
  - WebSubnet — intended CIDR 10.0.1.0/24 in an AZ near us (us-east-1a or the account's available AZ).
  - DBSubnet — intended CIDR 10.0.2.0/24 in another AZ (us-east-1b).
- Verified the subnets were present and correctly tagged with Name=WebSubnet and Name=DBSubnet.

Why

Separate subnets isolate web-facing and database resources; tags make resource grouping and cleanup simple.

What happened / outputs to record

- WebSubnet created successfully and tagged as WebSubnet.
- DBSubnet creation had multiple attempts:
  - We initially hit “invalid subnet range” / overlap errors when CIDR or VPC ID mismatched; this led to a cleanup cycle removing transient/incorrect subnets and recreating the correct ones.
  - Final state: both WebSubnet and DBSubnet present and visible in the VPC.

Challenges & fixes (detailed)

Problem 1: Backslash-style, multi-line shell syntax sometimes failed when pasted in Kali led to command parse errors.

Fix: We switched to single-line CLI invocations when necessary and, later, used the CLI’s simplest tag/string quoting that worked on Kali. (In the final doc we avoid repeating the exact broken multiline forms.)

Problem 2: CIDR overlap / invalid range errors.

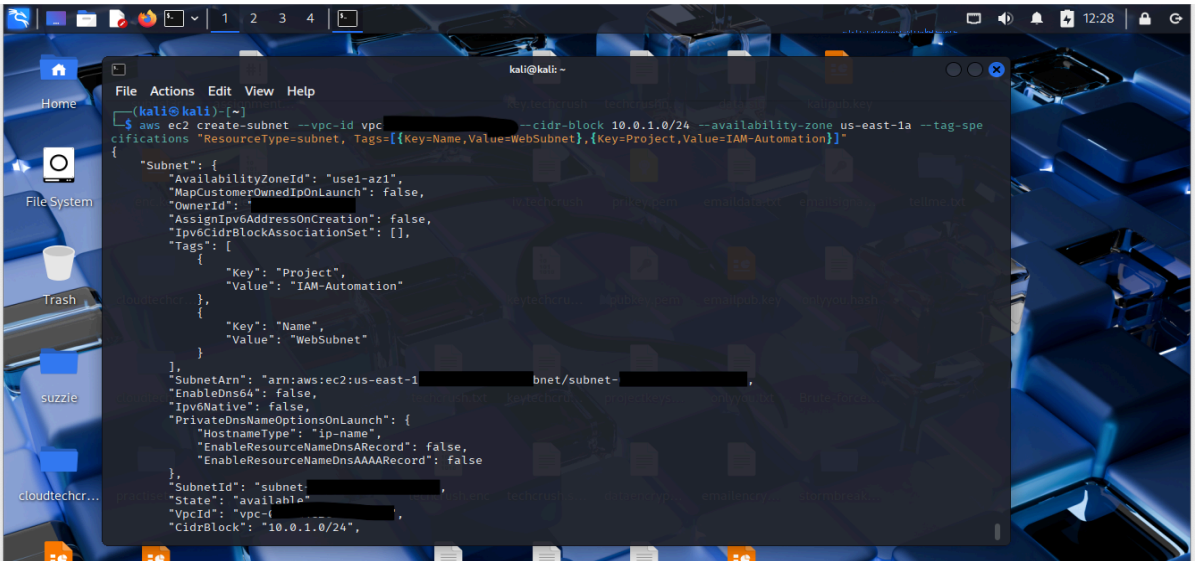
Fix: Verified the VPC CIDR (10.0.0.0/16) and checked existing subnets; chose unique /24 blocks that fit (10.0.1.0/24 and 10.0.2.0/24).

Problem 3: Some created subnets persisted from earlier attempts and confused results.

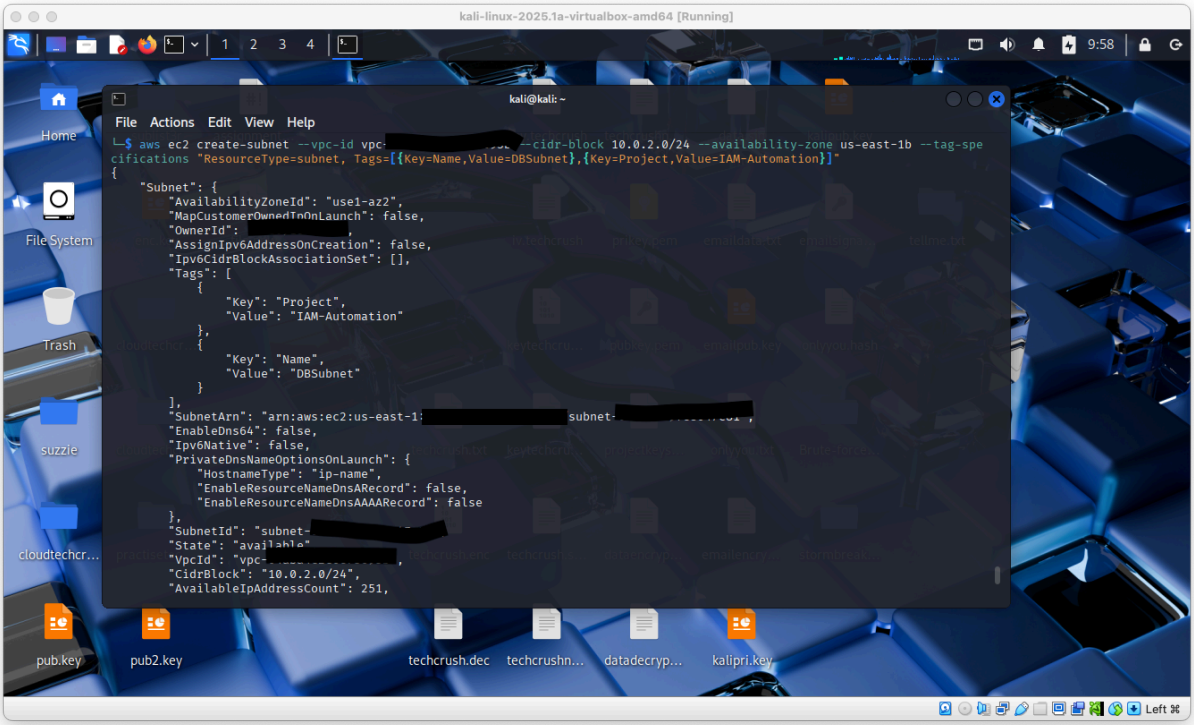
Fix: Systematically deleted all subnets on the VPC, then recreated WebSubnet and DBSubnet cleanly, verifying after each deletion/create cycle.

CLI output (describe-subnets) showing both subnets and their IDs.

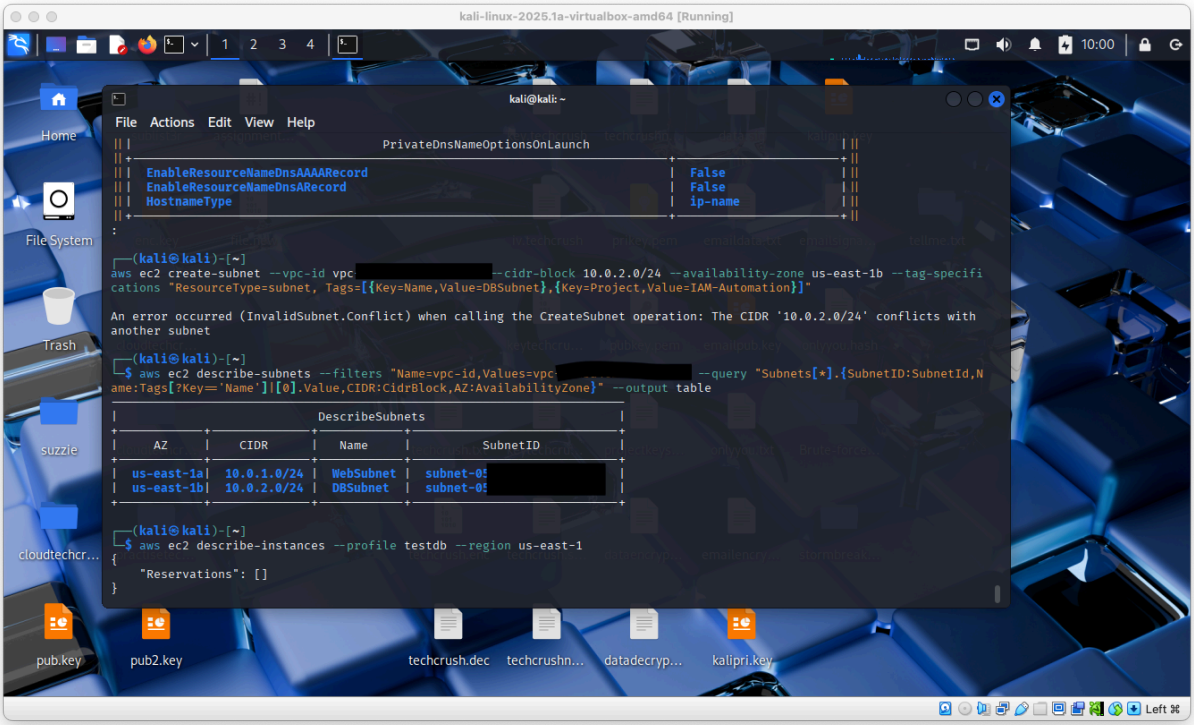
Web Subnet created



DB Subnet created



Console subnets view showing both WebSubnet and DBSubnet with CIDRs and AZs.



4.4 IAM groups creation

What I did (summary)

- Created two IAM groups: DBAdmins and WebAdmins.

Why

Groups provide role-based permission assignment and make it easy to manage multiple users.

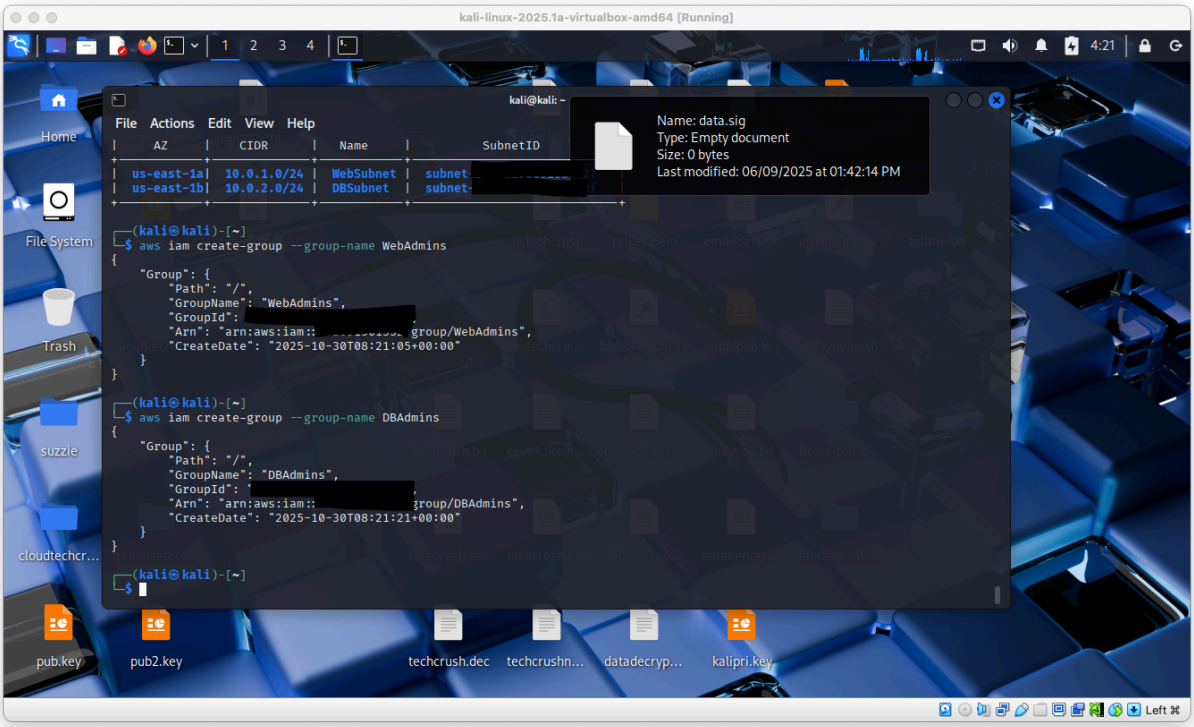
What happened / outputs to record

- Group creation confirmed in CLI and visible in IAM console.

Challenges & fixes

- None critical here — group creation was straightforward.

CLI output for creating groups.



4.5 IAM policies, exact JSONs used

These are the exact policy JSONs we used. Save each into the named file and create the policy from that file.

DBAdminsReadOnlyPolicy.json
(this is the working DB read-only policy I used)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeSubnets",
        "ec2:DescribeVpcs",
        "ec2:DescribeInstances",
        "ec2:DescribeSecurityGroups",
        "ec2:GetSubnet*",
        "ec2:GetVpc*"
      ],
      "Resource": "*"
    }
  ]
}
```

WebAdminsReadOnlyPolicy.json
(this is the working Web read-only policy I used)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:Describe*",
        "ec2:Get*"
      ],
      "Resource": "*"
    }
  ]
}
```

Why
These policy documents restrict each group to read-only EC2/VPC operations, preventing deletions or modifications.

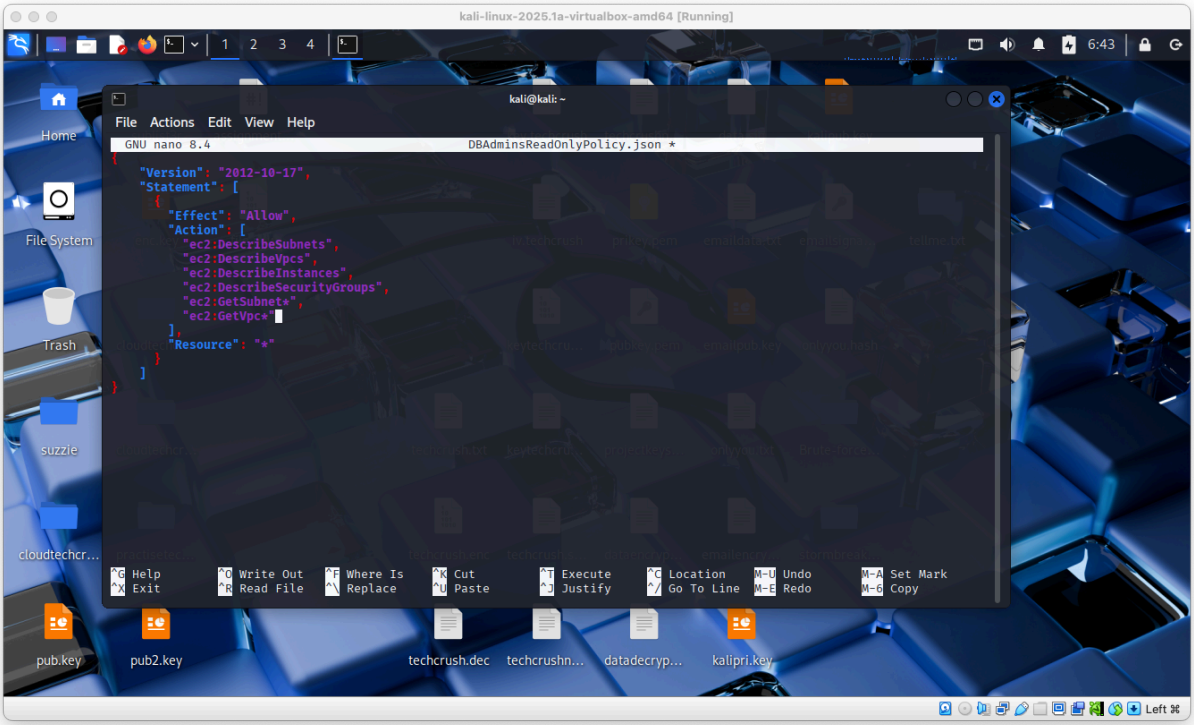
What happened / outputs to record

- Policies were created in IAM and returned Policy ARNs.
- Initial error: we once attached an incorrect JSON (or attempted to attach a file directly) — that caused the WebAdmins test to fail until corrected.

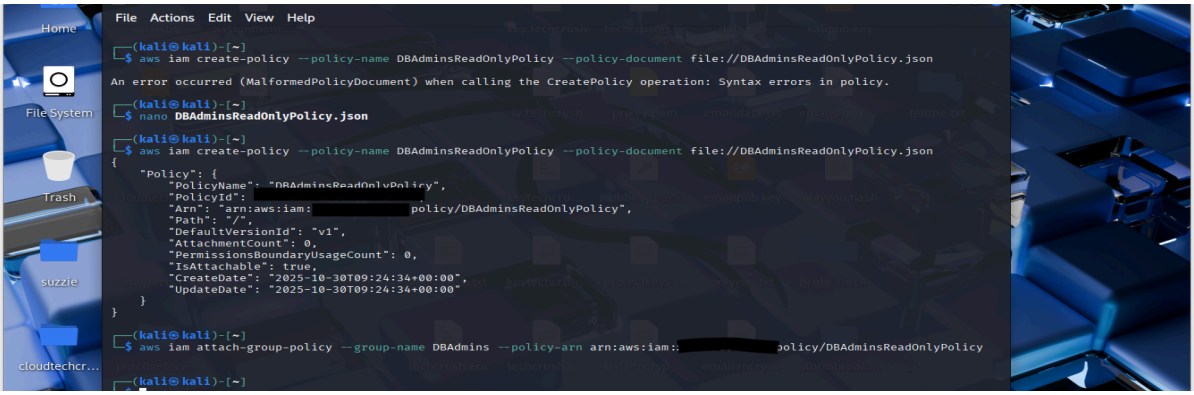
Challenges & fixes

Problem: Incorrect attach attempt (attached JSON path instead of an ARN).
Fix: Created the policy resource from the JSON file and attached the returned ARN to the group; detached any wrongly attached resource when needed.

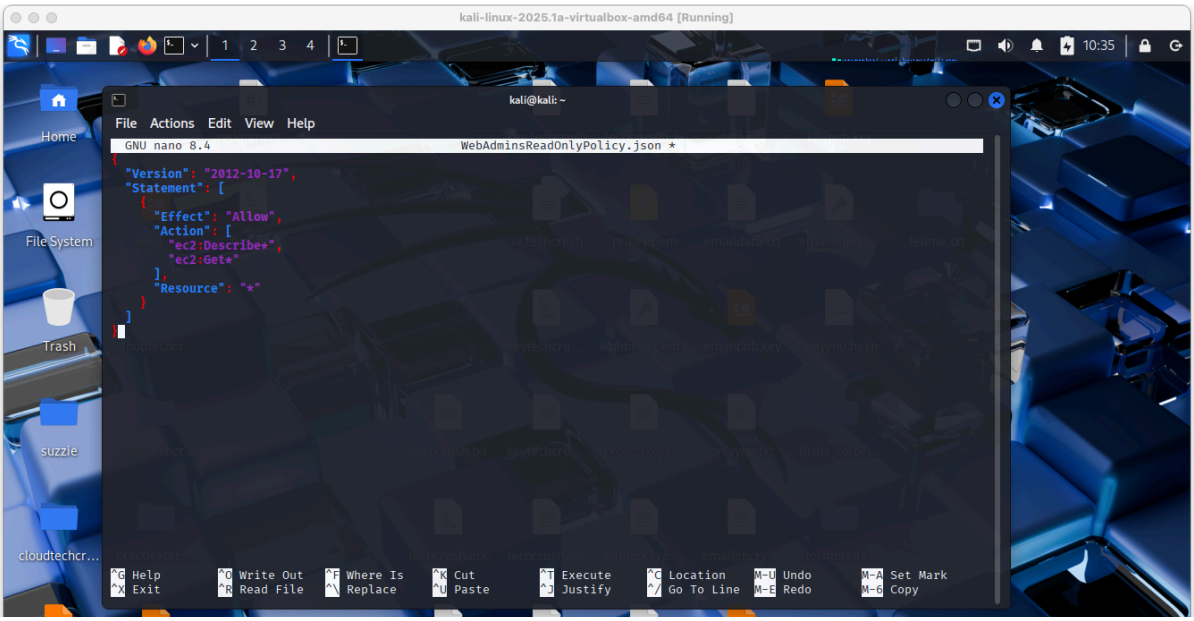
The DBAdmins JSON file contents in terminal



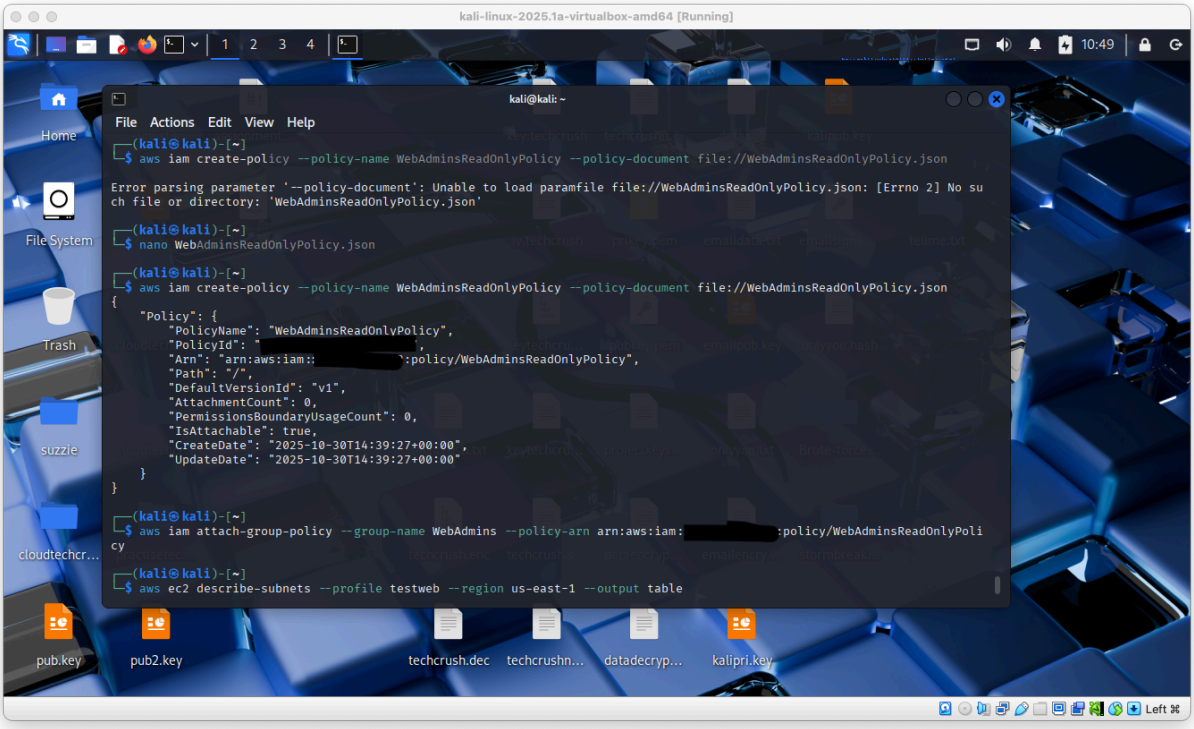
DBAdmins Policy created & attached on AWS & policy attached to its group.



The WEBAdmins JSON file contents in terminal



WEBAdmins Policy created & attached on AWS & policy attached to its group.



4.6 Adding test users and assigning to groups

What I did (summary)

Created two users: TestDBUser and TestWebUser.

Added TestDBUser to DBAdmins, and TestWebUser to WebAdmins.

Why

To validate group-based permissions with real credentials.

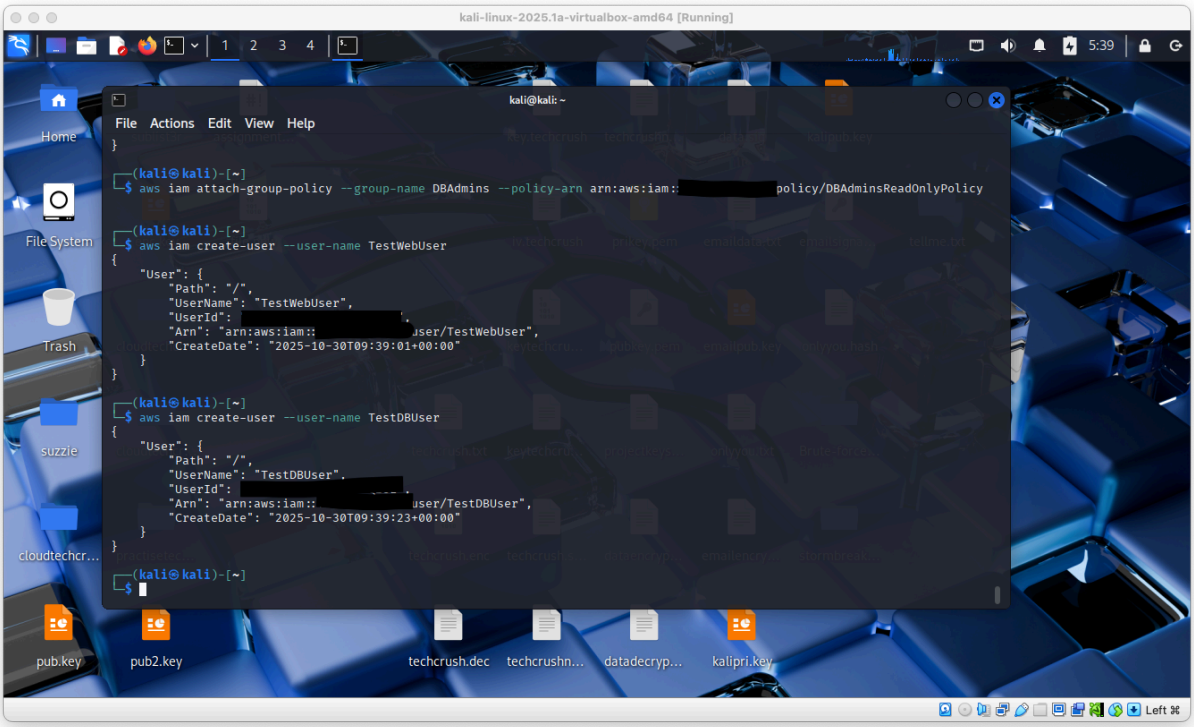
What happened / outputs to record

- Users created and visible in IAM.
- Each user added to its group successfully.

Challenges & fixes

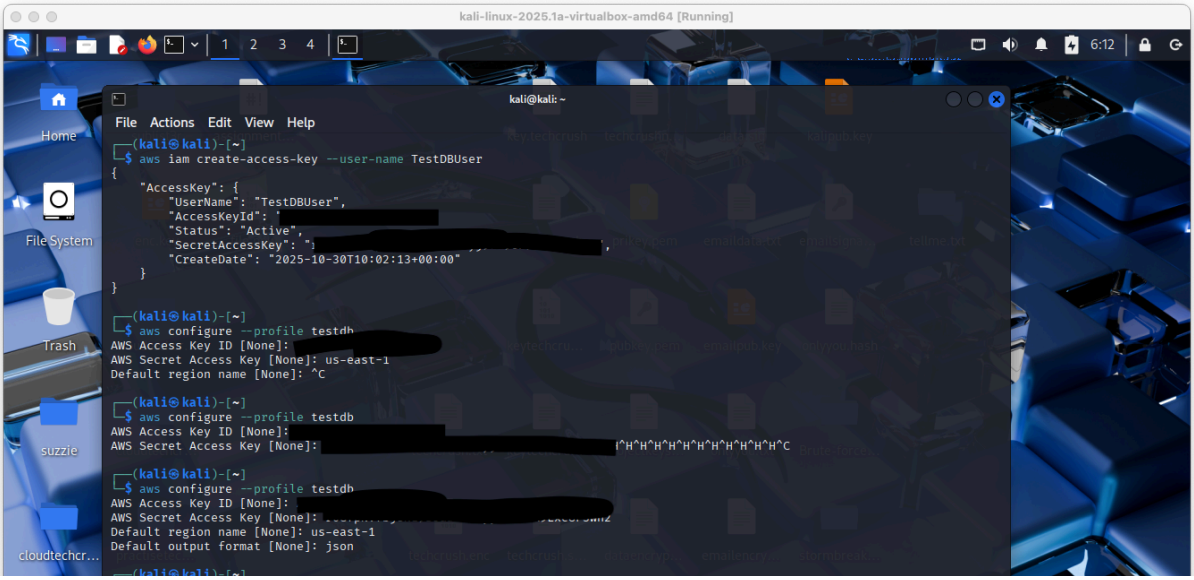
- No major errors on create; just be sure to copy the User ARN for records.

Both test users created

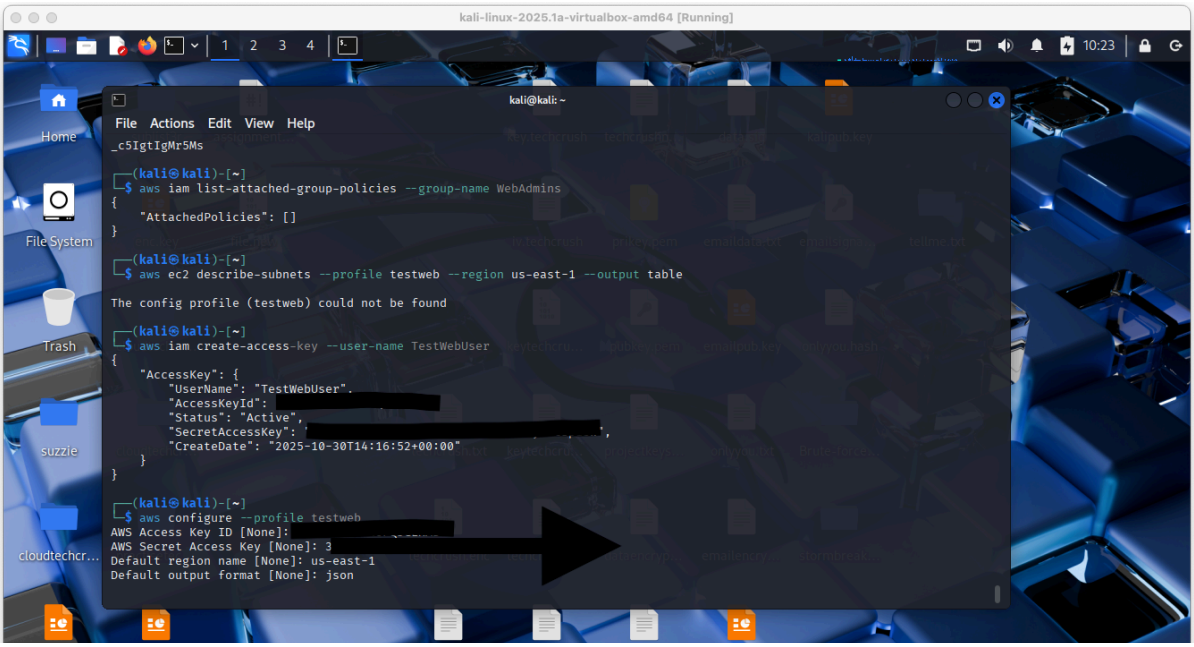


Both test users added to group & access key creation for TestDBUser

Access keys created & configure for TestDBUser



Access keys created & configure for TestWEBUser



4.7 Creating access keys and configuring test CLI profiles

What I did (summary)

- Created Access Key ID + Secret Access Key for each test user.

- Configured two named CLI profiles (testdb and testweb) to test permissions from the CLI.

Why

Profiles let us simulate each user's permissions via the CLI without using the main admin account.

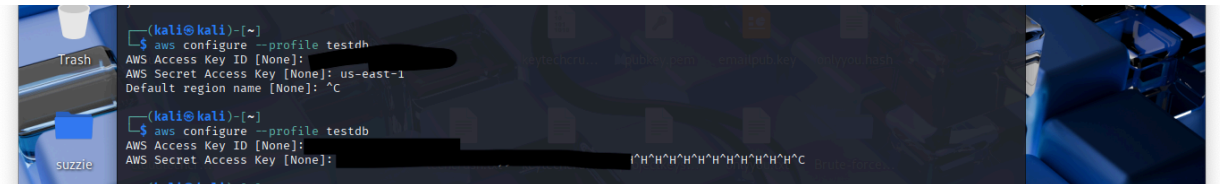
What happened / outputs to record

- Each create-access-key returned an AccessKeyId and SecretAccessKey.
- I used those to configure CLI profiles (one profile per user).
- At one point we aborted aws configure while entering keys — Ctrl+C cancels without saving; we re-ran configuration with correct values.

Challenges & fixes

- We accidentally made an access key we didn't want — we listed keys and deleted the unwanted key.
- Reminder: never paste keys into chat or save them in public places.

Access key output error



4.8 Permission verification (what I tested and the results)

What I tested (summary)

For each test profile, we ran read actions and attempted a write/delete action to ensure read-only enforcement.

DBAdmins (testdb) checks

- Read: described subnets and VPCs — success (DBSubnet visible to testdb).
- Write: attempted to delete DBSubnet — denied (AccessDenied / UnauthorizedOperation).

Result: DBAdmins policy enforced correctly

WebAdmins (testweb) checks

- Read: initially failed because the wrong policy had been attached; after fixing and reattaching the correct WebAdminsReadOnlyPolicy, testweb could describe subnets and VPCs (WebSubnet visible).
- Write: attempted to delete WebSubnet — denied.

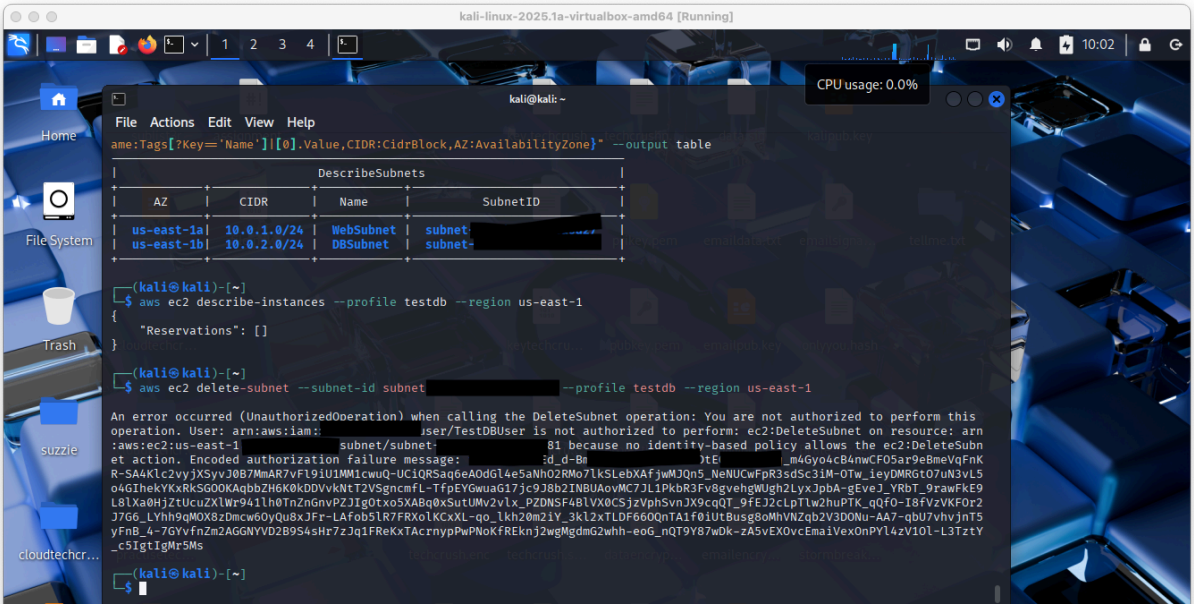
Result: WebAdmins policy enforced correctly after repair.

Challenges & fixes

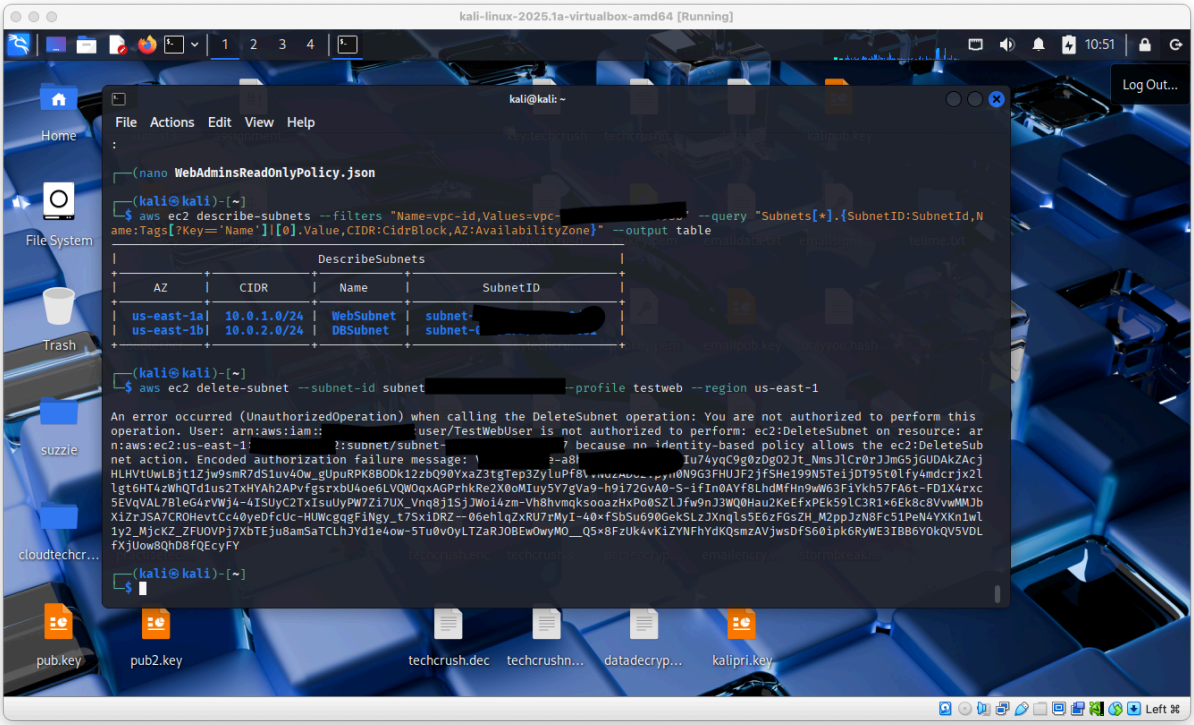
- **Propagation delay:** IAM policy changes sometimes took a minute to propagate — we waited and retried.
- **Wrong policy attached first:** Detached, recreated and attached correct policy ARN.

Testdb & TestWeb describe-subnets showing DBSubnet & WEBSubnet

Testdb attempt to delete subnet and AccessDenied output.



Testweb attempt to delete subnet and AccessDenied output.



4.10 Final outcome & state

What the final environment included

- One working VPC: IAM-Automation-VPC (CIDR 10.0.0.0/16) — use the final VPC ID in the report attachments.
- Two subnets:
  - WebSubnet — e.g., 10.0.1.0/24, AZ: us-east-1a (replace with your chosen AZ if different).
  - DBSubnet — e.g., 10.0.2.0/24, AZ: us-east-1b.
- IAM groups: DBAdmins, WebAdmins.
- IAM users: TestDBUser, TestWebUser assigned to their groups.
- Policies: DBAdminsReadOnlyPolicy and WebAdminsReadOnlyPolicy attached to respective groups.
- Verified behavior: read-only allowed; delete/modify forbidden for both test profiles.

5.0 Challenges and Solutions

During the implementation of the IAM Roles and Secure Access Automation Project, several challenges were encountered. Below is a summary of the main issues and how each was resolved:

Challenge	Description	Solution Implemented	Outcome
AWS CLI command syntax issues	Some AWS CLI commands failed due to incorrect syntax, especially when using quotation marks (' vs "), backslashes, and wrong line continuation symbols.	Adjusted command formatting to a single line using proper quotation marks. Verified correct syntax from working commands.	Commands executed successfully after correction.
Subnet creation conflicts	Attempts to create subnets with overlapping CIDR ranges resulted in “Invalid subnet range” errors.	Verified the main VPC CIDR and adjusted subnet ranges to avoid overlap (e.g., using 10.0.1.0/24 for WebSubnet and 10.0.2.0/24 for DBSubnet).	Both subnets created successfully and verified under the same VPC.
Multiple VPC confusion	AWS showed more than one VPC ID, causing difficulty in identifying the correct one used for the project.	Described and filtered all VPCs using aws ec2 describe-vpcs to confirm the correct project VPC by its tags and CIDR.	Correct VPC identified and reused consistently throughout setup.
Policy JSON errors	The IAM policy for DBAdmins initially failed due to syntax errors in the JSON document.	Recreated the JSON file using the correct format and indentation verified with aws iam create-policy.	Policy created successfully and attached to the intended IAM group.

Access key misconfiguration	A wrong access key was created for the test user, leading to wrong identity verification results.	Revoked the incorrect key and generated a new one. Updated AWS CLI profile credentials for the correct user.	Test user authenticated successfully with correct permissions.
Permissions validation errors	Some test users couldn't see or access subnet details due to missing role assignments.	Reattached IAM policies to groups and re-validated using separate profiles for testdb and testweb.	Permissions matched as intended DBAdmins had read-only access to DB resources, WebAdmins could manage web subnet.

6.0 Conclusion and Recommendations

The IAM Roles and Secure Access Automation Project was successfully completed using AWS infrastructure instead of Azure, while maintaining the original project objectives.

All configurations, including VPC setup, subnet segmentation, IAM group and policy creation, user configuration, and permission testing — were automated and verified. The project demonstrated a clear understanding of identity and access management, subnet segmentation, and secure access control principles within a cloud environment.

Key Takeaways:

- Proper tagging and documentation help prevent confusion when managing multiple cloud resources.
- Testing permissions with separate user profiles ensures that role-based access control works as intended.
- Using AWS CLI provides automation flexibility but requires careful attention to syntax.

Recommendations:

1. Always document VPC and subnet CIDRs during creation to prevent overlap or misconfiguration.

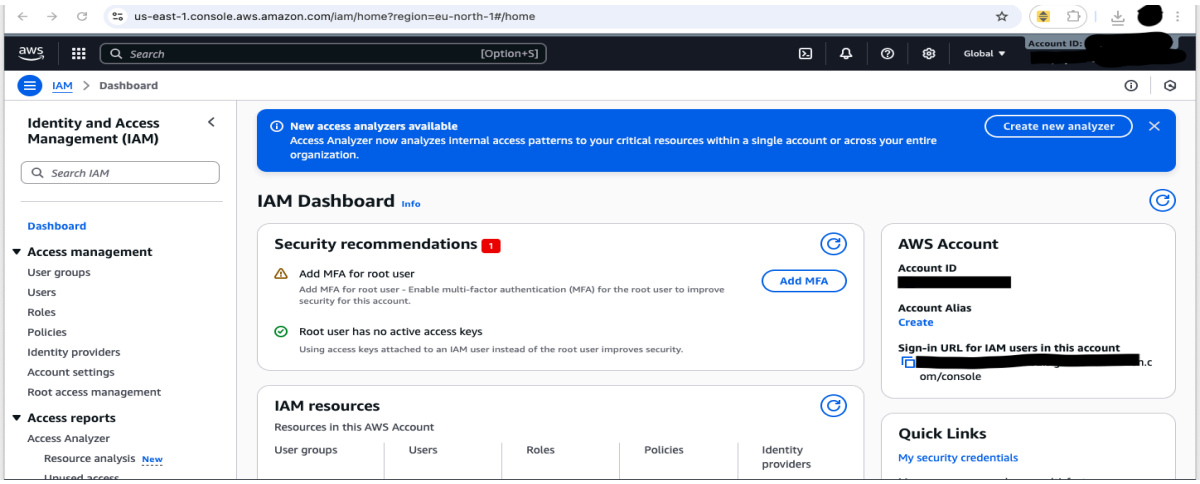
2. Store IAM policy files in a version-controlled directory (e.g., GitHub) for reuse and CI/CD integration.
3. Implement cleanup scripts (aws iam delete-*, aws ec2 delete-*) as part of automation pipelines for future scalability.
4. Use environment-specific profiles in the AWS CLI to separate development, testing, and production credentials.

7.0 References

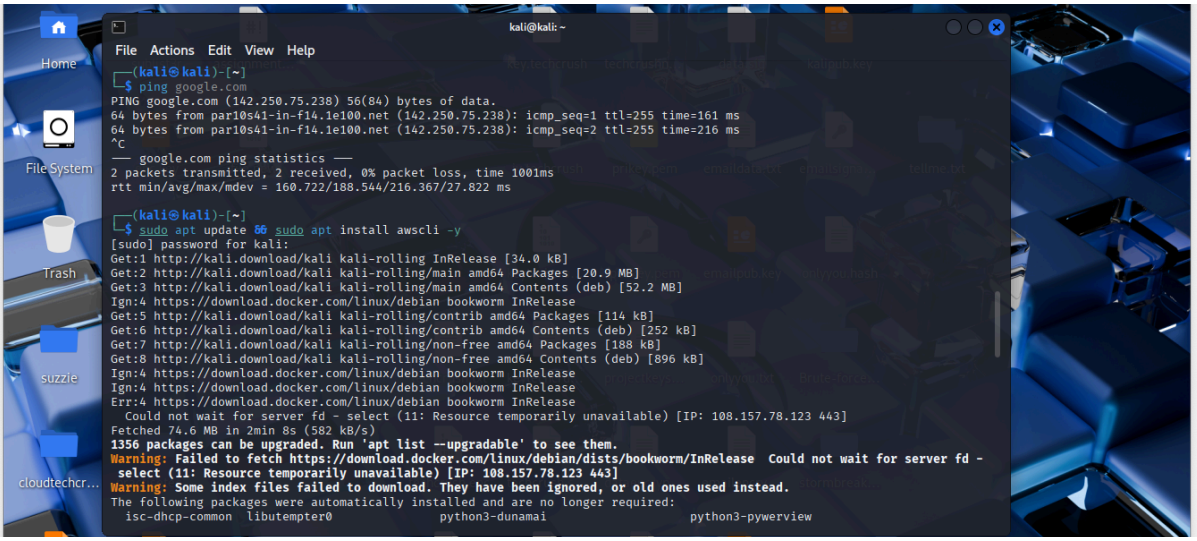
1. Amazon Web Services (2024). AWS Identity and Access Management (IAM) Documentation. Retrieved from: https://docs.aws.amazon.com/iam/
2. Amazon Web Services (2024). AWS Command Line Interface User Guide. Retrieved from: https://docs.aws.amazon.com/cli/latest/userguide/
3. Amazon Web Services (2024). AWS CLI Command Reference. Retrieved from: https://docs.aws.amazon.com/cli/latest/reference/
4. Amazon Web Services (2024). Best Practices for IAM Policies and Permissions. Retrieved from: https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html
5. Amazon Web Services (2024). Managing Access Keys for IAM Users. Retrieved from: https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_access-keys.html
6. Amazon Web Services (2024). Working with AWS VPCs and Subnets. Retrieved from: https://docs.aws.amazon.com/vpc/latest/userguide/
7. Amazon Web Services (2024). AWS Security Best Practices Whitepaper. Retrieved from: https://aws.amazon.com/whitepapers/aws-security-best-practices/
8. HashiCorp (2023). Terraform for AWS Infrastructure as Code. Retrieved from: https://developer.hashicorp.com/terraform/tutorials/aws-get-started
9. TechCrunch Cloud Academy (2024). Capstone Project Guide: IAM Roles and Secure Access Automation. Internal course material provided by TechCrush.
10. Kavis, M. (2023). Architecting the Cloud: Design Decisions for Cloud Computing Service Models (SaaS, PaaS, and IaaS). John Wiley & Sons.
11. Microsoft Learn (2024). Comparison of IAM Between AWS and Azure. Retrieved from: https://learn.microsoft.com/en-us/azure/active-directory/fundamentals/
12. 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
13. CompTIA (2024). Security+ Certification Exam Objectives (SY0-701). Retrieved from: https://www.comptia.org/certifications/security

8.0 Appendices

1. AWS account created



2. awscli installed



3. AWS CLI successfully configured