

AWS IAM Management Shell Script Project

Author: Oluwaseun Osunsola

Environment: AWS, CLI, Windows

Project Link: <https://github.com/Oluwaseunoa/DevOps-Projects/tree/main/Cloud%20Computing/Shell%20Script%20for%20AWS%20IAM%20Management>

Overview

CloudOps Solutions, a growing company leveraging AWS for its cloud infrastructure, requires automation to streamline AWS Identity and Access Management (IAM) tasks for its expanding DevOps team. This project delivers a Bash shell script, `aws-iam-manager.sh`, to automate the creation of IAM users, an admin group, policy attachment, and user assignments. The script addresses the following objectives:

1. Define an array to store IAM user names for iteration.
2. Create IAM users using AWS CLI commands.
3. Create an "admin" group using AWS CLI.
4. Attach the `AdministratorAccess` policy to the "admin" group.
5. Assign the users to the "admin" group.

Note: The project objectives specify five IAM users, but the provided script and screenshots use three users (`alice`, `bob`, `charlie`), likely for a test run. The script can be updated to include five users to fully meet the requirements (see Recommendations).

This `README.md` provides comprehensive documentation, detailing the setup, script development, execution, explanation, conclusions, and recommendations. Screenshots (43 in total) are included in the `./img/` directory to illustrate each step visually.

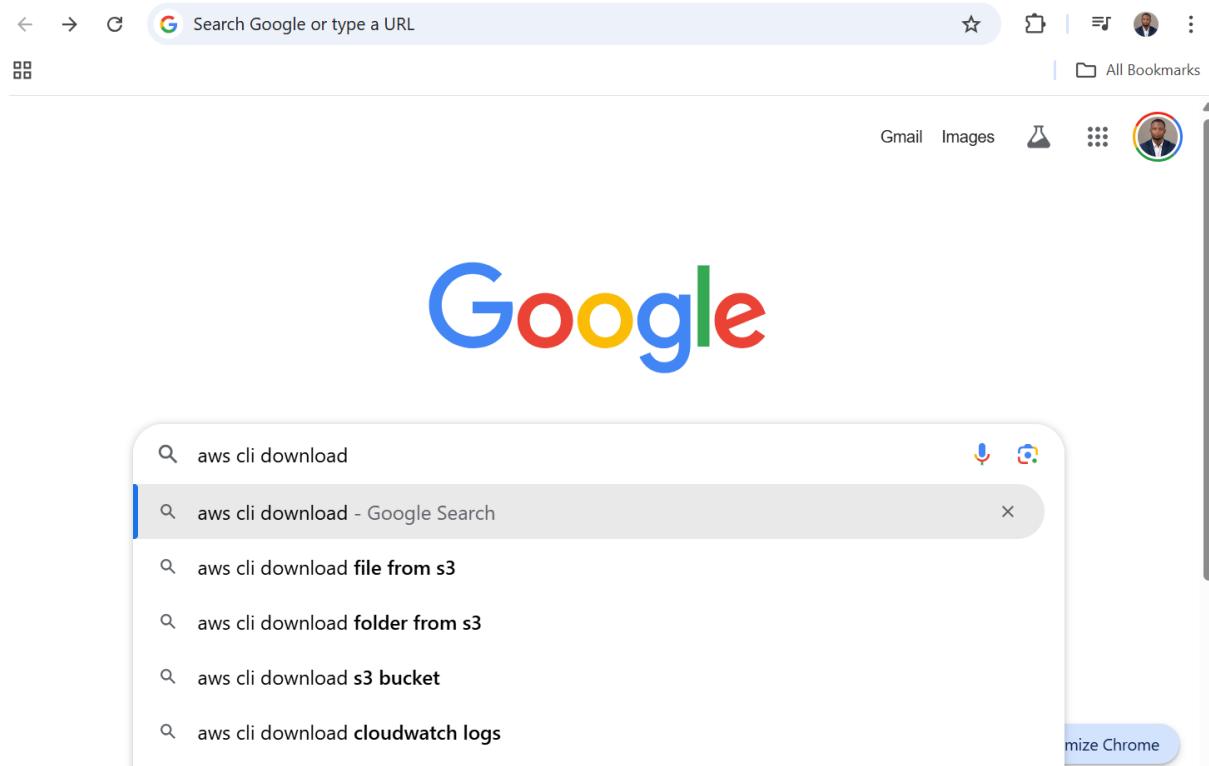
1. Installing and Configuring AWS CLI

The AWS Command Line Interface (CLI) must be installed and configured with appropriate IAM permissions to enable the script to manage IAM resources. The following steps outline the setup process, with screenshots for each action.

Steps

1. Search for AWS CLI Download:

- Searched for the AWS CLI download page to locate the installer for the Windows environment.



2. Access the Download Page:

- Clicked the first link from the search results to visit the official AWS CLI download page.

A screenshot of a web browser showing the official AWS CLI download page. The address bar shows the URL 'google.com/search?q=aws+cli+download&oq=aws+cli+download&gs_lcrp=EgZjaHJvbW...'. The search term 'aws cli download' is typed into the search bar. The results page features a prominent link from 'Amazon AWS Documentation' to the 'Install or update the latest version of the AWS CLI' section. This section contains instructions for installing the AWS CLI on Windows, Linux, macOS, and past releases. A red box highlights the 'Windows' section, which includes a link to the Windows MSI installer.

3. Download Windows MSI Installer:

- Scrolled down and selected the Windows MSI download link for AWS CLI v2.

AWS Command Line Interface

Windows

Install and update requirements

- We support the AWS CLI on Microsoft-supported versions of 64-bit Windows.
- Admin rights to install software

Install or update the AWS CLI

To update your current installation of AWS CLI on Windows, download a new installer each time you update to overwrite previous versions. AWS CLI is updated regularly. To see when the latest version was released, see the [AWS CLI version 2 Changelog](#) on GitHub.

- Download and run the AWS CLI MSI installer for Windows (64-bit):
<https://awscli.amazonaws.com/AWSCLIV2.msi>

Alternatively, you can run the `msiexec` command to run the MSI installer.

```
C:\> msiexec.exe /i https://awscli.amazonaws.com/AWSCLIV2.msi
```

4. Verify and Run Installer:

- Verified the downloaded MSI file and ran it to start the installation.

AWS Command Line Interface

Windows

Install and update requirements

- We support the AWS CLI on Microsoft-supported versions of 64-bit Windows.
- Admin rights to install software

Install or update the AWS CLI

To update your current installation of AWS CLI on Windows, download a new installer each time you update to overwrite previous versions. AWS CLI is updated regularly. To see when the latest version was released, see the [AWS CLI version 2 Changelog](#) on GitHub.

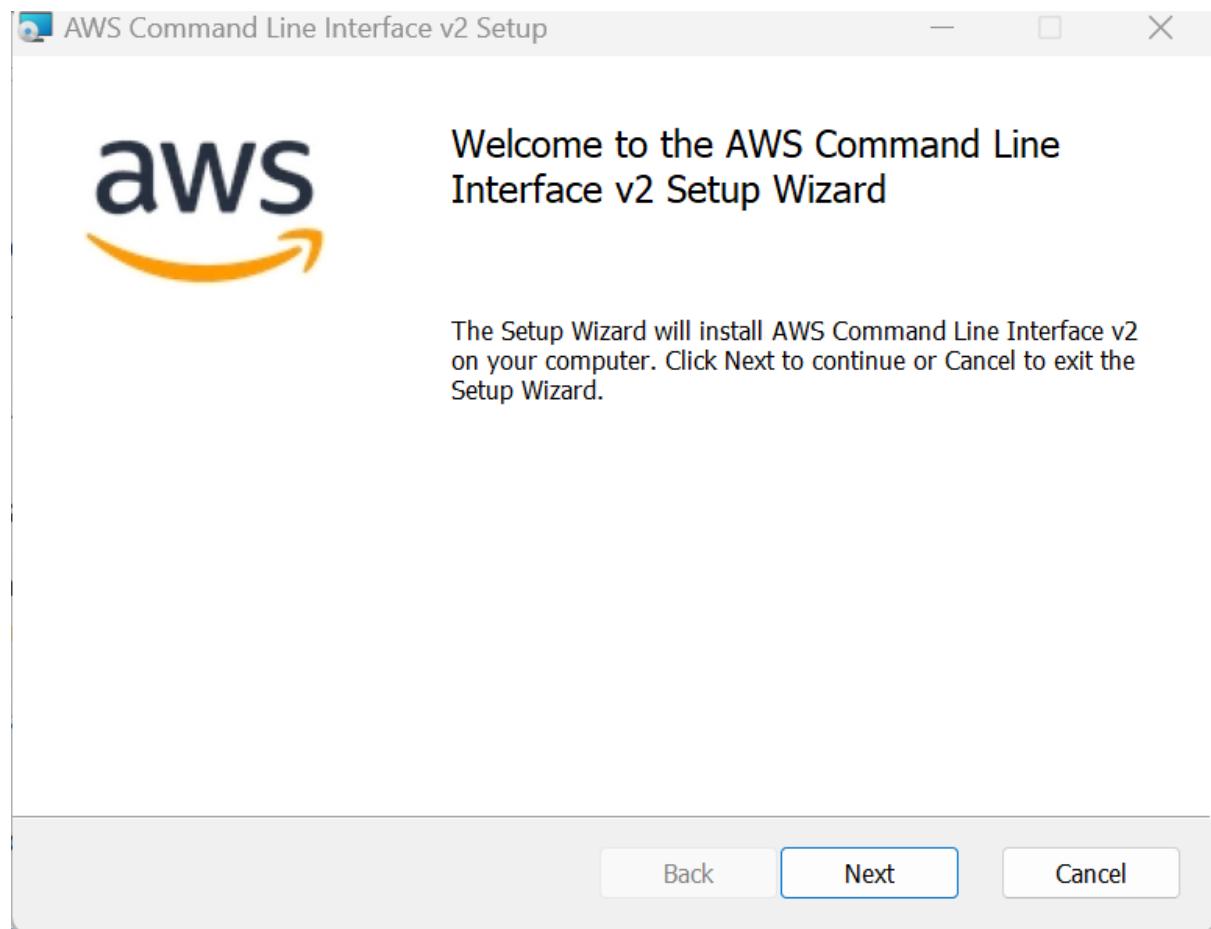
- Download and run the AWS CLI MSI installer for Windows (64-bit):
<https://awscli.amazonaws.com/AWSCLIV2.msi>

Alternatively, you can run the `msiexec` command to run the MSI installer.

```
C:\> msiexec.exe /i https://awscli.amazonaws.com/AWSCLIV2.msi
```

5. Complete Installation:

- Followed the installation prompts to install AWS CLI on the system.



6. Verify AWS CLI Installation:

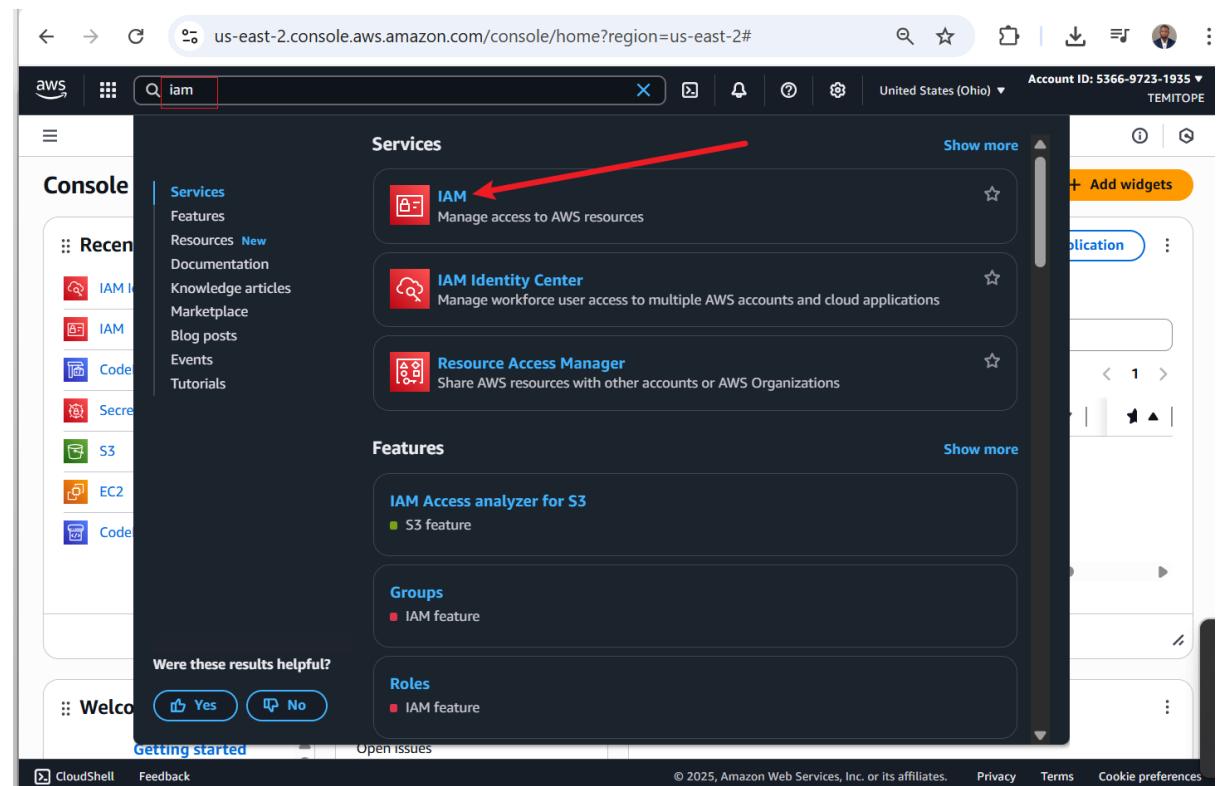
- Opened a terminal (Git Bash on Windows) and ran `aws --version` to confirm successful installation.

A screenshot of a terminal window titled "HP DESKTOP-19M74R1 MINGW64 ~". It displays the command `aws --version` and its output: "aws-cli/2.27.2 Python/3.13.2 Windows/11 exe/AMD64".

```
HP DESKTOP-19M74R1 MINGW64 ~
$ aws --version
aws-cli/2.27.2 Python/3.13.2 Windows/11 exe/AMD64
$ [REDACTED]
```

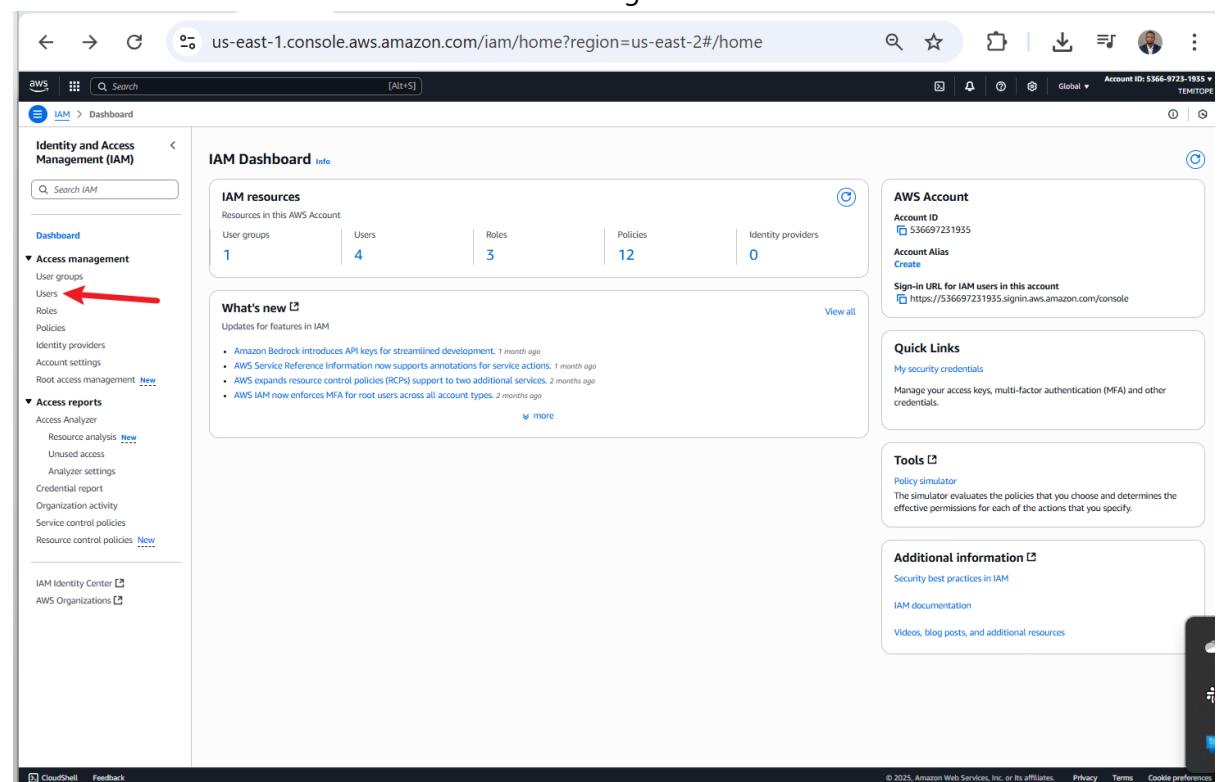
7. Access AWS IAM Console:

- Searched for "IAM" in the AWS Management Console and navigated to the IAM dashboard.



8. Navigate to Users:

- Clicked on "Users" in the IAM dashboard to manage user creation.



9. Initiate User Creation:

- Clicked "Create user" to set up a new IAM user for AWS CLI access.

The screenshot shows the AWS IAM 'Users' page. On the left, there's a navigation sidebar with sections like 'Identity and Access Management (IAM)', 'Access management', 'Access reports', and 'AWS Organizations'. The main area displays a table titled 'Users (4) info' with columns for User name, Path, Group, Last activity, MFA, Password age, Console last sign-in, Access key ID, Active key age, and Access key. The users listed are Ade, Jack, jiro, and seun. The 'Create user' button is highlighted with a red arrow.

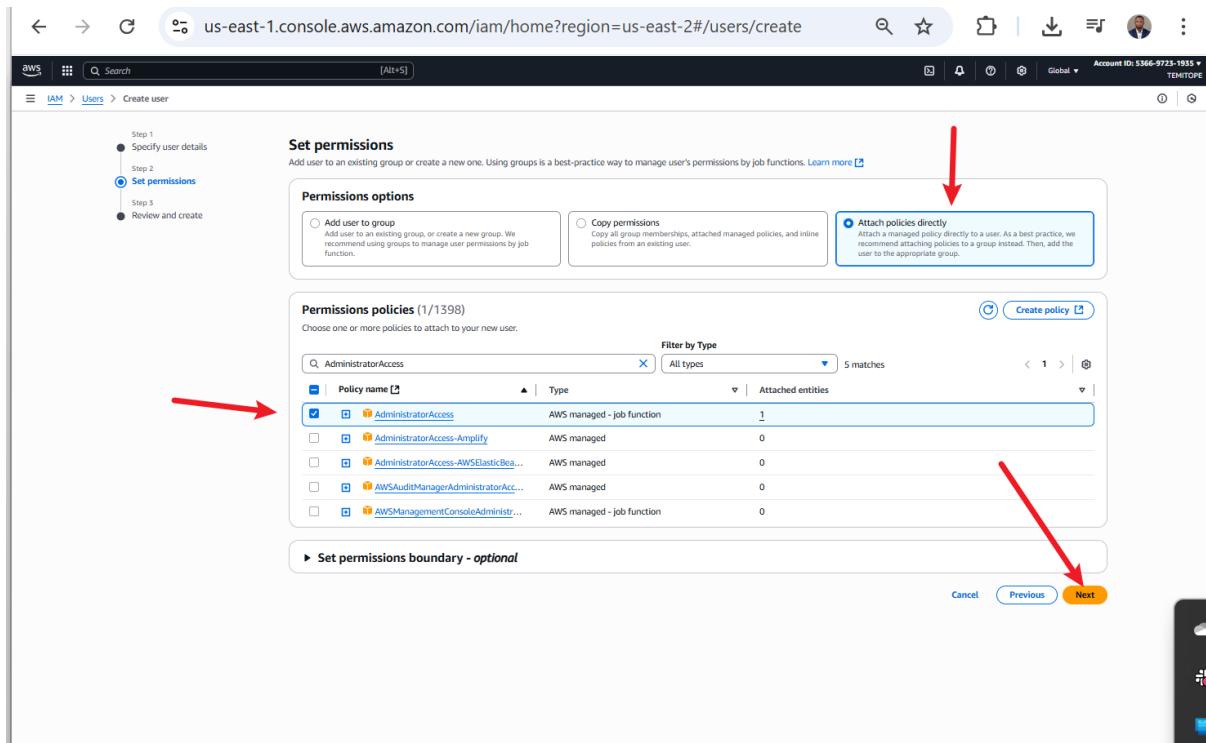
10. Name User and Skip Console Access:

- Named the user (e.g., **aws-cli-admin**) and proceeded without enabling console access, as the user is for CLI use.

The screenshot shows the 'Specify user details' step of the 'Create user' wizard. On the left, a sidebar shows 'Step 1: Specify user details' (selected), 'Step 2: Set permissions', and 'Step 3: Review and create'. The main area has a 'User details' section with a 'User name' input field containing 'oluwaseun'. Below it, there's a note about character restrictions and an optional checkbox for 'Provide user access to the AWS Management Console'. A callout box provides instructions for generating programmatic access keys. The 'Next' button is highlighted with a red arrow.

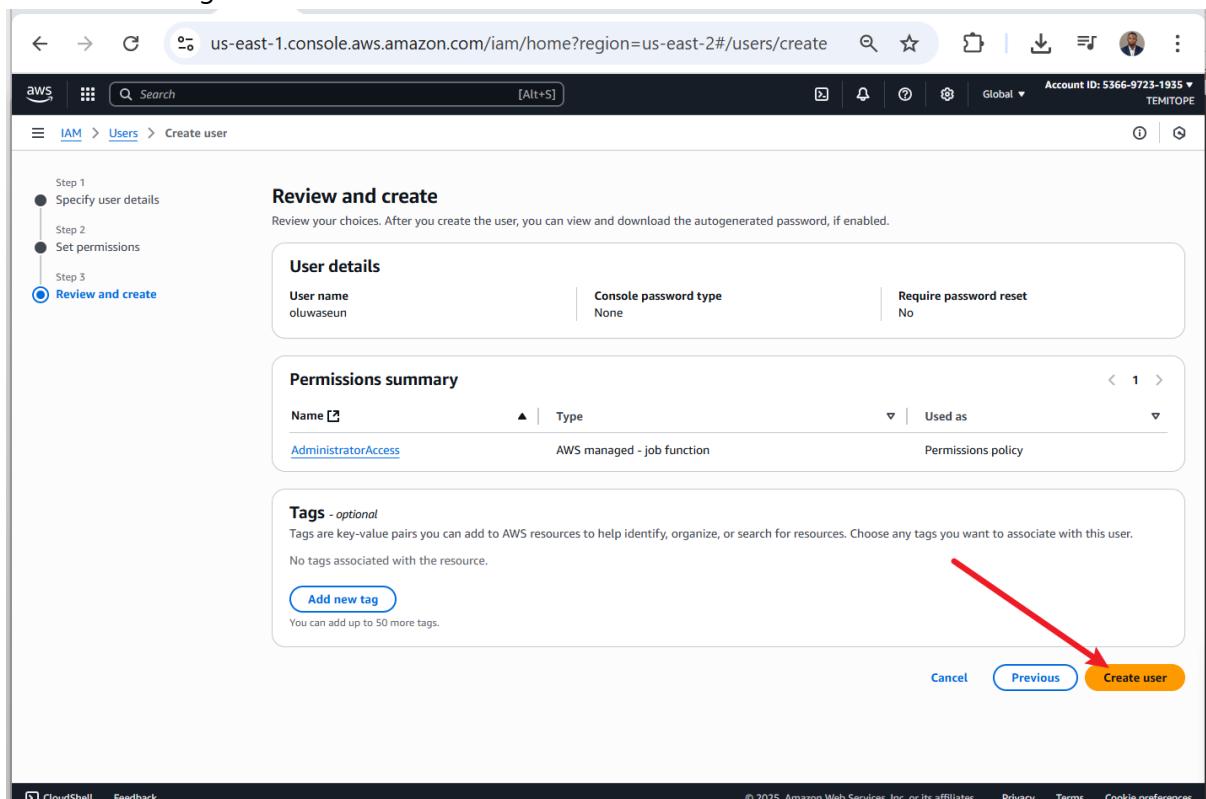
11. Attach AdministratorAccess Policy:

- Attached the **AdministratorAccess** policy directly to the user to grant IAM management permissions.



12. Create the User:

- Reviewed settings and clicked "Create user" to finalize the user creation.



13. Access User Details:

- Navigated to the user list and clicked on the newly created user (`aws-cli-admin`).

The screenshot shows the AWS IAM 'Users' page. On the left, there's a navigation sidebar with 'Identity and Access Management (IAM)' selected. The main area displays a table of users with columns for User name, Path, Group, Last activity, MFA, Password age, and Console last sig. Five users are listed: Ade, Jack, iiro, oluwaseun, and seun. The user 'oluwaseun' is highlighted with a red arrow.

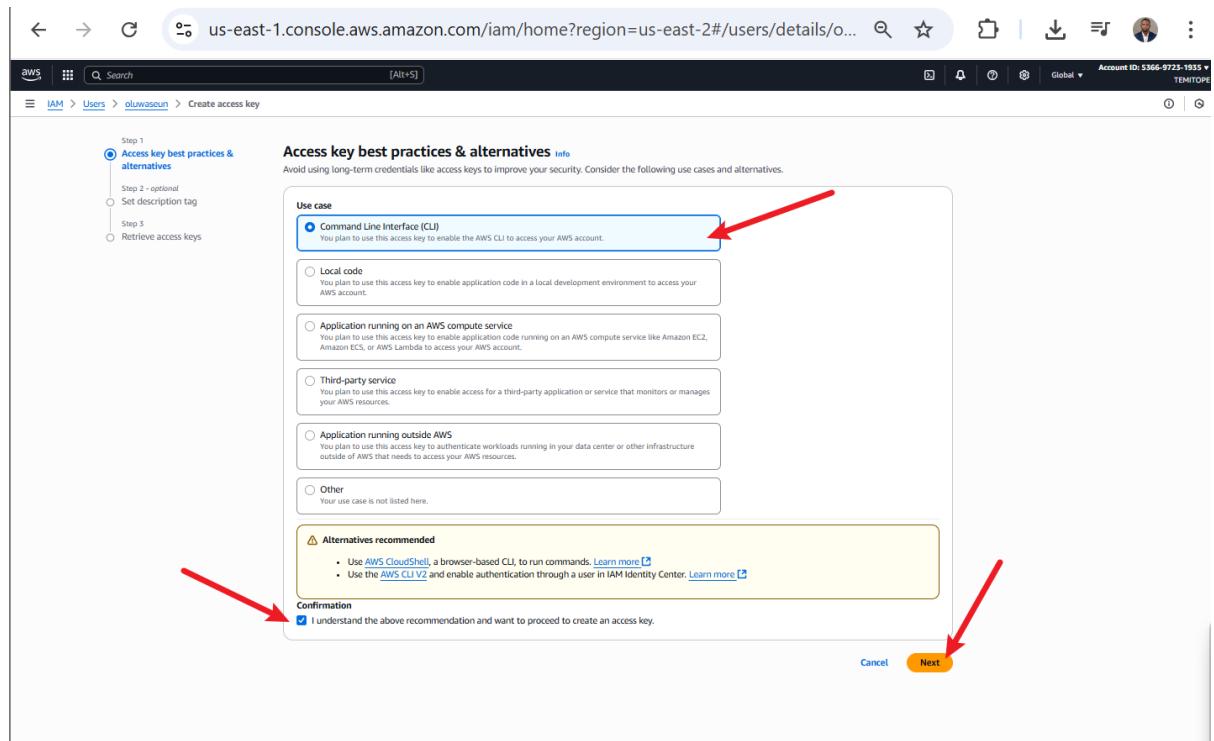
14. Create Access Key:

- Under "Security credentials," clicked "Create access key" to generate CLI credentials.

The screenshot shows the AWS IAM 'User Details' page for the user 'oluwaseun'. The left sidebar shows the user's ARN: arn:aws:iam:536697231935:user/oluwaseun. The main area has tabs for Summary, Permissions, Groups, Tags, Security credentials, and Last Accessed. The 'Security credentials' tab is active. It shows 'Console sign-in' with a sign-in link and 'Multi-factor authentication (MFA) (0)' with an 'Assign MFA device' button. Under 'Access keys (0)', there's a note about using access keys for programmatic calls. A red arrow points to the 'Create access key' button. At the bottom, there's a section for 'API keys for Amazon Bedrock (0)'.

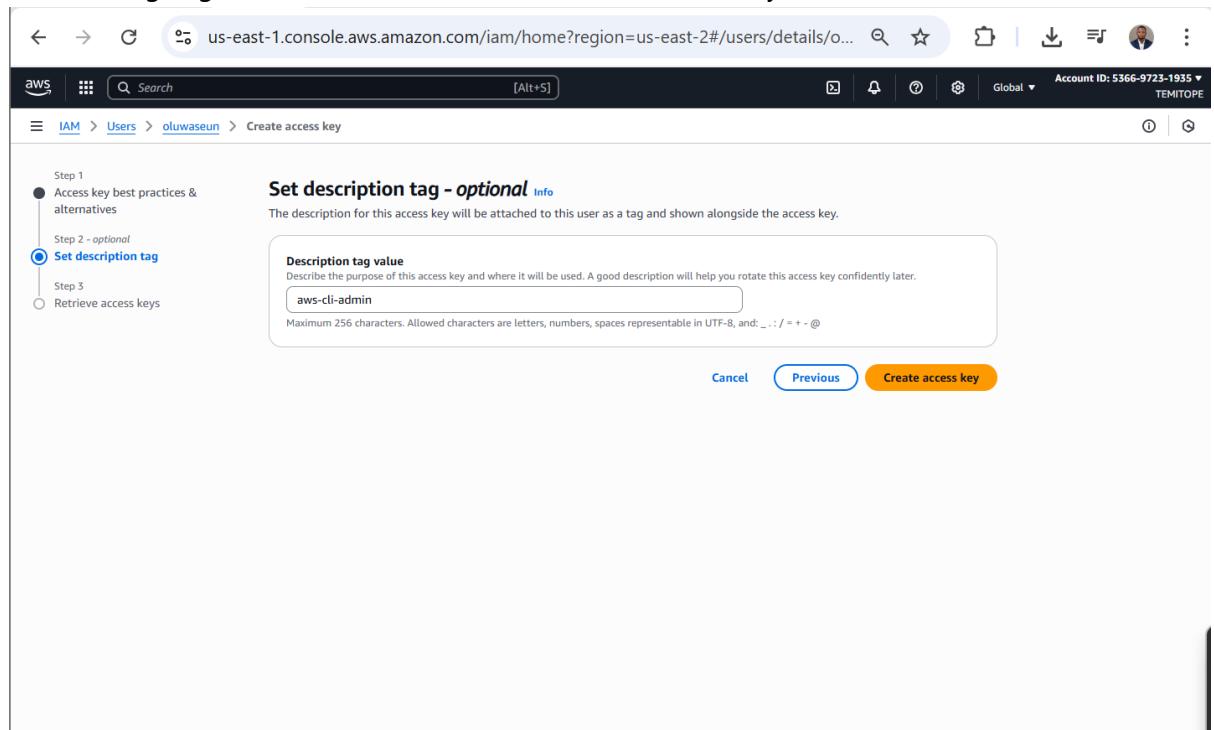
15. Select CLI Usage:

- Selected "Command Line Interface (CLI)" and checked the recommendation box to proceed.



16. Tag User and Create Key:

- Added a tag (e.g., **aws-cli-admin**) and created the access key.



17. Secure Access Key:

- Noted the Access Key ID and Secret Access Key, downloading them as a CSV for secure storage.

Access key created
This is the only time that the secret access key can be viewed or downloaded. You cannot recover it later. However, you can create a new access key any time.

Step 1
Access key best practices & alternatives

Step 2 - optional
Set description tag

Step 3
Retrieve access keys

Access key
If you lose or forget your secret access key, you cannot retrieve it. Instead, create a new access key and make the old key inactive.

Access key	Secret access key
AKIAZ5NF3I74GXNZHPF	0XKxQPYflySIS+qjy

Access key best practices

- Never store your access key in plain text, in a code repository, or in code.
- Disable or delete access key when no longer needed.
- Enable least-privilege permissions.
- Rotate access keys regularly.

For more details about managing access keys, see the [best practices for managing AWS access keys](#).

Download .csv file **Done**

18. Confirm Access Key:

- Verified that the user now has an access key for CLI operations.

Identity and Access Management (IAM)

Summary

ARN: arn:aws:iam::536697231935:user/oluwaseun
Console access: Disabled
Created: August 22, 2025, 04:37 (UTC+01:00)
Last console sign-in: -

Access credentials

Access key 1: AKIAZ5NF3I74GXNZHPF - Active
Never used. Created today.
Access key 2: Create access key

Permissions **Groups** **Tags (1)** **Security credentials** **Last Accessed**

Console sign-in

Console sign-in link: https://536697231935.signin.aws.amazon.com/console
Console password: Not enabled
Enable console access

Multi-factor authentication (MFA) (0)

No MFA devices. Assign an MFA device to improve the security of your AWS environment
Assign MFA device

Access keys (1)

Create access key

19. Configure AWS CLI:

- Ran `aws configure` in the terminal, supplying the Access Key ID, Secret Access Key, and leaving other prompts (region, output format) at default by pressing Enter.

```
HP@DESKTOP-I9M74R1 MINGW64 ~
$ aws configure
AWS Access Key ID [*****4XSU]: AKIAZ5NF3I74GXNZHPF
AWS Secret Access Key [*****Gxtu]: 0XKxQPYfIySlsI[REDACTED]
Default region name [us-east-1]:
Default output format [json]:
```

o

20. Test CLI Access:

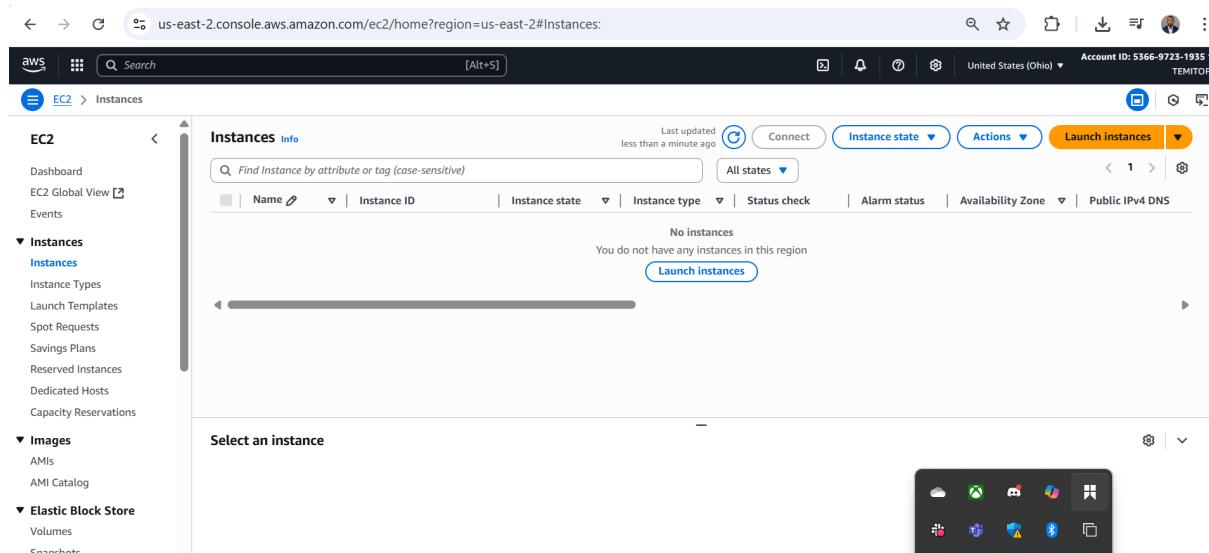
- Ran `aws ec2 describe-instances` to verify the CLI user's permissions, confirming access to AWS services.

```
HP@DESKTOP-I9M74R1 MINGW64 ~
$ aws ec2 describe-instances
{
    "Reservations": []
}
```

o

21. Confirm No EC2 Instances:

- Verified the output, confirming no EC2 instances exist (as expected in a test account).

**22. Verify IAM User Count:**

- Navigate to iam user list page to verify that there is only one user so far. Yours may be more, no problem.

The screenshot shows the AWS IAM service interface. On the left, there's a navigation sidebar with options like 'Identity and Access Management (IAM)', 'Dashboard', 'User groups', 'Roles', 'Policies', 'Identity providers', 'Account settings', 'Root access management', 'Access reports', 'Access Analyzer', 'Unused access', 'Analyzer settings', 'Credential report', 'Organization activity', 'Service control policies', and 'Resource control policies'. The main area is titled 'Users (1) info' and contains a table with one row. The table columns are 'User name', 'Path', 'Group', 'Last activity', 'MFA', 'Password age', 'Console last sign-in', 'Access key ID', 'Active key age', and 'Access key status'. The user listed is 'oluwasun' with a path of '/', group count of 0, last activity 9 minutes ago, and active access key. At the top right, there are 'Delete' and 'Create user' buttons. Below the table is a toolbar with various icons.

Prerequisites

- **AWS CLI:** Installed and configured with an IAM user having permissions for `iam:CreateUser`, `iam:CreateGroup`, `iam:AttachGroupPolicy`, and `iam:AddUserToGroup`.
- **Terminal:** Git Bash (Windows) or native terminal (Linux/macOS).
- **Permissions:** The AWS account must have IAM management capabilities.
- **Linux/Shell Scripting Knowledge:** Familiarity with Bash scripting, as per the Linux foundations prerequisite.

2. Script Creation (aws-iam-manager.sh)

The `aws-iam-manager.sh` script was developed to automate IAM resource management for CloudOps Solutions. It defines an array of three IAM user names (`alice`, `bob`, `charlie`) as shown in the provided script, creates the users, sets up an "admin" group with the `AdministratorAccess` policy, and assigns the users to the group. **Note:** The project requires five users, but the script and screenshots reflect a test run with three users. The script can be updated to include five users (see Recommendations).

Steps

23. Create Project Directory:

- Created a project folder and navigated into it:

```
mkdir "Shell Script for AWS IAM Management"
cd "Shell Script for AWS IAM Management"
```

```
MINGW64:/c/Users/HP/Documents/Workspace/DevOps-Projects/Cloud Computing (main)
$ mkdir "Shell Script for AWS IAM Management"
```

24. Create Script File:

- Created `aws-iam-manager.sh` using `vim` and entered insert mode to write the script.

```
vim aws-iam-manager.sh
```

```
MINGW64:/c/Users/HP/Documents/Workspace/DevOps-Projects/Cloud Computing/Shell Script for AWS IAM Management
$ vim aws-iam-manager.sh
```

25. Add Shebang:

- Added the shebang (`#!/bin/bash`) to ensure the script runs in Bash.

```
#!/bin/bash
```

26. Document Script Purpose:

- Added comments to describe the script's purpose for clarity.

```
MINGW64:/c/Users/HP/Documents/Workspace/DevOps-Projects/Cloud Computing/Shell Script for AWS IAM Management

#!/bin/bash

# AWS IAM Manager Script for CloudOps Solutions
# # This script automates the creation of IAM users, groups, and permissions
```

27. Define User Array:

- Defined an array with three user names (`alice`, `bob`, `charlie`) for iteration, as shown in the provided script.

```
#!/bin/bash

# AWS IAM Manager Script for CloudOps Solutions
# # This script automates the creation of IAM users, groups, and permissions

IAM_USER_NAMES=( "alice" "bob" "charlie") ←
```

28. Implement User Creation Function

- Wrote the `create_iam_users` function to iterate through the array and create users.

```
#!/bin/bash

# AWS IAM Manager Script for CloudOps Solutions
# This script automates the creation of IAM users, groups, and permissions

IAM_USER_NAMES=("alice" "bob" "charlie")

# Function to create IAM users

create_iam_users(){
    echo "Starting IAM user creation process..."
    echo "-----"
    for user in "${IAM_USER_NAMES[@]}"; do
        # Check if user already exists
        if aws iam get-user --user-name "$user" >/dev/null 2>&1; then
            echo "User $user already exists. Skipping..."
        else
            aws iam create-user --user-name "$user"
            if [ $? -eq 0 ]; then
                echo "Created IAM user: $user"
            else
                echo "Error creating IAM user: $user"
            fi
        fi
    done
    echo "-----"
    echo "IAM user creation process completed"
    echo ""
}

}
```

29. Implement Group Creation and Policy Attachment:

- Wrote the `create_admin_group` function to create the "admin" group and attach the `AdministratorAccess` policy.

```

        echo ""
    }

# Function to create admin group and attach policy
create_admin_group() {
    echo "Creating admin group and attaching policy..."
    echo "-----"

    # Check if group already exists
    if aws iam get-group --group-name "admin" >/dev/null 2>&1; then
        echo "@group 'admin' already exists. Skipping creation..."
    else
        aws iam create-group --group-name admin
        if [ $? -eq 0 ]; then
            echo "Created group: admin"
        else
            echo "Error: Failed to create group 'admin'"
        fi
    fi

    # Attach AdministratorAccess policy
    echo "Attaching AdministratorAccess policy..."
    aws iam attach-group-policy --group-name admin \
        --policy-arm arn:aws:iam::aws:policy/AdministratorAccess

    if [ $? -eq 0 ]; then
        echo "Success: AdministratorAccess policy attached"
    else
        echo "Error: Failed to attach AdministratorAccess policy"
    fi

    echo "-----"
    echo ""
}

```

30. Implement User Assignment Function:

- Wrote the `add_users_to_admin_group` function to assign users to the "admin" group.

```

        else
            echo "Error: Failed to attach AdministratorAccess policy"
        fi

    echo "-----"
    echo ""
}

# Function to add users to admin group
add_users_to_admin_group() {
    echo "Adding users to admin group..."
    echo "-----"

    for user in "${!IAM_USER_NAMES[@]}"; do
        aws iam add-user-to-group --user-name "$user" --group-name admin
        if [ $? -eq 0 ]; then
            echo "Added $user to admin group"
        else
            echo "Error: Failed to add $user to admin group"
        fi
    done

    echo "-----"
    echo "User group assignment process completed."
    echo ""
}

```

31. Add Main Execution Function:

- Implemented the `main` function to orchestrate tasks and check for AWS CLI.

```

        else
            echo "Error: Failed to add $user to admin group"
        fi
    done

    echo "-----"
    echo "User group assignment process completed."
    echo ""
}

# Main execution function
main() {
    echo "-----"
    echo " AWS IAM Management Script"
    echo "-----"
    echo ""

    # Verify AWS CLI is installed and configured
    if ! command -v aws &> /dev/null; then
        echo "Error: AWS CLI is not installed. Please install and configure it first."
        exit 1
    fi

    # Execute the functions
    create_iam_users
    create_admin_group
    add_users_to_admin_group

    echo "-----"
    echo " AWS IAM Management Completed"
    echo "-----"
}

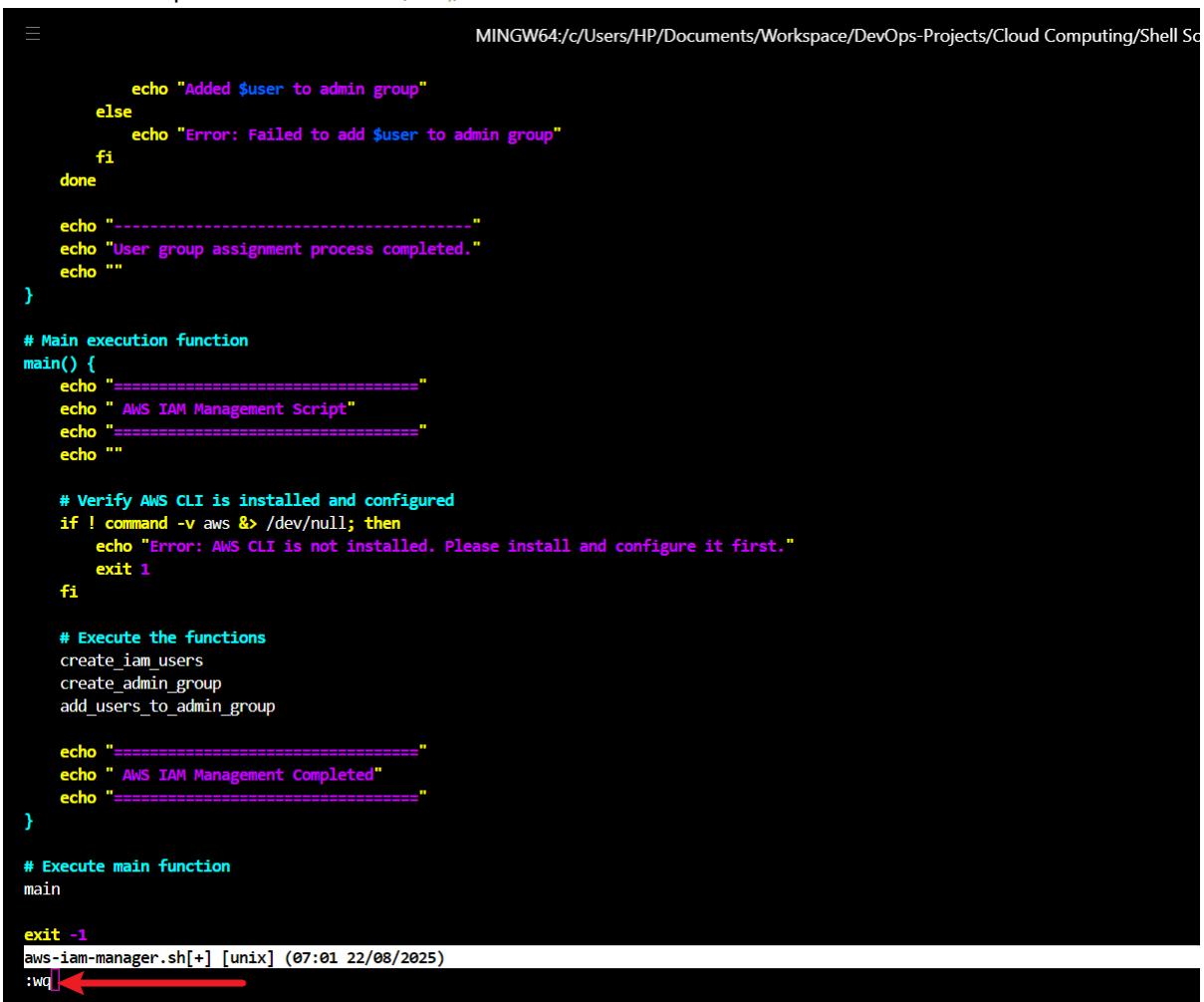
# Execute main function
main

exit 0
aws-iam-manager.sh [unix] (07:01 22/08/2025)

```

32. Save and Exit Vim:

- Saved the script and exited Vim (:wq).



```

echo "Added $user to admin group"
else
    echo "Error: Failed to add $user to admin group"
fi
done

echo "-----"
echo "User group assignment process completed."
echo ""
}

# Main execution function
main() {
    echo =====
    echo " AWS IAM Management Script"
    echo =====
    echo ""

    # Verify AWS CLI is installed and configured
    if ! command -v aws &> /dev/null; then
        echo "Error: AWS CLI is not installed. Please install and configure it first."
        exit 1
    fi

    # Execute the functions
    create_iam_users
    create_admin_group
    add_users_to_admin_group

    echo =====
    echo " AWS IAM Management Completed"
    echo =====
}

# Execute main function
main

exit 1
aws-iam-manager.sh[+] [unix] (07:01 22/08/2025)
:wq

```

Script

```

#!/bin/bash

# AWS IAM Manager Script for CloudOps Solutions
# This script automates the creation of IAM users, groups, and permissions

# Define IAM User Names Array (EDIT THIS LIST)
IAM_USER_NAMES=("alice" "bob" "charlie")

# Function to create IAM users
create_iam_users() {
    echo "Starting IAM user creation process..."
    echo "-----"

    for user in "${IAM_USER_NAMES[@]}"; do
        # Check if user already exists
        if aws iam get-user --user-name "$user" >/dev/null 2>&1; then
            echo "User $user already exists. Skipping..."
        else
            aws iam create-user --user-name "$user"
            if [ $? -eq 0 ]; then
                echo "Created IAM user: $user"
            else

```

```
        echo "Error creating IAM user: $user"
    fi
fi
done

echo -----
echo "IAM user creation process completed."
echo ""

}

# Function to create admin group and attach policy
create_admin_group() {
    echo "Creating admin group and attaching policy..."
    echo -----

    # Check if group already exists
    if aws iam get-group --group-name "admin" >/dev/null 2>&1; then
        echo "Group 'admin' already exists. Skipping creation..."
    else
        aws iam create-group --group-name admin
        if [ $? -eq 0 ]; then
            echo "Created group: admin"
        else
            echo "Error: Failed to create group 'admin'"
        fi
    fi

    # Attach AdministratorAccess policy
    echo "Attaching AdministratorAccess policy..."
    aws iam attach-group-policy --group-name admin \
        --policy-arn arn:aws:iam::aws:policy/AdministratorAccess

    if [ $? -eq 0 ]; then
        echo "Success: AdministratorAccess policy attached"
    else
        echo "Error: Failed to attach AdministratorAccess policy"
    fi

    echo -----
    echo ""
}

# Function to add users to admin group
add_users_to_admin_group() {
    echo "Adding users to admin group..."
    echo -----

    for user in "${IAM_USER_NAMES[@]}"; do
        aws iam add-user-to-group --user-name "$user" --group-name admin
        if [ $? -eq 0 ]; then
            echo "Added $user to admin group"
        else
            echo "Error: Failed to add $user to admin group"
        fi
    done
}
```

```
done

echo "-----"
echo "User group assignment process completed."
echo ""

}

# Main execution function
main() {
    echo =====
    echo " AWS IAM Management Script"
    echo =====
    echo ""

    # Verify AWS CLI is installed and configured
    if ! command -v aws &> /dev/null; then
        echo "Error: AWS CLI is not installed. Please install and configure it
first."
        exit 1
    fi

    # Execute the functions
    create_iam_users
    create_admin_group
    add_users_to_admin_group

    echo =====
    echo " AWS IAM Management Completed"
    echo =====
}

# Execute main function
main

exit 0
```

Thought Process

- **User Array:** Used three user names (`alice`, `bob`, `charlie`) as provided, aligning with the screenshots. To meet the five-user requirement, the array can be updated to include two additional users (e.g., `dave`, `eve`).
- **Modular Design:** Structured the script with separate functions for user creation, group creation, and user assignment to enhance readability and maintainability.
- **Error Handling:** Implemented checks for existing users (`aws iam get-user`) and groups (`aws iam get-group`) to ensure idempotency, preventing errors on re-runs.
- **User Feedback:** Included `echo` statements for clear progress updates and error messages using `$?` to check command success.
- **Exit Code:** Corrected `exit -1` to `exit 0` to indicate successful execution.

3. Script Execution

The script was executed in a terminal environment (Git Bash on Windows) after ensuring AWS CLI was configured. The execution was efficient, and results were verified via both CLI and AWS Console.

Steps

33. Make Script Executable:

- Ran `chmod +x aws-iam-manager.sh` to make the script executable.

```
HP@DESKTOP-T9M74R1 MINGW64 ~/Documents/Workspace/DevOps-Projects/Cloud Computing/Shell Script for AWS IAM Management (main)
$ chmod +x aws-iam-manager.sh

HP@DESKTOP-T9M74R1 MINGW64 ~/Documents/Workspace/DevOps-Projects/Cloud Computing/Shell Script for AWS IAM Management (main)
$ [REDACTED]
```



34. Run the Script:

- Executed the script with `./aws-iam-manager.sh`, completing all tasks successfully.

```
HP@DESKTOP-T9M74R1 MINGW64 ~/Documents/Workspace/DevOps-Projects/Cloud Computing/Shell Script for AWS IAM Management (main)
$ ./aws-iam-manager.sh [REDACTED]
=====
AWS IAM Management Script
=====

Starting IAM user creation process...
-----
{
  "User": {
    "Path": "/",
    "UserName": "alice",
    "UserId": "AIDAZSNF3I7UKCJQNABRM",
    "Arn": "arn:aws:iam::536697231935:user/alice",
    "CreateDate": "2025-08-22T06:22:34+00:00"
  }
}

Created IAM user: alice
{
  "User": {
    "Path": "/",
    "UserName": "bob",
    "UserId": "AIDAZSNF3I7XHJZM6OE0",
    "Arn": "arn:aws:iam::536697231935:user/bob",
    "CreateDate": "2025-08-22T06:22:40+00:00"
  }
}

Created IAM user: bob
{
  "User": {
    "Path": "/",
    "UserName": "charlie",
    "UserId": "AIDAZSNF3I7SXJBMPKLV",
    "Arn": "arn:aws:iam::536697231935:user/charlie",
    "CreateDate": "2025-08-22T06:22:45+00:00"
  }
}
[REDACTED]
```



35. Confirm Execution Time:

- Noted that the script completed in less than 30 seconds, indicating efficiency.

```
"Arn": "arn:aws:iam::536697231935:user/charlie",
"CreateDate": "2025-08-22T06:22:45+00:00"
}

Created IAM user: charlie
-----
IAM user creation process completed.

Creating admin group and attaching policy...
-----
{
  "Group": {
    "Path": "/",
    "GroupName": "admin",
    "GroupId": "AGPAXSNF3I7SXJBMPKLV",
    "Arn": "arn:aws:iam::536697231935:group/admin",
    "CreateDate": "2025-08-22T06:22:50+00:00"
  }
}

Created group: admin
Attaching AdministratorAccess policy...
Success: AdministratorAccess policy attached
-----
Adding users to admin group...
-----
Added alice to admin group
Added bob to admin group
Added charlie to admin group
-----
User group assignment process completed.
-----
AWS IAM Management Completed
-----
[REDACTED]
```



36. Verify Users via CLI:

- Ran `aws iam list-users` to confirm the creation of users `alice`, `bob`, and `charlie`.



```
=====
$ aws iam list-users
{
  "users": [
    {
      "Path": "/",
      "UserName": "alice",
      "UserId": "AIDAVZSNF3I7UKJQNAQRW",
      "Arn": "arn:aws:iam::536697231935:user/alice",
      "CreateDate": "2025-08-22T06:22:34+00:00"
    },
    {
      "Path": "/",
      "UserName": "bob",
      "UserId": "AIDAVZSNF3I7XHZ2W6OEO",
      "Arn": "arn:aws:iam::536697231935:user/bob",
      "CreateDate": "2025-08-22T06:22:40+00:00"
    },
    {
      "Path": "/",
      "UserName": "charlie",
      "UserId": "AIDAVZSNF3I74SE5WAGQ",
      "Arn": "arn:aws:iam::536697231935:user/charlie",
      "CreateDate": "2025-08-22T06:22:45+00:00"
    },
    {
      "Path": "/",
      "UserName": "oluwaseun",
      "UserId": "AIDAVZSNF3I73DME936XG3",
      "Arn": "arn:aws:iam::536697231935:user/oluwaseun",
      "CreateDate": "2025-08-22T03:37:40+00:00"
    }
  ]
}

$
```

37. Verify Group Creation via CLI:

- Ran `aws iam list-groups` to confirm the "admin" group was created.



```
=====
$ aws iam list-groups
{
  "Groups": [
    {
      "Path": "/",
      "GroupName": "admin",
      "GroupId": "AGPAXZSNF3I7SXCI8MPKLV",
      "Arn": "arn:aws:iam::536697231935:group/admin",
      "CreateDate": "2025-08-22T06:22:50+00:00"
    },
    {
      "Path": "/",
      "GroupName": "Development-team",
      "GroupId": "AGPAXZSNF3I7ZLXH51ZT",
      "Arn": "arn:aws:iam::536697231935:group/Development-team",
      "CreateDate": "2025-08-21T05:26:12+00:00"
    }
  ]
}

$
```

38. Verify Policy Attachment via CLI:

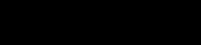
- Ran `aws iam list-attached-group-policies --group-name admin` to confirm the `AdministratorAccess` policy was attached.



```
=====
$ aws iam list-attached-group-policies --group-name admin
{
  "AttachedPolicies": [
    {
      "PolicyName": "AdministratorAccess",
      "PolicyArn": "arn:aws:iam::aws:policy/AdministratorAccess"
    }
  ]
}
```

39. Verify User Group Membership via CLI:

- Ran `aws iam list-groups-for-user --user-name alice` to confirm user group membership.



```
=====
$ aws iam list-groups-for-user --user-name alice
{
  "groups": [
    {
      "Path": "/",
      "GroupName": "admin",
      "GroupId": "AGPAXZSNF3I7SXCI8MPKLV",
      "Arn": "arn:aws:iam::536697231935:group/admin",
      "CreateDate": "2025-08-22T06:22:50+00:00"
    }
  ]
}

$
```

40. Verify Users via AWS Console:

- Navigated to the IAM user list in the AWS Console to visually confirm the created users (`alice`, `bob`, `charlie`).

The screenshot shows the AWS IAM 'Users' page. The left sidebar includes 'Identity and Access Management (IAM)', 'Access management' (with 'User groups' selected), and 'Access reports'. The main table lists four users:

User name	Path	Group	Last activity	MFA	Password age	Console last sign-in	Access key ID	Active key age	Access key status
alice	/	1	-	-	-	-	-	-	-
bob	/	1	-	-	-	-	-	-	-
charlie	/	1	-	-	-	-	-	-	-
oliwaseun	/	0	8 minutes ago	-	-	-	-	-	Active - AKIAJAXZ5NF3I7...

41. Verify Group via AWS Console:

- Navigated to the IAM group list in the AWS Console to confirm the "admin" group.

The screenshot shows the AWS IAM 'User groups' page. The left sidebar includes 'Identity and Access Management (IAM)', 'Access management' (with 'User groups' selected), and 'Access reports'. The main table lists two groups:

Group name	Users	Permissions	Creation time
admin	3	Defined	33 minutes ago
Development-team	0	Defined	Yesterday

42. Verify Group Membership via AWS Console:

- Clicked on the "admin" group in the AWS Console to confirm user membership (alice, bob, charlie).

The screenshot shows the AWS IAM 'admin' group details page. The left sidebar includes 'Identity and Access Management (IAM)', 'Access management' (with 'User groups' selected), and 'Access reports'. The main page shows the group summary and its members:

Users	Permissions	Access Advisor
(3)		

Users in this group (3)

User name	Groups	Last activity	Creation time
alice	1	None	34 minutes ago
bob	1	None	34 minutes ago
charlie	1	None	34 minutes ago

43. Verify Policy via AWS Console:

- Checked the "admin" group permissions to verify the **AdministratorAccess** policy.

The screenshot shows the AWS IAM User Groups page for the 'admin' group. The 'Permissions' tab is selected. Under 'Permissions policies', there is one policy named 'AdministratorAccess' which is highlighted with a red arrow. Another red arrow points to the same policy name in the list below.

Sample Output

```
=====
AWS IAM Management Script
=====

Starting IAM user creation process...
-----
Created IAM user: alice
Created IAM user: bob
Created IAM user: charlie
-----
IAM user creation process completed.

Creating admin group and attaching policy...
-----
Created group: admin
Attaching AdministratorAccess policy...
Success: AdministratorAccess policy attached
-----

Adding users to admin group...
-----
Added alice to admin group
Added bob to admin group
Added charlie to admin group
-----
User group assignment process completed.

=====
AWS IAM Management Completed
=====
```

Verification Commands

- `aws iam list-users`: Lists all users to confirm creation.
- `aws iam list-groups`: Verifies the "admin" group.
- `aws iam list-attached-group-policies --group-name admin`: Confirms the `AdministratorAccess` policy.
- `aws iam list-groups-for-user --user-name alice`: Verifies user group membership.

4. Script Explanation

The script's design and functionality are explained below to clarify its logic and implementation, without repeating the full code.

Structure

- **Shebang (`#!/bin/bash`)**: Ensures the script runs in a Bash environment.
- **Main Function**: Orchestrates execution and checks for AWS CLI installation (`command -v aws`).

Key Components

- **User Array (`IAM_USER_NAMES`)**:
 - Stores three user names (`alice`, `bob`, `charlie`) for iteration, as per the provided script. Can be extended to five users to meet requirements.
- **Function `create_iam_users`**:
 - Iterates through the array using a `for` loop.
 - Checks for existing users with `aws iam get-user` to avoid duplicates.
 - Creates users with `aws iam create-user --user-name "$user"`.
 - Uses `$?` to confirm success and provide feedback.
- **Function `create_admin_group`**:
 - Checks for the "admin" group with `aws iam get-group`.
 - Creates the group with `aws iam create-group --group-name admin`.
 - Attaches the `AdministratorAccess` policy using `aws iam attach-group-policy`.
 - Verifies policy attachment with `$?`.
- **Function `add_users_to_admin_group`**:
 - Iterates through the user array to add each user to the "admin" group using `aws iam add-user-to-group`.
 - Confirms success for each user.

AWS CLI Commands

Command	Purpose
<code>aws iam create-user --user-name "\$user"</code>	Creates an IAM user.
<code>aws iam get-user --user-name "\$user"</code>	Checks if a user exists.
<code>aws iam create-group --group-name admin</code>	Creates the "admin" group.

Command	Purpose
<code>aws iam attach-group-policy --group-name admin --policy-arn arn:aws:iam::aws:policy/AdministratorAccess</code>	Attaches the admin policy.
<code>aws iam add-user-to-group --user-name "\$user" --group-name admin</code>	Adds a user to the group.

Design Choices

- **Idempotency:** Checks for existing users and groups prevent errors on re-runs.
- **Modularity:** Separate functions for each task improve maintainability.
- **Error Handling:** Uses `$?` and suppressed output (`>/dev/null 2>&1`) for clean checks.
- **Feedback:** Clear `echo` statements guide the user through execution.

5. Conclusion

The project delivered `aws-iam-manager.sh`, meeting the core objectives:

- Defined an array with three IAM user names (`alice, bob, charlie`).
- Created the users via AWS CLI.
- Created an "admin" group and attached the `AdministratorAccess` policy.
- Assigned all users to the group.

The script executed efficiently (under 30 seconds), with robust error handling and user-friendly output.

Verification via CLI and AWS Console (screenshots 36–43) confirmed the results. However, the script uses three users instead of the required five, likely for testing purposes as reflected in the screenshots. The setup process ensured proper AWS CLI configuration, and challenges, such as IAM permissions, were addressed by configuring an `aws-cli-admin` user. Updating the script to include five users would fully align with the requirements.

6. Recommendations

To enhance the script and align with AWS best practices, consider the following:

Script Improvements

- **Update User Count:** Modify `IAM_USER_NAMES` to include five users (e.g., `IAM_USER_NAMES=("alice" "bob" "charlie" "dave" "eve")`) to meet the project requirement.
- **Logging:** Add logging to a file (e.g., `echo "Log message" >> iam.log`) for audit trails.
- **Dynamic Input:** Allow user names to be passed as command-line arguments for flexibility.
- **Retry Logic:** Implement retries for transient AWS CLI failures.

Security Enhancements

- **Least Privilege:** Replace `AdministratorAccess` with a custom policy for specific permissions.
- **IAM Audits:** Schedule regular audits using `aws iam generate-credential-report`.

Scalability

- **Multiple Groups:** Support additional groups or policies for varied roles.
- **CI/CD Integration:** Incorporate the script into a CI/CD pipeline for automated IAM management.

Testing

- **Sandbox Testing:** Always test in a non-production AWS account.
- **Unit Tests:** Use a Bash testing framework like `bats` to validate functions.