

**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

1. Introduction
2. Objective
3. Project Tools and Environment
4. Methodology / Step-by-Step Implementation
  - 4.1 Creating VPC
  - 4.2 Creating Subnets (Web and DB)
  - 4.3 Creating IAM Groups (WebAdmins & DBAdmins)
  - 4.4 Creating and Assigning Test Users
  - 4.5 Attaching Policies
  - 4.6 Validating Role Assignments
5. Challenges Faced and Solutions
6. Outcomes / Results
7. Conclusion
8. References
9. Appendices

# 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.

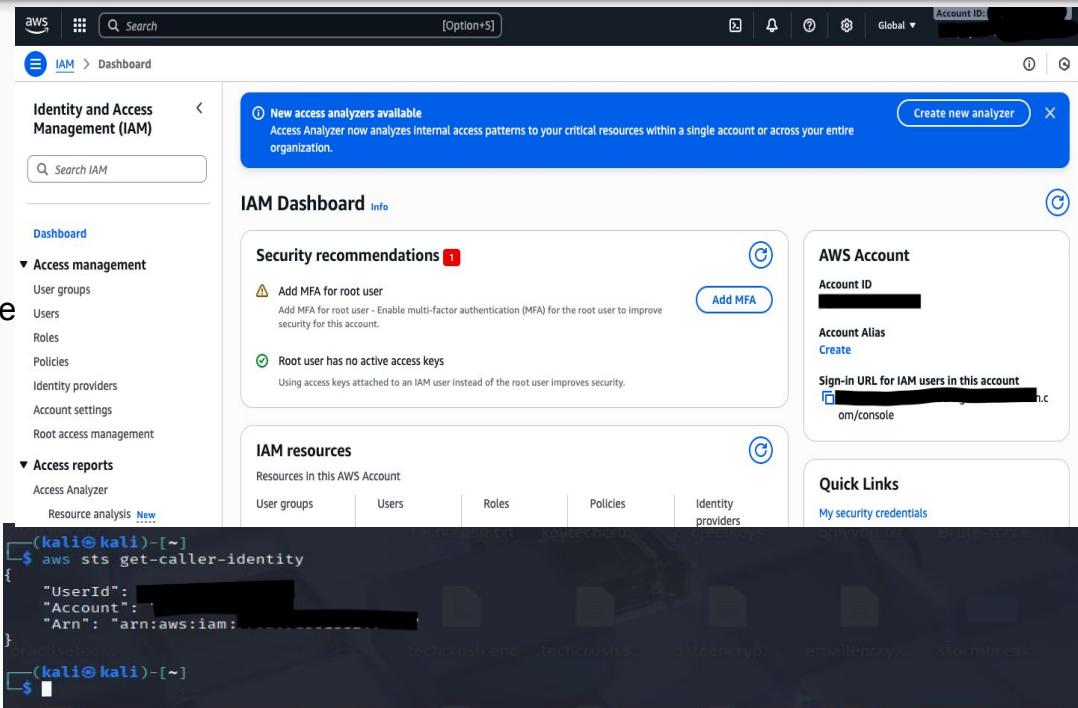
The idea was to simulate a real-world organization where two administrative teams — Database Administrators (DBAdmins) and Web Administrators (WebAdmins) — need different permissions to perform their roles.

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 that 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.

```
[kali㉿kali]:~[ -] ping google.com
PING 172.21.12.75 (172.250.75.238) 56(84) bytes of data:
64 bytes from paro541-in-f14.1e100.net (142.250.75.238): icmp_seq=1 ttl=255 time=161 ms
64 bytes from paro541-in-f14.1e100.net (142.250.75.238): icmp_seq=2 ttl=255 time=216 ms
^C
-- google.com ping statistics --
2 packets transmitted, 2 received, 0% packet loss, time 100ms
rtt min/avg/max/mdev = 160.722/188.544/216.367/27.822 ms

[kali㉿kali]:~[ -] sudo apt update & sudo apt install awscli -y
[sudo] password for kali: 
Get:1 http://Kali/kali kali-kali-rolling InRelease [34.0 kB]
Get:2 http://Kali/kali kali-kali-rolling/main amd64 Packages [20.9 MB]
Get:3 http://Kali/kali kali-kali-rolling/main i386 Packages (debs) [52.2 MB]
Ign:4 https://download.docker.com/linux/debian InRelease
Get:5 https://kali.download/kali kali-rolling/contrib amd64 Packages [114 kB]
Get:6 https://kali.download/kali kali-rolling/contrib i386 Packages [352 kB]
Get:7 https://kali.download/kali kali-rolling/non-free amd64 Packages [188 kB]
Get:8 https://kali.download/kali kali-rolling/non-free i386 Packages [896 kB]
Ign:9 https://download.docker.com/linux/debian InRelease
Err:10 https://download.docker.com/linux/debian bookworm InRelease
Err:11 https://download.docker.com/linux/debian bookworm InRelease
  Could not connect to download.docker.com:80 (10.157.78.123), connection timed out.
  Could not wait for server to respond (11: Resource temporarily unavailable) [IP: 108.157.78.123 443]
Fetched 22.0 MB in 1s (19.9 kB/s)
1356 packages can be upgraded. Run 'apt list --upgradable' to see them.
Warning: Failed to fetch https://download.docker.com/linux/debian/dists/bookworm/InRelease Could not wait for server fd ->
Warning: Failed to fetch https://download.docker.com/linux/debian/dists/bookworm/main/binary-amd64/Packages.gpg Could not wait for server fd ->
Warning: Some index files failed to download. They have been ignored, or old ones used instead.
The following packages were automatically installed and are no longer required:
```

```
Home File Actions Edit View Help

Setting up python3-roman-numerals (3.1.0-2) ...
Processing triggers for man-db (2.13.1-1) ...
Processing triggers for sgml-base (1.31+mu1) ...
Setting up python3-docutils (0.22.2+dfsg-2) ...
Processing triggers for kali-menu (2025.3.0) ...
Setting up awscli (2.23.6-1) ...
Scanning processes...
Scanning linux images...

Running kernel seems to be up-to-date.

No services need to be restarted.

No containers need to be restarted.

No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.

[kali㉿kali]:[~]
$ aws --version
aws-cli/2.23.6 Python/3.13.5 Linux/6.12.33+kali-amd64 source/x86_64.kali.2025

[kali㉿kali]:[~]
$ aws configure
AWS Access Key ID [None]:
AWS Secret Access Key [None]:
Default region name [None]: us-east-1
Default output format [None]: json

[kali㉿kali]:[~]
$ aws sts get-caller-identity
{
    "UserId": "█████████████████████",
    "Account": "█████████████████████",
    "Arn": "arn:aws:iam:█████████████████████"
}

[kali㉿kali]:[~]
```

# 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 resources.

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.

```
kali㉿kali: ~
File Actions Edit View Help
ion-VPC}, {Key=Project, Value=IAM-Automation}]
quote>
└─(kali㉿kali: ~)
$ aws ec2 create-vpc --cidr-block 10.0.0.0/16 --tag-specifications "ResourceType=vpc, Tags=[{Key=Name, Value=IAM-Automation-VPC}, {Key=Project, Value=IAM-Automation}]"
{
    "Vpc": {
        "OwnerId": "752691501352",
        "InstanceTenancy": "default",
        "Ipv6CidrBlockAssociationSet": [],
        "CidrBlockAssociationSet": [
            {
                "AssociationId": "vpc-cidr-assoc-0a3b6b44b29b72ab5",
                "CidrBlock": "10.0.0.0/16",
                "CidrBlockState": {
                    "State": "associated"
                }
            }
        ],
        "IsDefault": false,
        "Tags": [
            {
                "Key": "Project",
                "Value": "IAM-Automation"
            },
            {
                "Key": "Name",
                "Value": "IAM-Automation-VPC"
            }
        ],
        "VpcId": "vpc-0a3b6b44b29b72ab5"
    }
}
File Actions Edit View Help
└─(kali㉿kali: ~)
$ aws ec2 describe-vpcs
{
    "Vpcs": [
        {
            "OwnerId": "752691501352",
            "InstanceTenancy": "default",
            "Ipv6CidrBlockAssociationSet": [],
            "CidrBlockAssociationSet": [
                {
                    "AssociationId": "vpc-cidr-assoc-0a3b6b44b29b72ab5",
                    "CidrBlock": "10.0.0.0/16",
                    "CidrBlockState": {
                        "State": "associated"
                    }
                }
            ],
            "IsDefault": false,
            "Tags": [
                {
                    "Key": "Project",
                    "Value": "IAM-Automation"
                },
                {
                    "Key": "Name",
                    "Value": "IAM-Automation-VPC"
                }
            ],
            "VpcId": "vpc-0a3b6b44b29b72ab5",
            "VpcOptionsId": "dept-0127b6a2cdc27e9f4"
        }
    ]
}
```

# 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**.

```
[kali㉿kali)-[~]
$ aws ec2 create-subnet --vpc-id vpc-12345678901234567890 --cidr-block 10.0.1.0/24 --availability-zone us-east-1a --tag-specifications "ResourceType=subnet", Tags=[{"Key":Name,Value=WebSubnet}, {"Key":Project,Value=IAM-Automation"}]
{
  "Subnet": {
    "AvailabilityZoneId": "use1-az1",
    "MapCustomerOwnedIpOnLaunch": false,
    "OwnerId": "12345678901234567890",
    "AssignIpv6AddressOnCreation": false,
    "Ipv6CidrBlockAssociationSet": [],
    "Tags": [
      {
        "Key": "Project",
        "Value": "IAM-Automation"
      },
      {
        "Key": "Name",
        "Value": "WebSubnet"
      }
    ],
    "SubnetArn": "arn:aws:ec2:us-east-1:bnet/subnet-12345678901234567890"
  },
  "State": "available",
  "VpcId": "vpc-12345678901234567890",
  "CidrBlock": "10.0.1.0/24",
  "AvailableIpAddressCount": 251
}
```

```
[kali㉿kali)-[~]
$ aws ec2 create-subnet --vpc-id vpc-12345678901234567890 --cidr-block 10.0.2.0/24 --availability-zone us-east-1b --tag-specifications "ResourceType=subnet", Tags=[{"Key":Name,Value=DBSubnet}, {"Key":Project,Value=IAM-Automation"}]
{
  "Subnet": {
    "AvailabilityZoneId": "use1-az2",
    "MapCustomerOwnedIpOnLaunch": false,
    "OwnerId": "12345678901234567890",
    "AssignIpv6AddressOnCreation": false,
    "Ipv6CidrBlockAssociationSet": [],
    "Tags": [
      {
        "Key": "Project",
        "Value": "IAM-Automation"
      },
      {
        "Key": "Name",
        "Value": "DBSubnet"
      }
    ],
    "SubnetArn": "arn:aws:ec2:us-east-1:subnet-12345678901234567890"
  },
  "State": "available",
  "VpcId": "vpc-12345678901234567890",
  "CidrBlock": "10.0.2.0/24",
  "AvailableIpAddressCount": 251
}
```



An error occurred (InvalidSubnet.Conflict) when calling the CreateSubnet operation: The CIDR '10.0.2.0/24' conflicts with another subnet.

```
[kali㉿kali)-[~]
$ aws ec2 create-subnet --vpc-id vpc-12345678901234567890 --cidr-block 10.0.2.0/24 --availability-zone us-east-1b --tag-specifications "ResourceType=subnet", Tags=[{"Key":Name,Value=DBSubnet}, {"Key":Project,Value=IAM-Automation"}]
An error occurred (InvalidSubnet.Conflict) when calling the CreateSubnet operation: The CIDR '10.0.2.0/24' conflicts with another subnet.

[kali㉿kali)-[~]
$ aws ec2 describe-subnets --filters "Name=vpc-id,Values=vpc-12345678901234567890", "Name:Tags[?Key='Name'][?Value='DBSubnet']", {"Key":Project,Value=IAM-Automation"} --query "Subnets[*].[SubnetId,Name,CidrBlock,AZ,AvailabilityZone]" --output table
+-----+-----+-----+-----+
| SubnetId | Subnet-12345678901234567890 | WebSubnet | subnet-01 | us-east-1a |
| SubnetId | Subnet-12345678901234567890 | DBSubnet | subnet-02 | us-east-1b |
+-----+-----+-----+-----+
[kali㉿kali)-[~]
$ aws ec2 describe-instances --profile testdb --region us-east-1
{
  "Reservations": []
}
```

# 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.

A screenshot of a terminal window on a Kali Linux system. The terminal shows two commands being run using the AWS CLI to create IAM groups:

```
(kali㉿kali)-[~]
$ aws iam create-group --group-name WebAdmins
{
    "Group": {
        "Path": "/",
        "GroupName": "WebAdmins",
        "GroupId": "12345678-1234-1234-1234-1234567890ab",
        "Arn": "arn:aws:iam::123456789012:group/WebAdmins",
        "CreateDate": "2025-10-30T08:21:05+00:00"
    }
}

(kali㉿kali)-[~]
$ aws iam create-group --group-name DBAdmins
{
    "Group": {
        "Path": "/",
        "GroupName": "DBAdmins",
        "GroupId": "12345678-1234-1234-1234-1234567890ab",
        "Arn": "arn:aws:iam::123456789012:group/DBAdmins",
        "CreateDate": "2025-10-30T08:21:21+00:00"
    }
}
```

After the commands, the terminal shows a file named `data.sig` in the current directory, which is described as an empty document with 0 bytes last modified on 06/09/2025 at 01:42:14 PM.

# 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.

The screenshot shows a terminal window titled "kali@kali: ~" running the "nano" text editor. The file being edited is "DBAdminsReadOnlyPolicy.json". The JSON content defines a policy with a single statement allowing the "ec2:DescribeSubnets", "ec2:DescribeInstances", "ec2:DescribeSecurityGroups", "ec2:GetSubnets", and "ec2:GetVpc\*" actions on all resources. Below this, a command-line session shows the use of the "aws iam create-policy" command with the --policy-document option pointing to the local JSON file. An error message is displayed, indicating a syntax error in the policy document. The user then corrects the policy document by adding a closing brace at the end of the JSON object, and the command runs successfully, returning the policy's ARN. Finally, the "aws iam attach-group-policy" command is used to attach this policy to the "DBAdmins" group.

```
File Actions Edit View Help
GNU nano 8.4
kali@kali: ~
DBAdminsReadOnlyPolicy.json *
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeSubnets",
                "ec2:DescribeInstances",
                "ec2:DescribeSecurityGroups",
                "ec2:GetSubnets",
                "ec2:GetVpc*"
            ],
            "Resource": "*"
        }
    ]
}

File Actions Edit View Help
kali@kali: ~
$ aws iam create-policy --policy-name DBAdminsReadOnlyPolicy --policy-document file:///DBAdminsReadOnlyPolicy.json
An error occurred (MalformedPolicyDocument) when calling the CreatePolicy operation: Syntax errors in policy.
$ nano DBAdminsReadOnlyPolicy.json
File Actions Edit View Help
kali@kali: ~
$ aws iam create-policy --policy-name DBAdminsReadOnlyPolicy --policy-document file:///DBAdminsReadOnlyPolicy.json
{
    "Policy": {
        "PolicyName": "DBAdminsReadOnlyPolicy",
        "PolicyId": "arn:aws:iam::policy/DBAdminsReadOnlyPolicy",
        "Path": "/",
        "DefaultVersionId": "v1",
        "AttachmentCount": 0,
        "PermissionsBoundaryUsageCount": 0,
        "IsAttachable": true,
        "CreateDate": "2025-10-30T09:24:34+00:00",
        "UpdateDate": "2025-10-30T09:24:34+00:00"
    }
}
File Actions Edit View Help
kali@kali: ~
$ aws iam attach-group-policy --group-name DBAdmins --policy-arn arn:aws:iam::policy/DBAdminsReadOnlyPolicy
```

# IAM Policies Creation (WebAdmin)

The image shows a Kali Linux desktop environment with two terminal windows open.

**Left Terminal:**

```
File Actions Edit View Help  
GNU nano 8.4  
WebAdminsReadOnlyPolicy.json *
```

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

**Right Terminal:**

```
kali㉿kali:~$ aws iam create-policy --policy-name WebAdminsReadOnlyPolicy --policy-document file://WebAdminsReadOnlyPolicy.json  
Error parsing parameter '--policy-document': Unable to load paramfile file://WebAdminsReadOnlyPolicy.json: [Errno 2] No such file or directory: 'WebAdminsReadOnlyPolicy.json'  
kali㉿kali:~$ nano WebAdminsReadOnlyPolicy.json  
kali㉿kali:~$ aws iam create-policy --policy-name WebAdminsReadOnlyPolicy --policy-document file://WebAdminsReadOnlyPolicy.json  
{  
  "Policy": {  
    "PolicyName": "WebAdminsReadOnlyPolicy",  
    "PolicyId": "d4a2e0c0-1a2f-11ef-9135-00163da4fbff",  
    "Arn": "arn:aws:iam::123456789012:policy/WebAdminsReadOnlyPolicy",  
    "Path": "/",  
    "DefaultVersionId": "v1",  
    "AttachmentCount": 0,  
    "PermissionsBoundaryUsageCount": 0,  
    "IsAttachable": true,  
    "CreateDate": "2025-10-30T14:39:27+00:00",  
    "UpdateDate": "2025-10-30T14:39:27+00:00"  
  }  
}  
kali㉿kali:~$ aws iam attach-group-policy --group-name WebAdmins --policy-arn arn:aws:iam::123456789012:policy/WebAdminsReadOnlyPolicy  
kali㉿kali:~$ aws ec2 describe-subnets --profile testweb --region us-east-1 --output table
```

# 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.

```
kali@kali:~$ aws iam attach-group-policy --group-name DBAdmins --policy-arn arn:aws:iam::aws:policy/DBAdminsReadOnlyPolicy
(kali㉿kali)-[~]
$ aws iam create-user --user-name TestWebUser
{
  "User": {
    "Path": "/",
    "UserName": "TestWebUser",
    "UserId": "AUGLJGKQH4V5I",
    "Arn": "arn:aws:iam::123456789012:root/testuser/TestWebUser",
    "CreateDate": "2025-10-30T09:39:01+00:00"
  }
}
(kali㉿kali)-[~]
$ aws iam create-user --user-name TestDBUser
{
  "User": {
    "Path": "/",
    "UserName": "TestDBUser",
    "UserId": "AUGLJGKQH4V5I",
    "Arn": "arn:aws:iam::123456789012:root/testuser/TestDBUser",
    "CreateDate": "2025-10-30T09:39:23+00:00"
  }
}
(kali㉿kali)-[~]
$ aws iam add-user-to-group --user-name TestWebUser --group-name WebAdmins
(kali㉿kali)-[~]
$ aws iam add-user-to-group --user-name TestDBUser --group-name DBAdmins
(kali㉿kali)-[~]
$ aws iam create-access-key --user-name TestDBUser
{
  "AccessKey": {
    "UserName": "TestDBUser",
    "AccessKeyId": "AKIA26P7I3EUNOYUQ7KB",
    "Status": "Active",
    "SecretAccessKey": "r0GfpNvfbj0N19Bjaz3vEfj9rZ9UK9Lx8P3Wh2",
    "CreateDate": "2025-10-30T10:02:13+00:00"
  }
}
```

# 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



```
File Actions Edit View Help
(kali㉿kali)-[~]
└─$ aws iam create-access-key --user-name TestDBUser
{
    "AccessKey": {
        "UserName": "TestDBUser",
        "AccessKeyId": "XXXXXXXXXXXXXX",
        "Status": "Active",
        "SecretAccessKey": "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
        "CreateDate": "2025-10-30T10:02:13+00:00"
    }
}

(kali㉿kali)-[~]
└─$ aws configure --profile testdb
AWS Access Key ID [None]:
AWS Secret Access Key [None]: us-east-1
Default region name [None]: ``

(kali㉿kali)-[~]
└─$ aws configure --profile testdb
AWS Access Key ID [None]:
AWS Secret Access Key [None]: ``

(kali㉿kali)-[~]
└─$ aws configure --profile testdb
AWS Access Key ID [None]:
AWS Secret Access Key [None]: ``

(kali㉿kali)-[~]
└─$ aws configure --profile testdb
AWS Access Key ID [None]:
AWS Secret Access Key [None]: ``

(kali㉿kali)-[~]
└─$ aws iam list-attached-group-policies --group-name WebAdmins
{
    "AttachedPolicies": []
}

(kali㉿kali)-[~]
└─$ aws ec2 describe-subnets --profile testweb --region us-east-1 --output table
The config profile (testweb) could not be found

(kali㉿kali)-[~]
└─$ aws iam create-access-key --user-name TestWebUser
{
    "AccessKey": {
        "UserName": "TestWebUser",
        "AccessKeyId": "XXXXXXXXXXXXXX",
        "Status": "Active",
        "SecretAccessKey": "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
        "CreateDate": "2025-10-30T14:16:52+00:00"
    }
}

(kali㉿kali)-[~]
└─$ aws configure --profile testweb
AWS Access Key ID [None]:
AWS Secret Access Key [None]: 3
Default region name [None]: us-east-1
Default output format [None]: json
```

# 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.

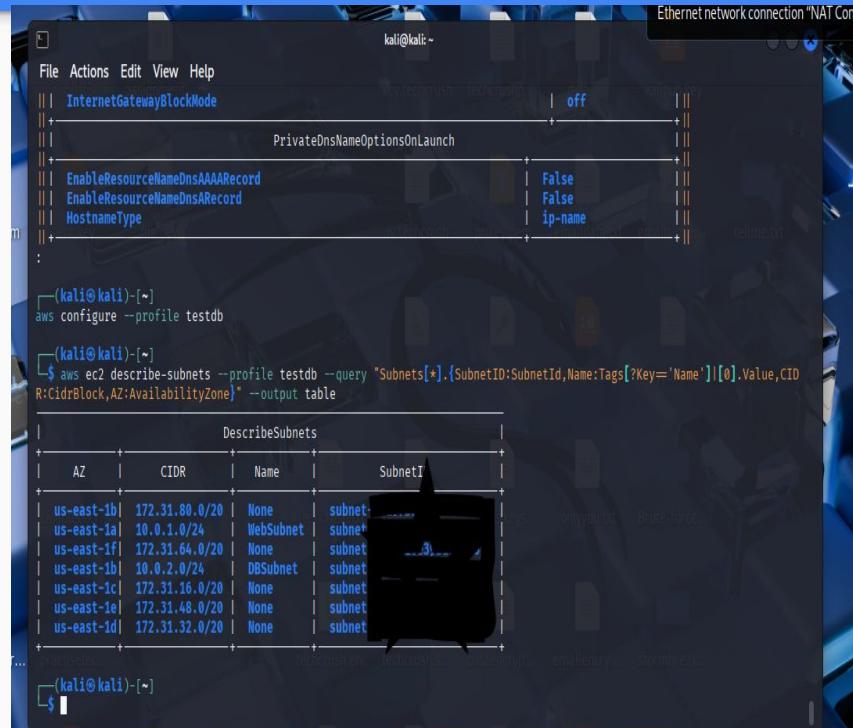
After troubleshooting, every step executed successfully and matched the intended outcome.

# 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 WebAdminsReadOnlyPolicy)
6. WebAdminsReadOnlyPolicy

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



The screenshot shows a terminal window with the following content:

```
kali㉿kali:~$ aws configure --profile testdb
(kali㉿kali:~$)
aws ec2 describe-subnets --profile testdb --query "Subnets[*].[SubnetId,Name:Tags[?Key='Name'][@].Value,CIDR,CidrBlock,AZ:AvailabilityZone]" --output table
(kali㉿kali:~$)

```

DescribeSubnets				
AZ	CIDR	Name	SubnetId	
us-east-1b	172.31.80.0/20	None	subnet-00000000	...
us-east-1a	10.0.1.0/24	Websubnet	subnet-00000001	...
us-east-1f	172.31.64.0/20	None	subnet-00000002	...
us-east-1b	10.0.2.0/24	DBSubnet	subnet-00000003	...
us-east-1c	172.31.16.0/20	None	subnet-00000004	...
us-east-1e	172.31.48.0/20	None	subnet-00000005	...
us-east-1d	172.31.32.0/20	None	subnet-00000006	...

```
(kali㉿kali:~$)
```

# 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.

...

The screenshot shows the AWS IAM Dashboard. On the left, a sidebar lists navigation options: Identity and Access Management (IAM) (selected), Dashboard, Access management, Policies, Identity providers, Account settings, Root access management, and Access reports. The main content area is titled "IAM Dashboard" and includes sections for "Security recommendations" and "IAM resources". The "Security recommendations" section has two items: "Add MFA for root user" and "Deactivate or delete access keys for root user". The "IAM resources" section displays statistics: User groups (2), Users (2), Roles (2), Policies (2), and Identity providers (0). To the right, there's a "AWS Account" summary with fields for Account ID (7526-9150-1352), Account Alias (Create), and Sign-in URL for IAM users in this account (https://7526-9150-1352.signin.aws.amazon.com/console). A "Quick Links" section at the bottom right links to "My security credentials". The bottom of the page includes CloudShell, Feedback, Console Mobile App, and footer links for 2025, Privacy, Terms, and Cookie preferences.

# Conclusion and Recommendations

The image displays four screenshots of the AWS IAM console interface, illustrating the management of user groups and policies.

- Screenshot 1: User groups**  
Shows the "User groups" page with two defined groups: DBAdmins and WebAdmins, each with one user assigned and created 4 days ago.
- Screenshot 2: Policy details**  
Shows the "Policy details" page for the "WebAdminsReadOnlyPolicy". It is a customer-managed policy created on October 30, 2025. The "Permissions" tab shows it allows EC2 limited access to List, Read operations on all resources.
- Screenshot 3: Policy details**  
Shows the "Policy details" page for the "DBAdminsReadOnlyPolicy". It is a customer-managed policy created on October 30, 2025. The "Permissions" tab shows it allows EC2 limited access to List, Read operations on all resources.
- Screenshot 4: Policies**  
Shows the "Policies" page listing 1401 policies. Two specific policies are highlighted: "DBAdminsReadOnlyPolicy" and "WebAdminsReadOnlyPolicy", both categorized as "Customer managed".

# Conclusion and Recommendations

The screenshot shows the AWS IAM Roles page. At the top, there is a search bar and a 'Create role' button. Below the search bar, a table lists two roles:

Role name	Trusted entities	Last activity
AWSServiceRoleForSupport	AWS Service: support (Service-Linked)	10 days ago
AWSServiceRoleForTrustedAdvisor	AWS Service: trustedadvisor (Service-Linked)	-

Below the table, there are three sections: 'Roles Anywhere', 'Access AWS from your non AWS workloads', and 'X.509 Standard'. Each section has a brief description and a 'Manage' button.

**Roles Anywhere**: Authenticate your non AWS workloads and securely provide access to AWS services.

**Access AWS from your non AWS workloads**: Operate your non AWS workloads using the same authentication and authorization strategy that you use within AWS.

**X.509 Standard**: Use your own existing PKI infrastructure or use AWS Certificate Manager Private Certificate Authority to authenticate identities.

**Temporary credentials**: Use temporary credentials with ease and benefit from the enhanced security they provide.

At the bottom, there is a footer with links to CloudShell, Feedback, and Console Mobile App, along with copyright information for 2025 Amazon Web Services, Inc. or its affiliates.

The screenshots show the AWS IAM User Groups pages for 'WebAdmins' and 'DBAdmins'.

**WebAdmins** (Summary):  
User group name: WebAdmins  
Creation time: October 50, 2025, 09:21 (UTC+01:00)  
ARN: arn:aws:iam::752691501352:group/WebAdmins  
Users (1): TestWebUser

**DBAdmins** (Summary):  
User group name: DBAdmins  
Creation time: October 50, 2025, 09:21 (UTC+01:00)  
ARN: arn:aws:iam::752691501352:group/DBAdmins  
Users (1): TestDBUser

Both pages have tabs for 'Users', 'Permissions', and 'Access Advisor'. The 'Users' tab shows a table with one user entry, and the 'Permissions' tab shows the ARN of the user.

# 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>