# 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).

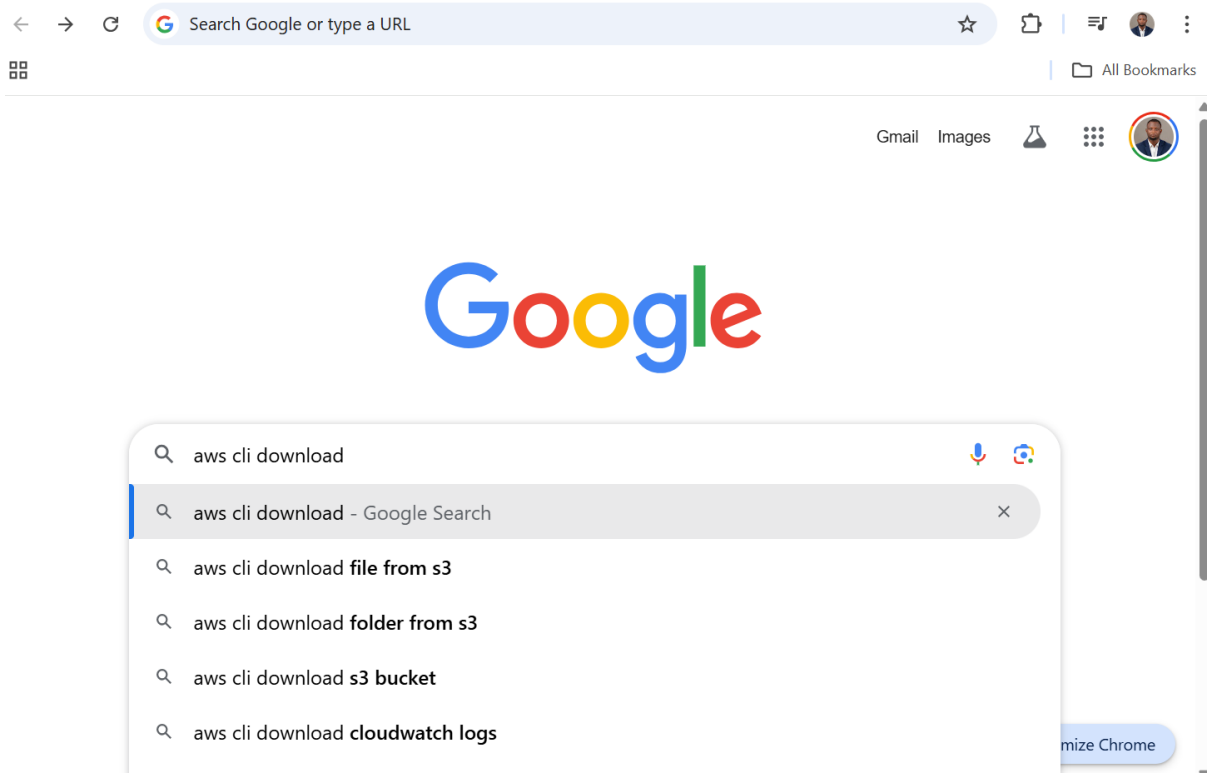
This `README.md` provides comprehensive documentation, detailing the setup, script development, execution, explanation, conclusions, and recommendations. Screenshots (43 in total) are included in the `./img/` directory to illustrate each step visually.

## 1. Installing and Configuring AWS CLI

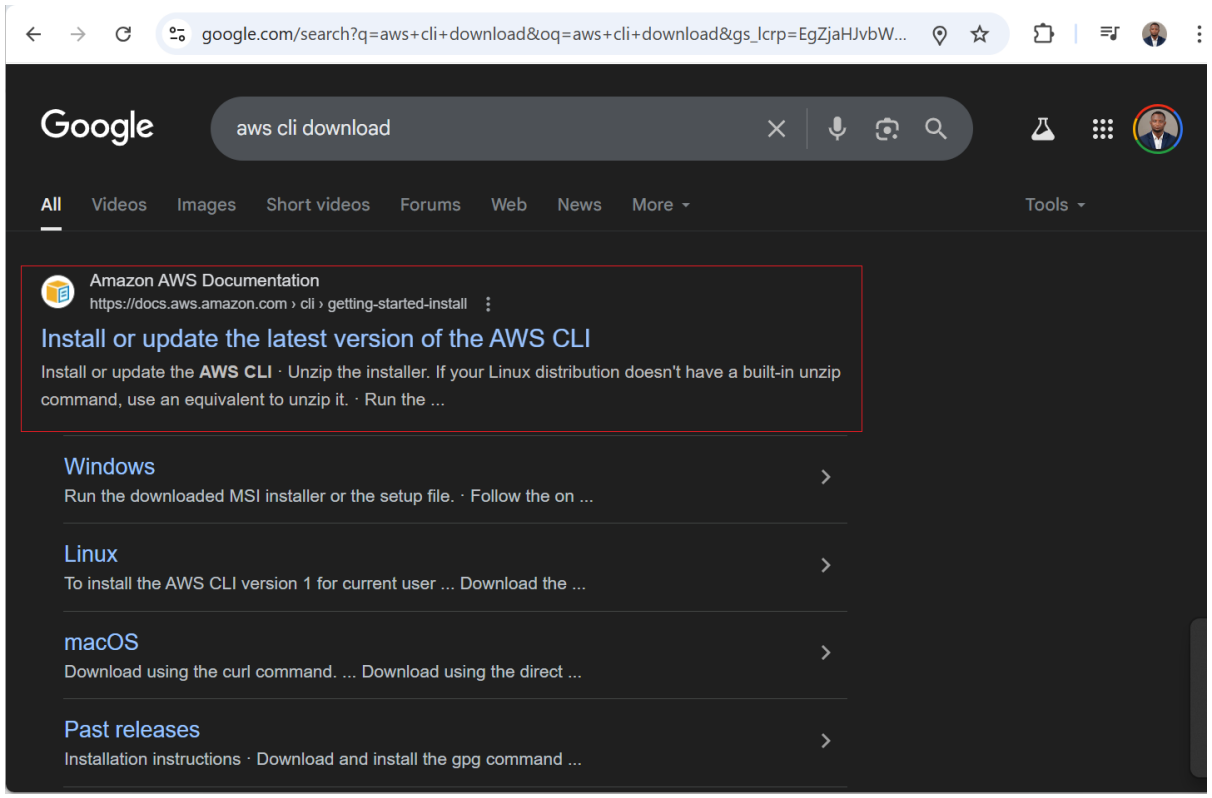
The AWS Command Line Interface (CLI) must be installed and configured with appropriate IAM permissions to enable the script to manage IAM resources. The following steps outline the setup process, with screenshots for each action.

### Steps

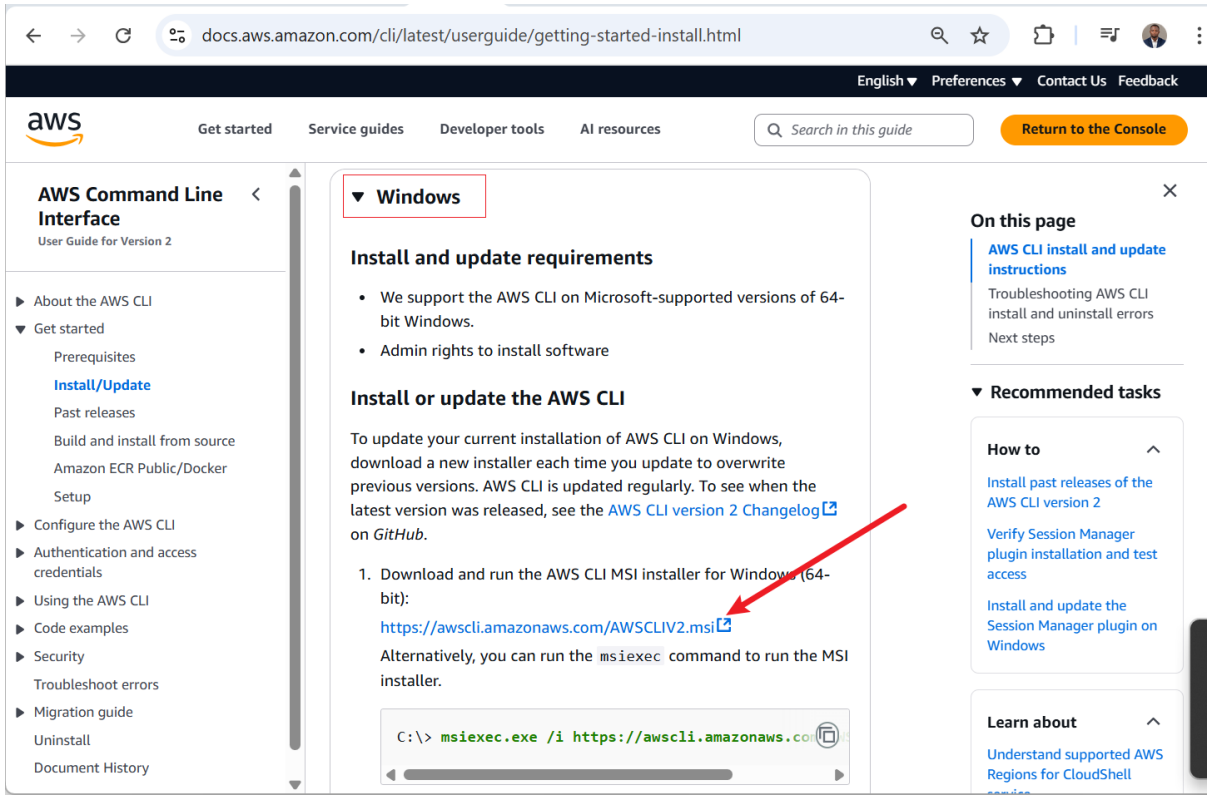
1. **Search for AWS CLI Download**:
    - Searched for the AWS CLI download page to locate the installer for the Windows environment.



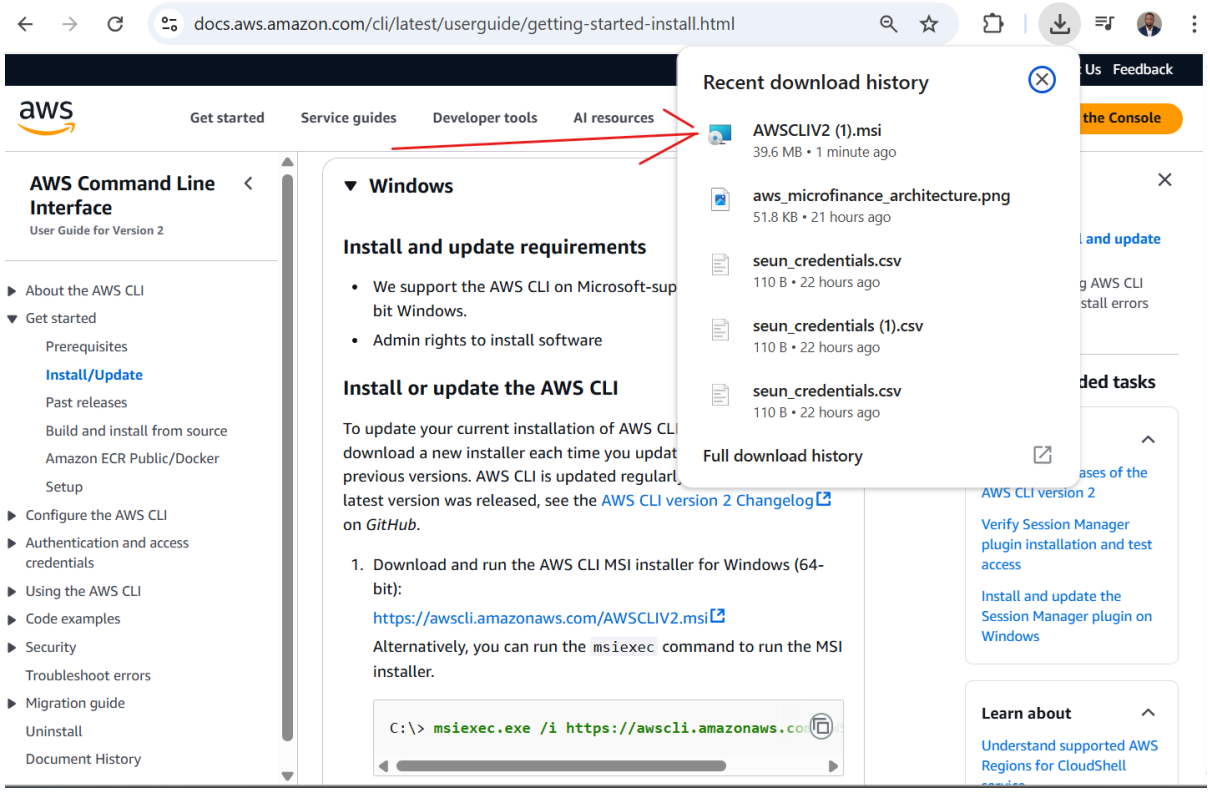

2. **Access the Download Page**:
   - Clicked the first link from the search results to visit the official AWS CLI download page.



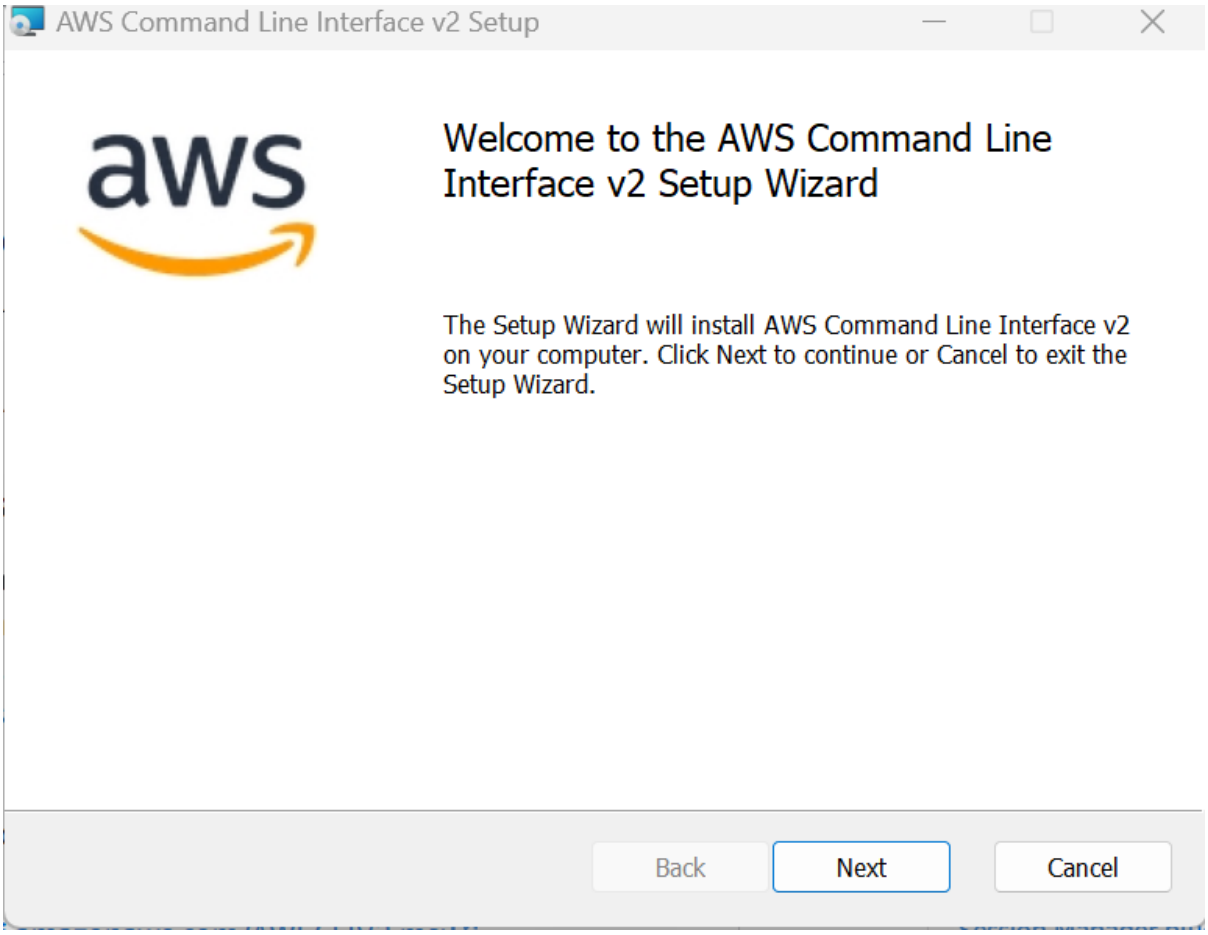

3. **Download Windows MSI Installer**:
   - Scrolled down and selected the Windows MSI download link for AWS CLI v2.

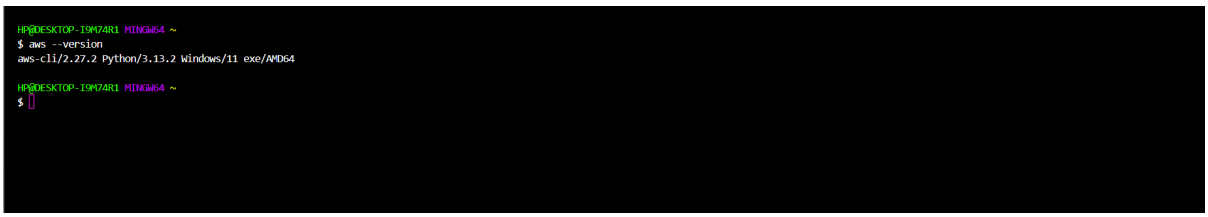



4. **Verify and Run Installer**:
   - Verified the downloaded MSI file and ran it to start the installation.

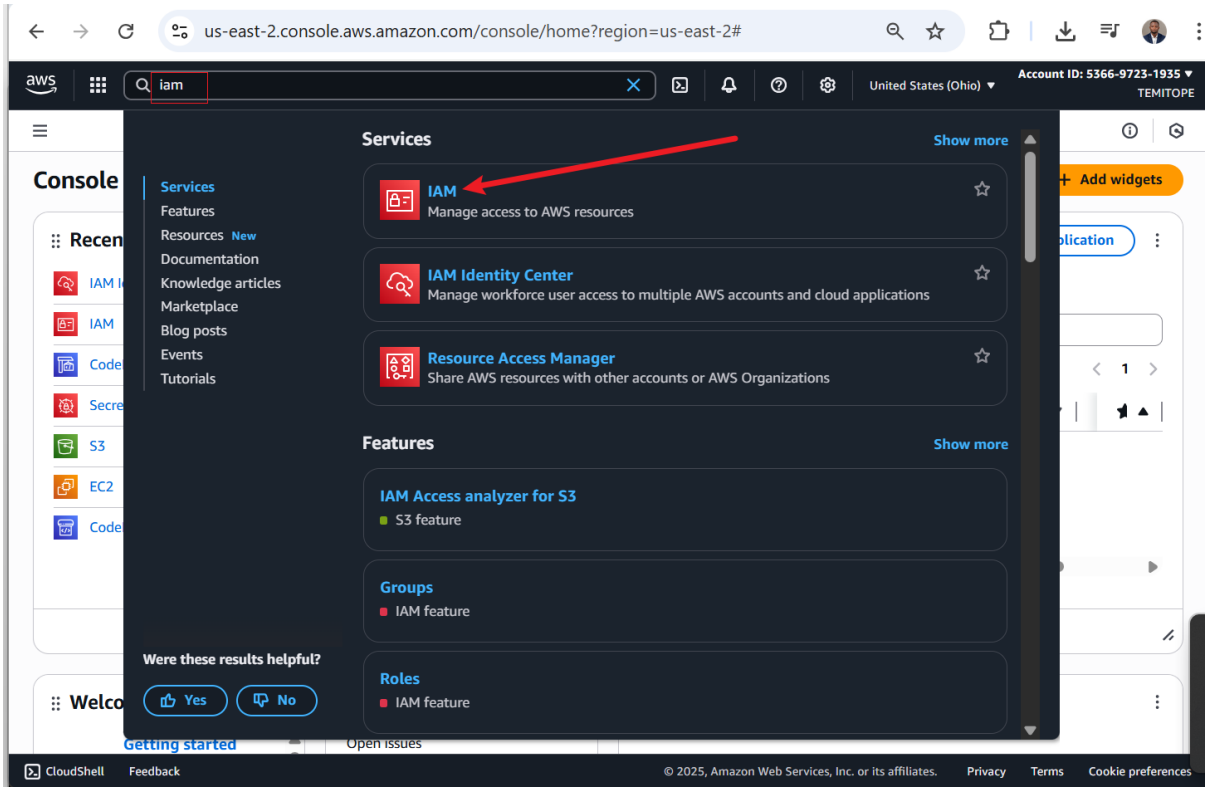



5. **Complete Installation**:
   - Followed the installation prompts to install AWS CLI on the system.

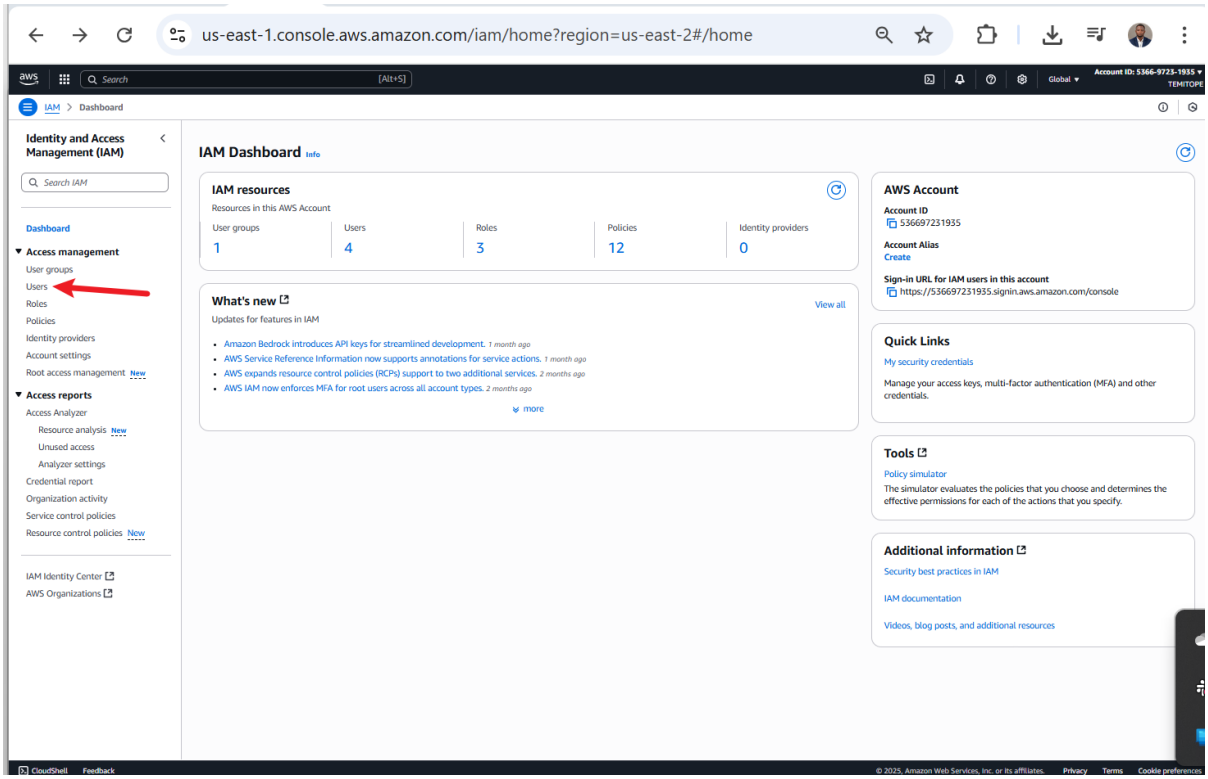
- 


6. **Verify AWS CLI Installation**:
   - Opened a terminal (Git Bash on Windows) and ran `aws --version` to confirm successful installation.
   - 

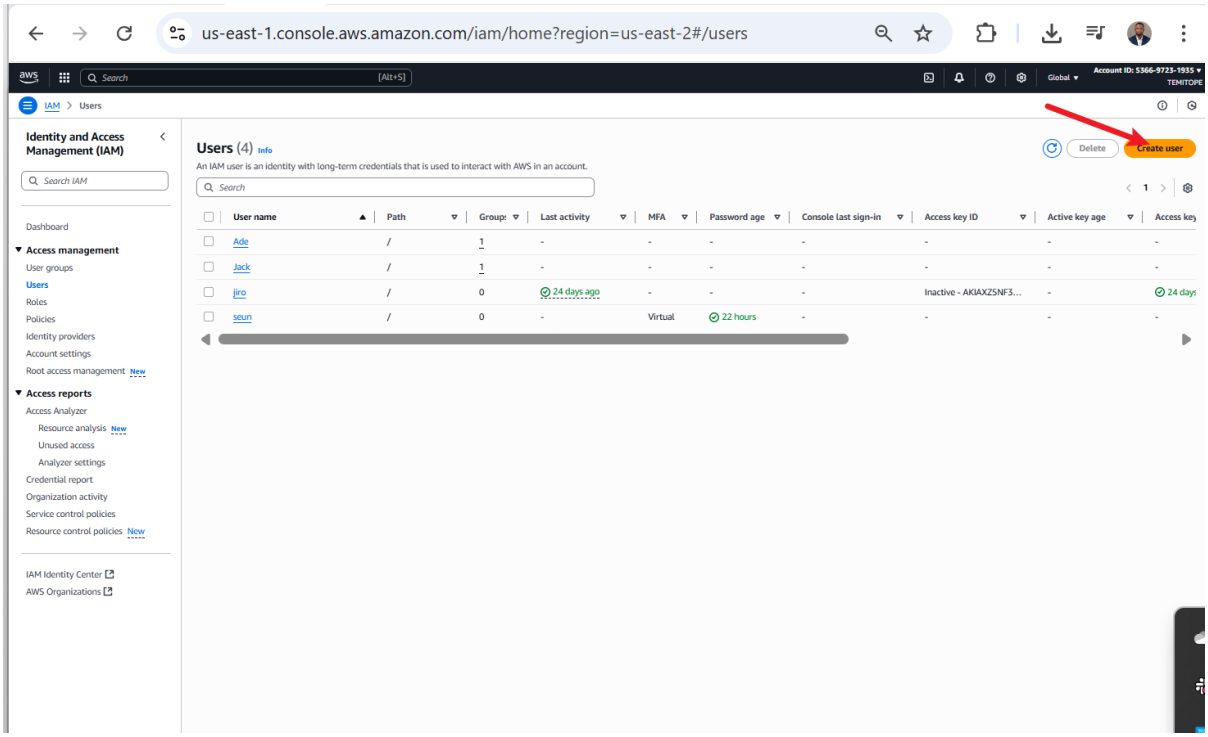

7. **Access AWS IAM Console**:
   - Searched for "IAM" in the AWS Management Console and navigated to the IAM dashboard.



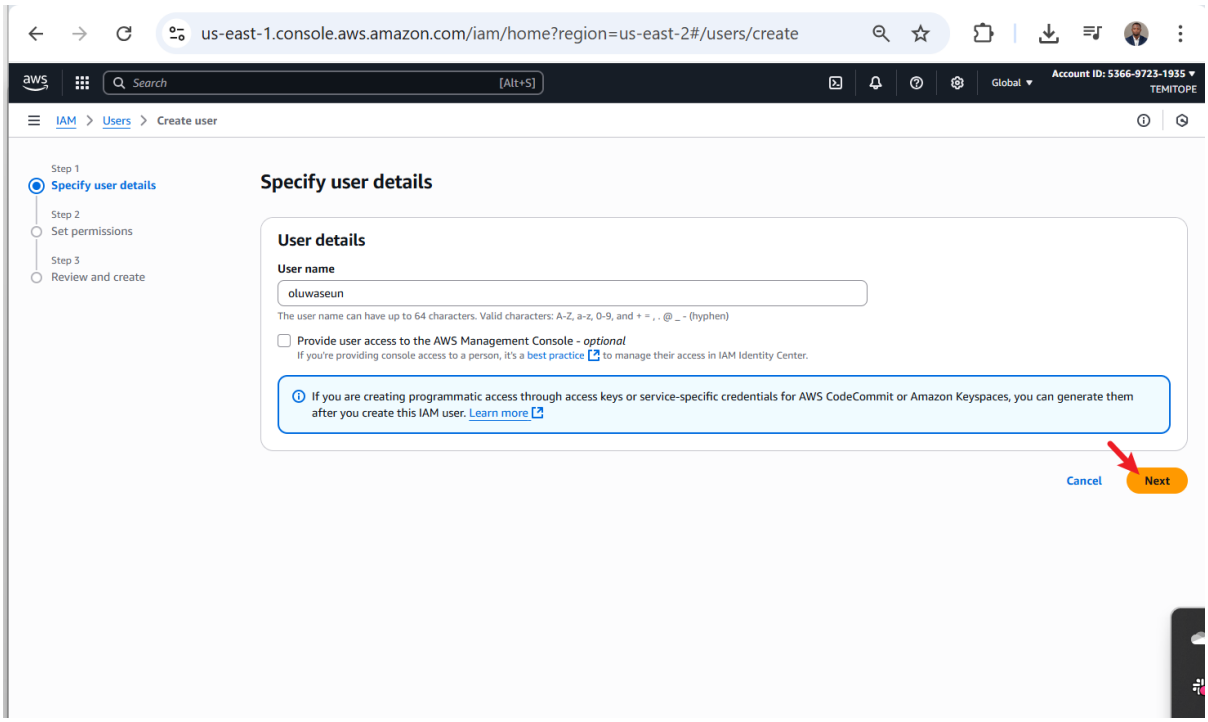

8. **Navigate to Users**:
   - Clicked on "Users" in the IAM dashboard to manage user creation.



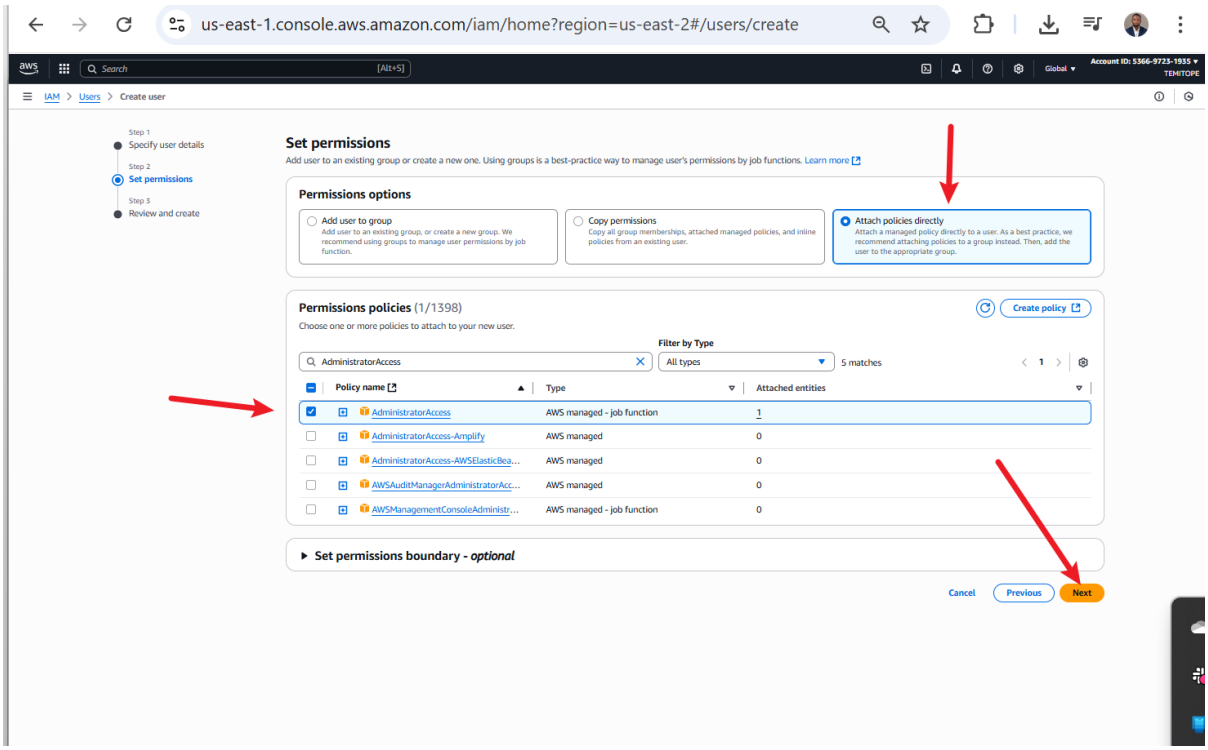

9. **Initiate User Creation**:
   - Clicked "Create user" to set up a new IAM user for AWS CLI access.

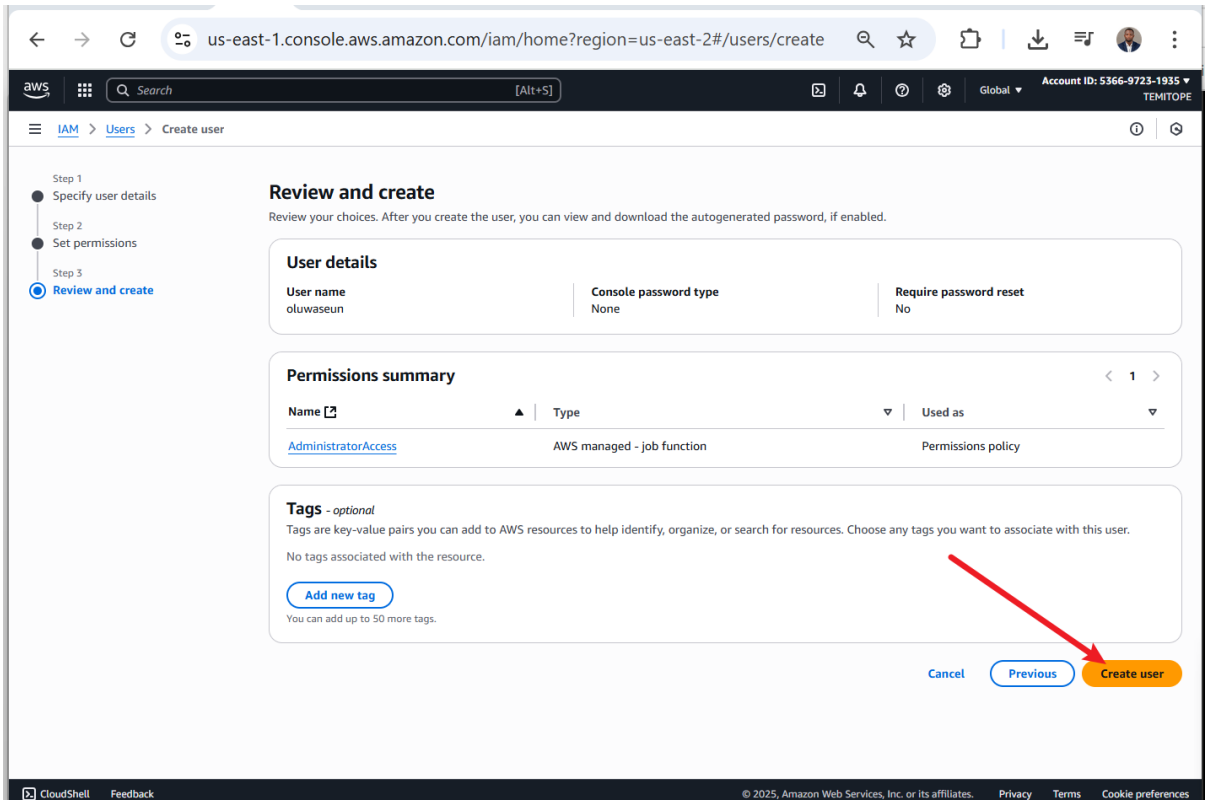



10. **Name User and Skip Console Access**:
    - Named the user (e.g., `aws-cli-admin`) and proceeded without enabling console access, as the user is for CLI use.

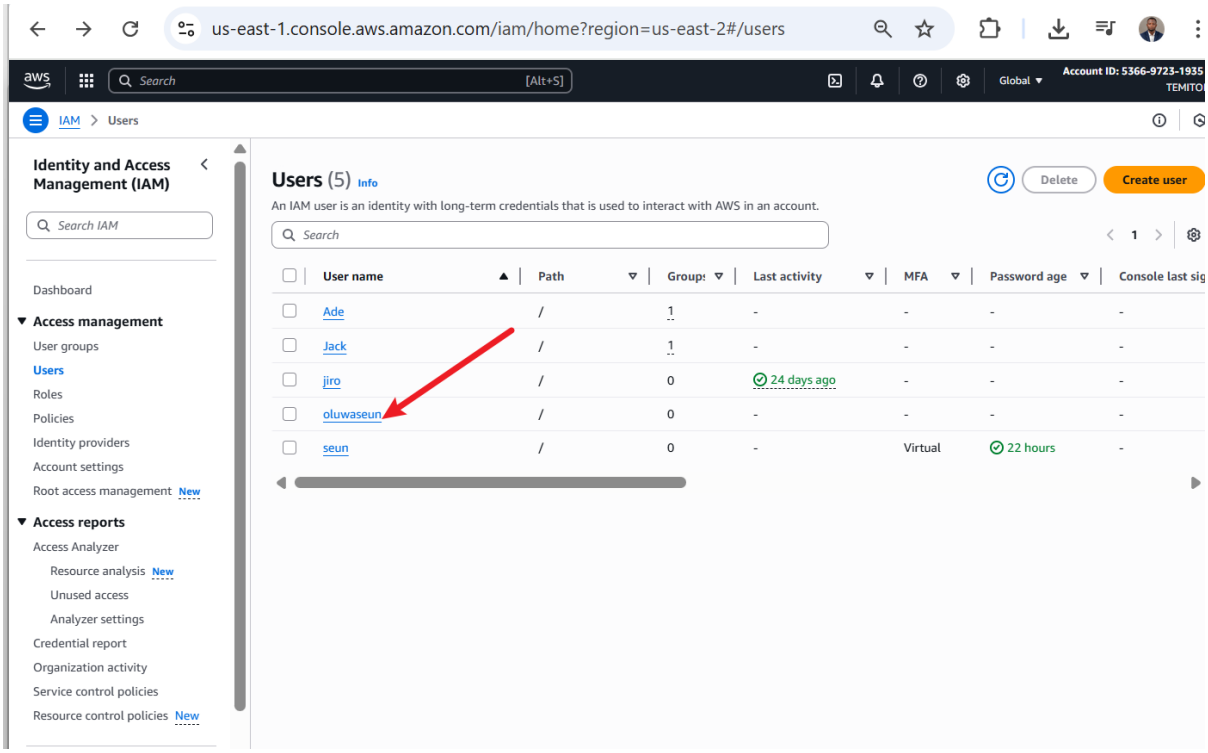



11. **Attach AdministratorAccess Policy**:
    - Attached the `AdministratorAccess` policy directly to the user to grant IAM management permissions.

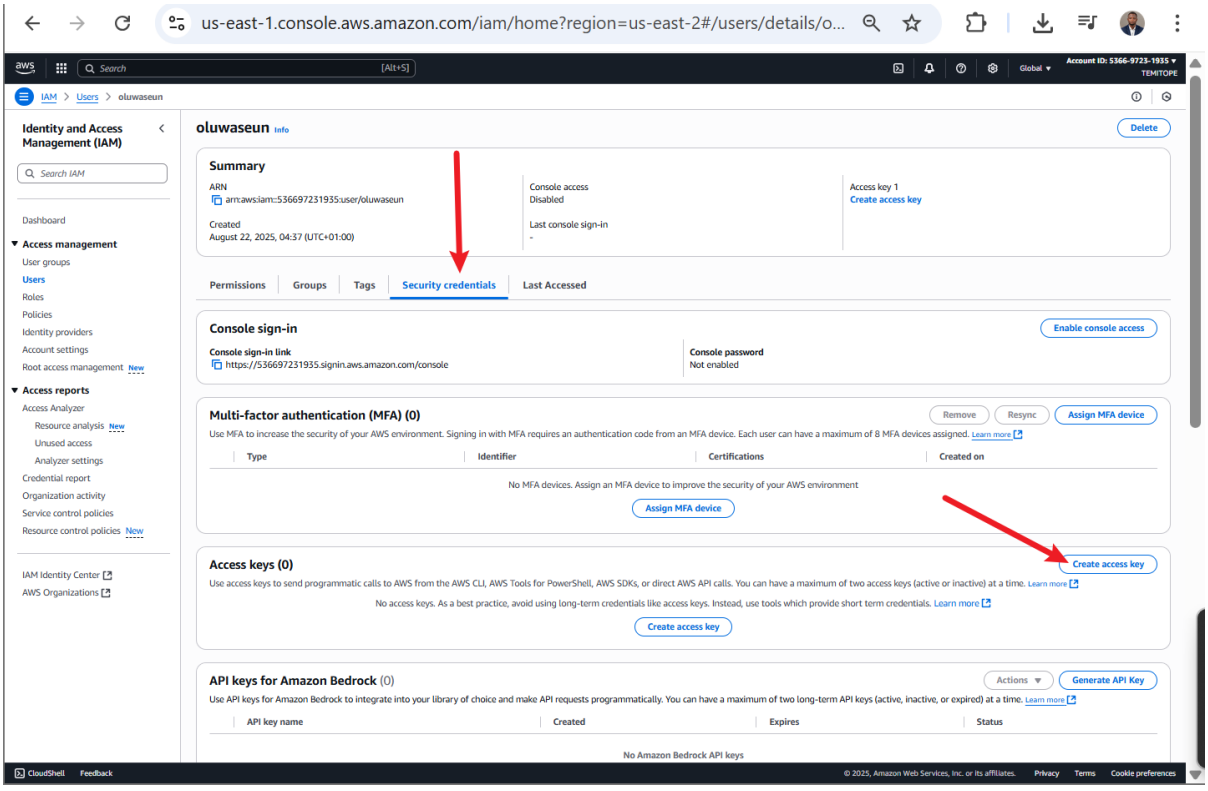
- 


12. **Create the User**:
    - Reviewed settings and clicked "Create user" to finalize the user creation.
    - 

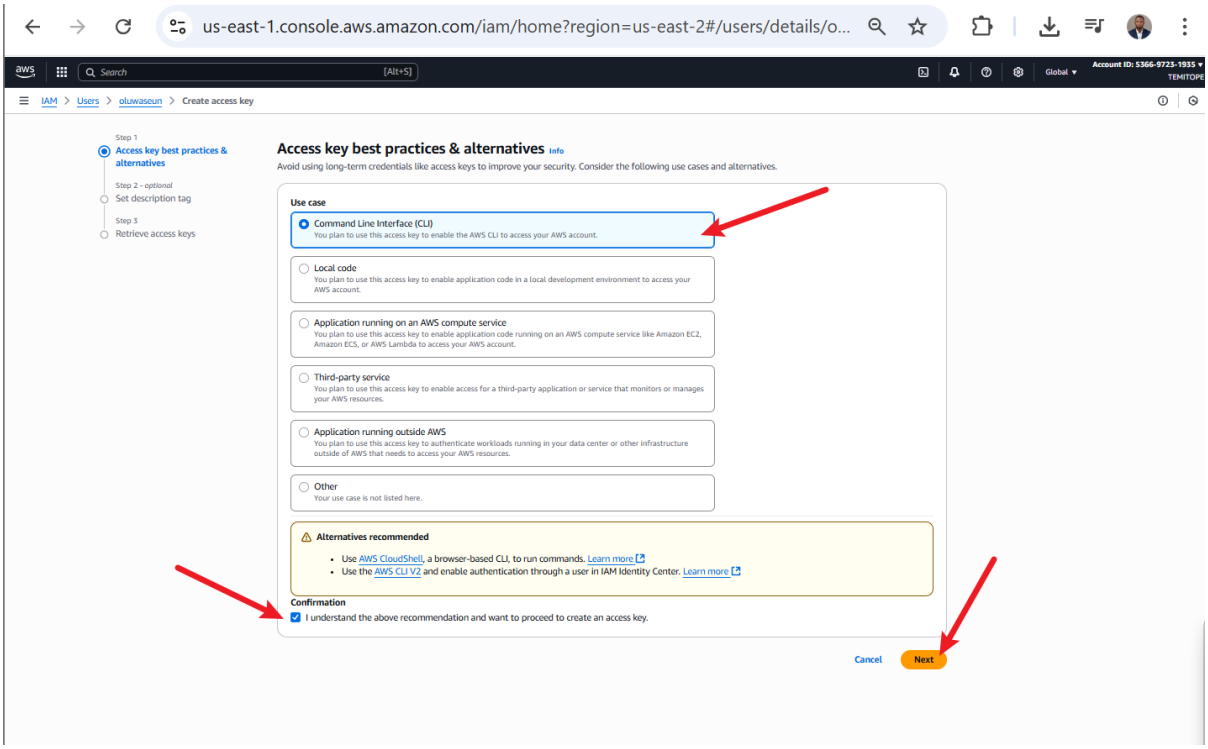

13. **Access User Details**:
    - Navigated to the user list and clicked on the newly created user (`aws-cli-admin`).



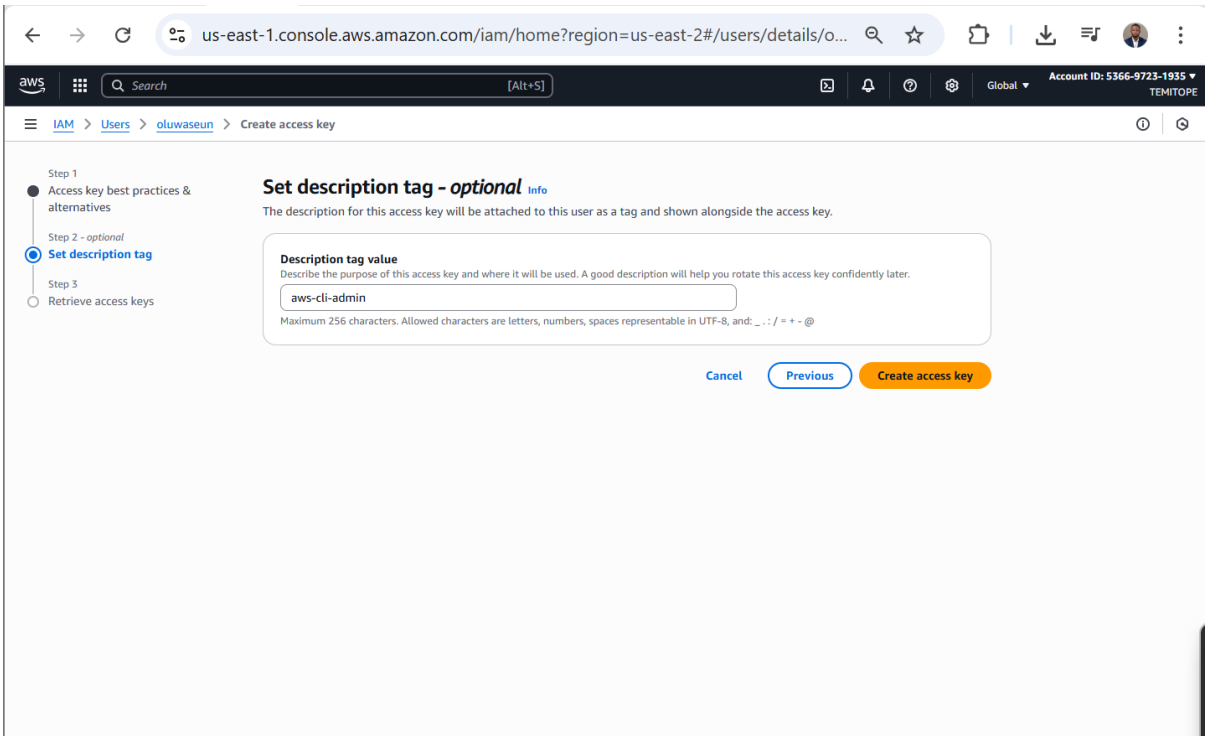

14. **Create Access Key**:
    - Under "Security credentials," clicked "Create access key" to generate CLI credentials.



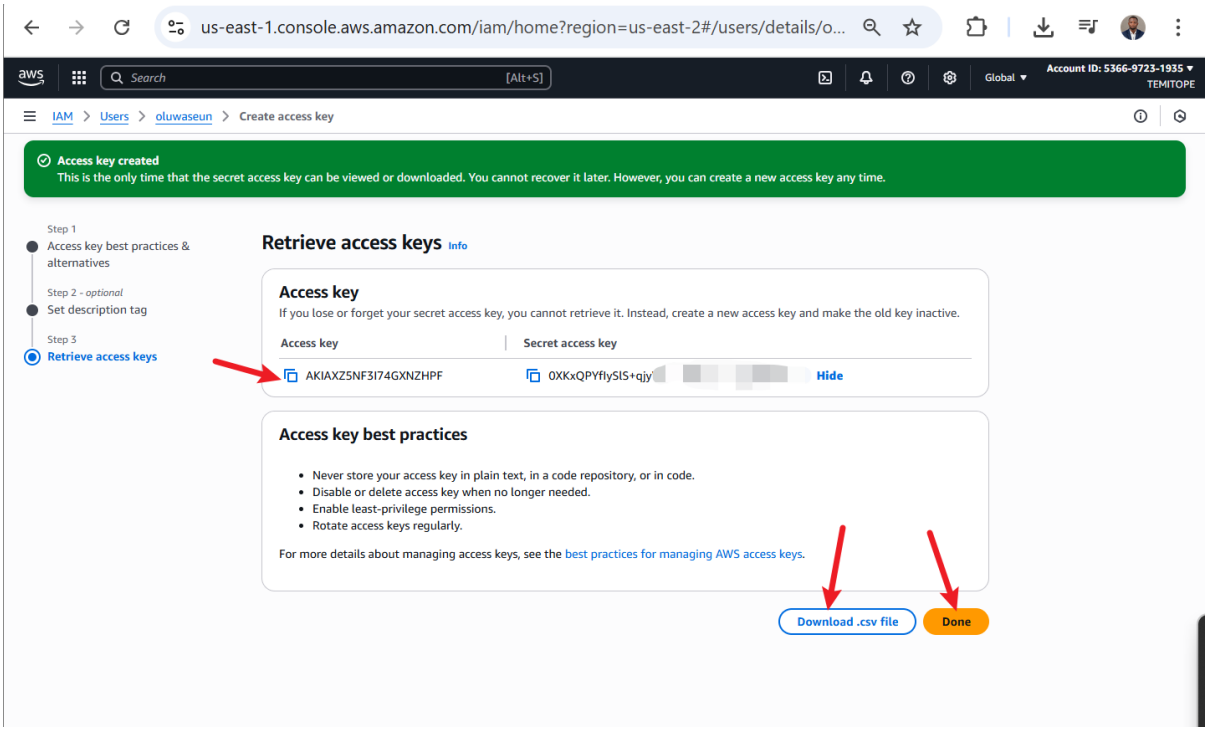

15. **Select CLI Usage**:
    - Selected "Command Line Interface (CLI)" and checked the recommendation box to proceed.

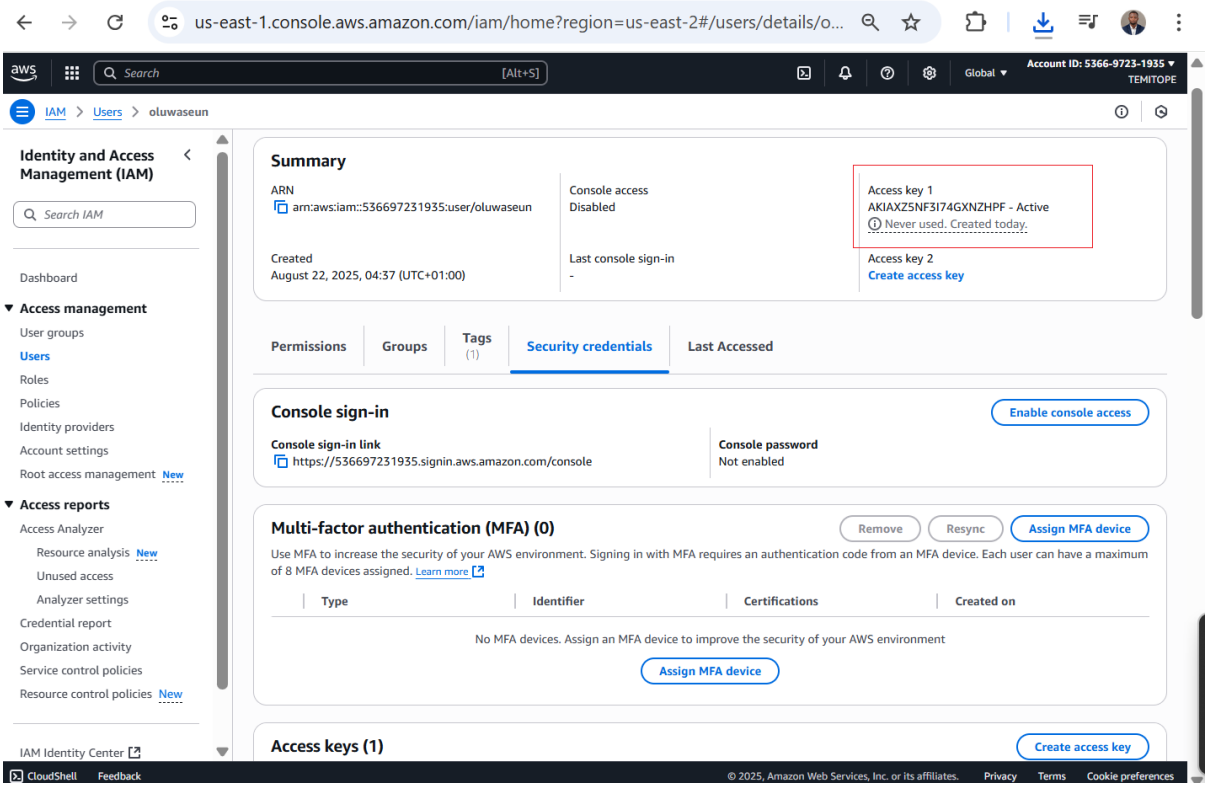



16. **Tag User and Create Key**:
    - Added a tag (e.g., `aws-cli-admin`) and created the access key.

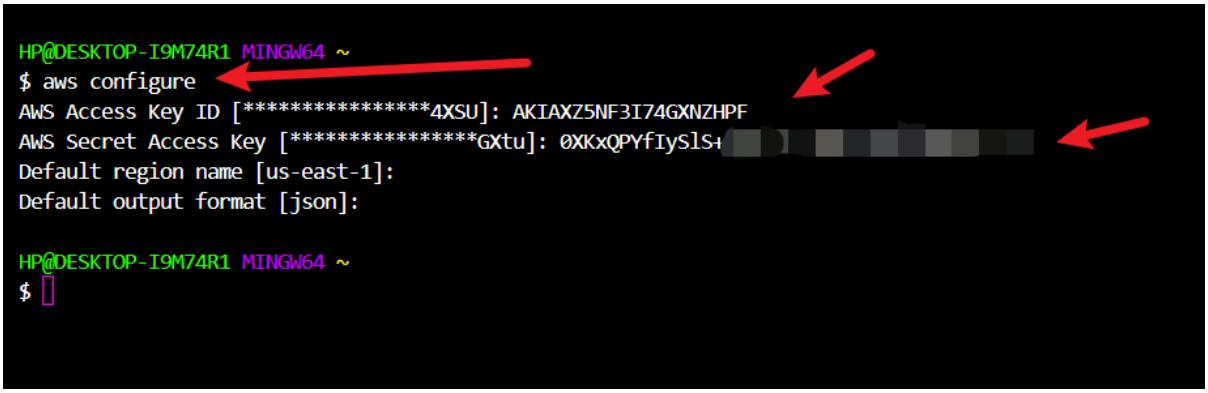



17. **Secure Access Key**:
    - Noted the Access Key ID and Secret Access Key, downloading them as a CSV for secure storage.

- 

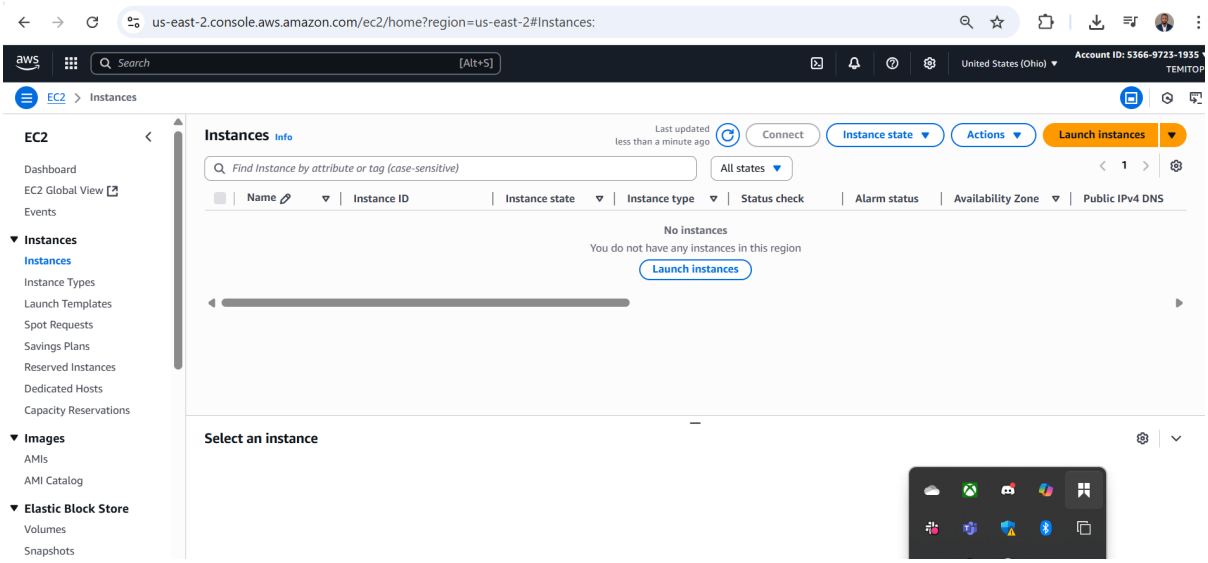

18. **Confirm Access Key**:
    - Verified that the user now has an access key for CLI operations.
    - 


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.

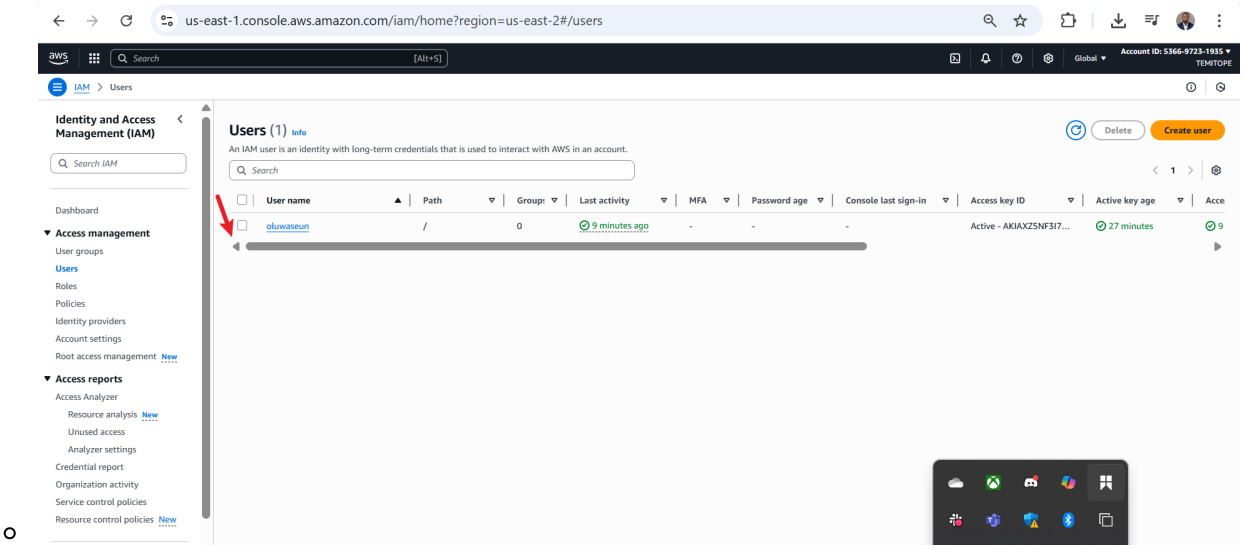



20. **Test CLI Access**:
    - Ran `aws ec2 describe-instances` to verify the CLI user's permissions, confirming access to AWS services.


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.




### 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.

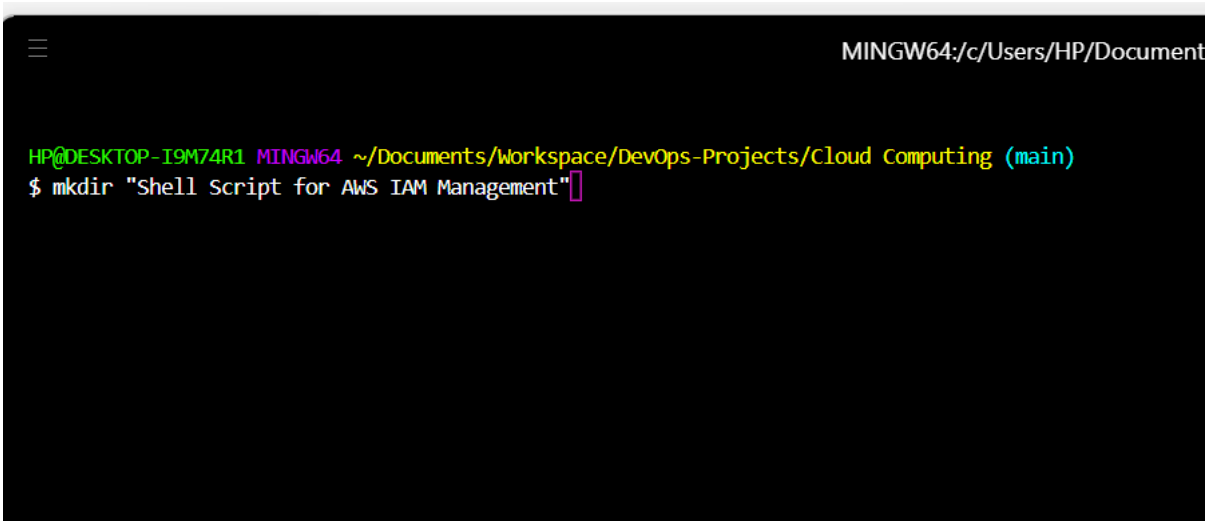
## 2. Script Creation (aws-iam-manager.sh)

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

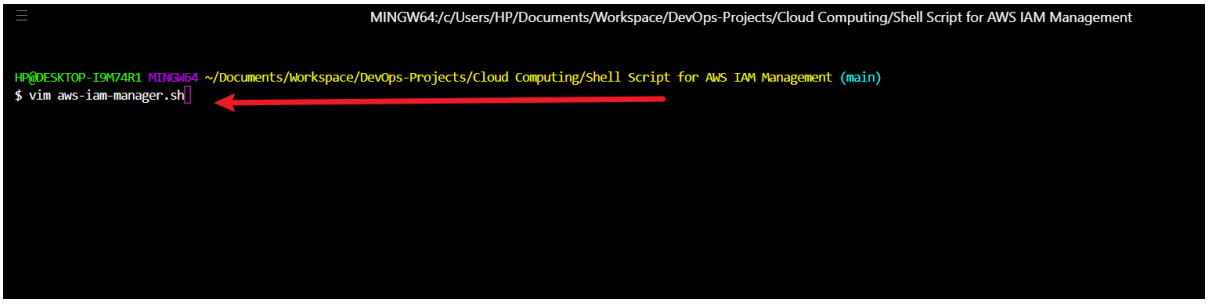
23. **Create Project Directory**:
    - Created a project folder and navigated into it:

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



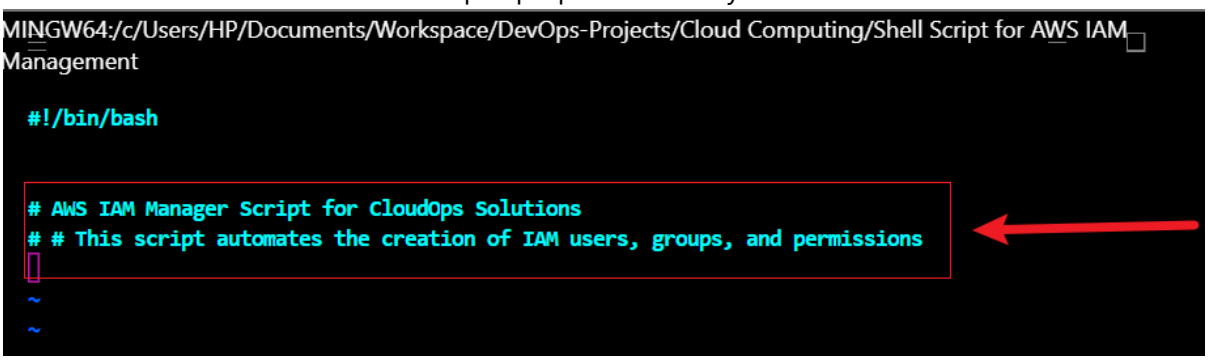

24. **Create Script File**:
    - Created `aws-iam-manager.sh` using `vim` and entered insert mode to write the script.

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




25. **Add Shebang**:
    - Added the shebang (`#!/bin/bash`) to ensure the script runs in Bash.


26. **Document Script Purpose**:
    - Added comments to describe the script's purpose for clarity.




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

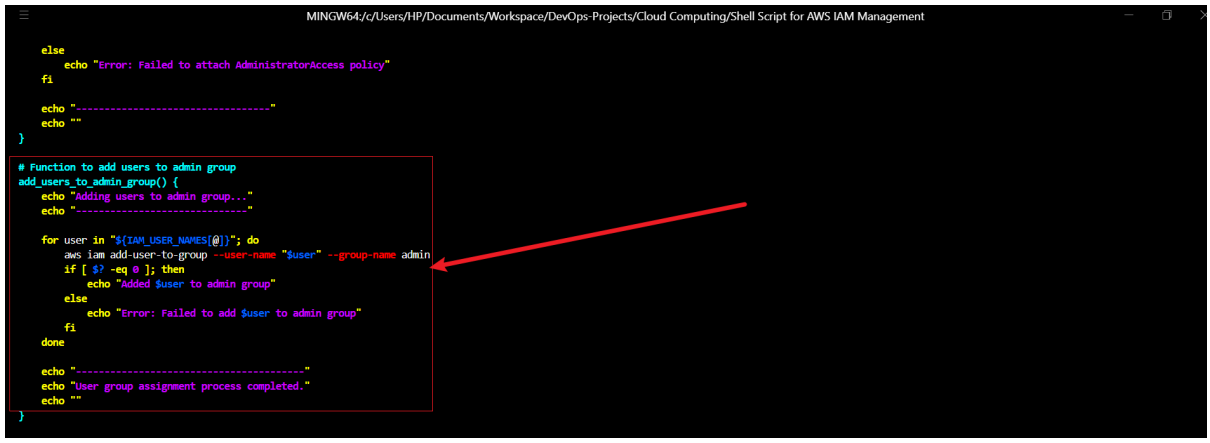
```
#!/bin/bash


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

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

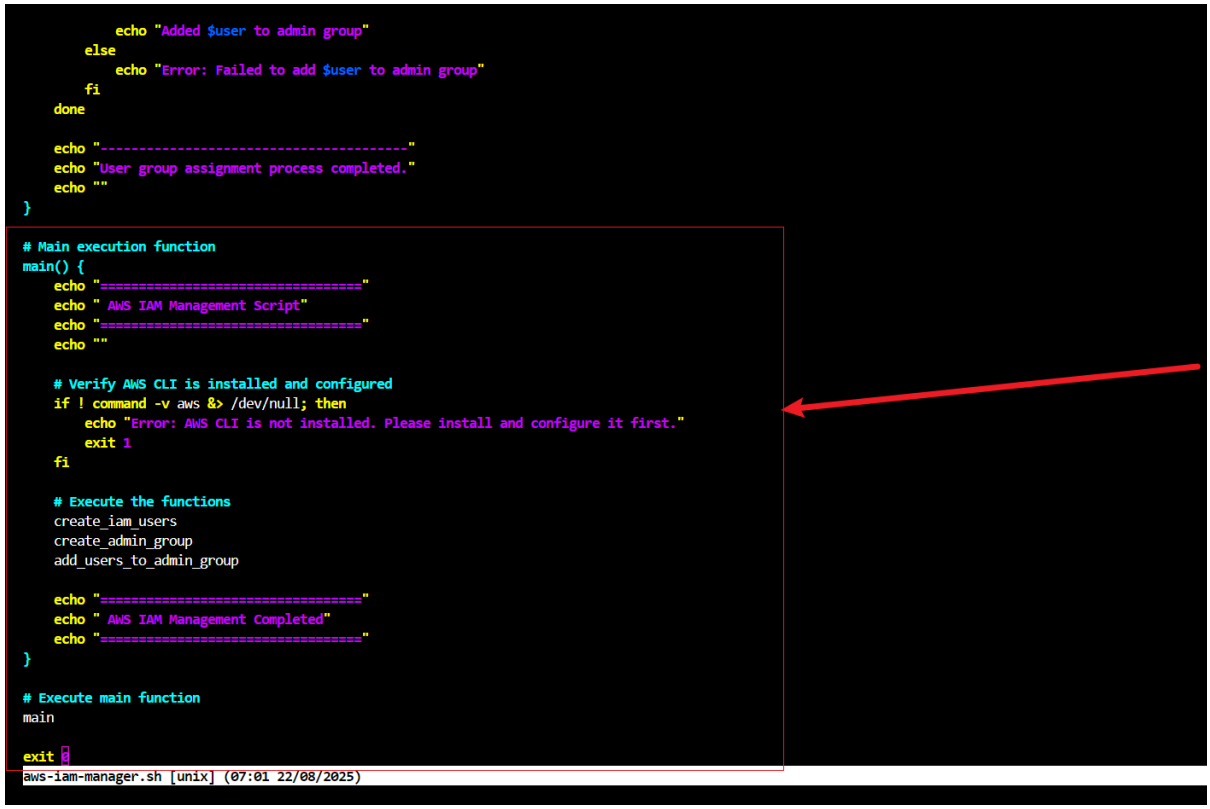
28. **Implement User Creation Function**:
    - Wrote the `create_iam_users` function to iterate through the array and create users.

```
#!/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.

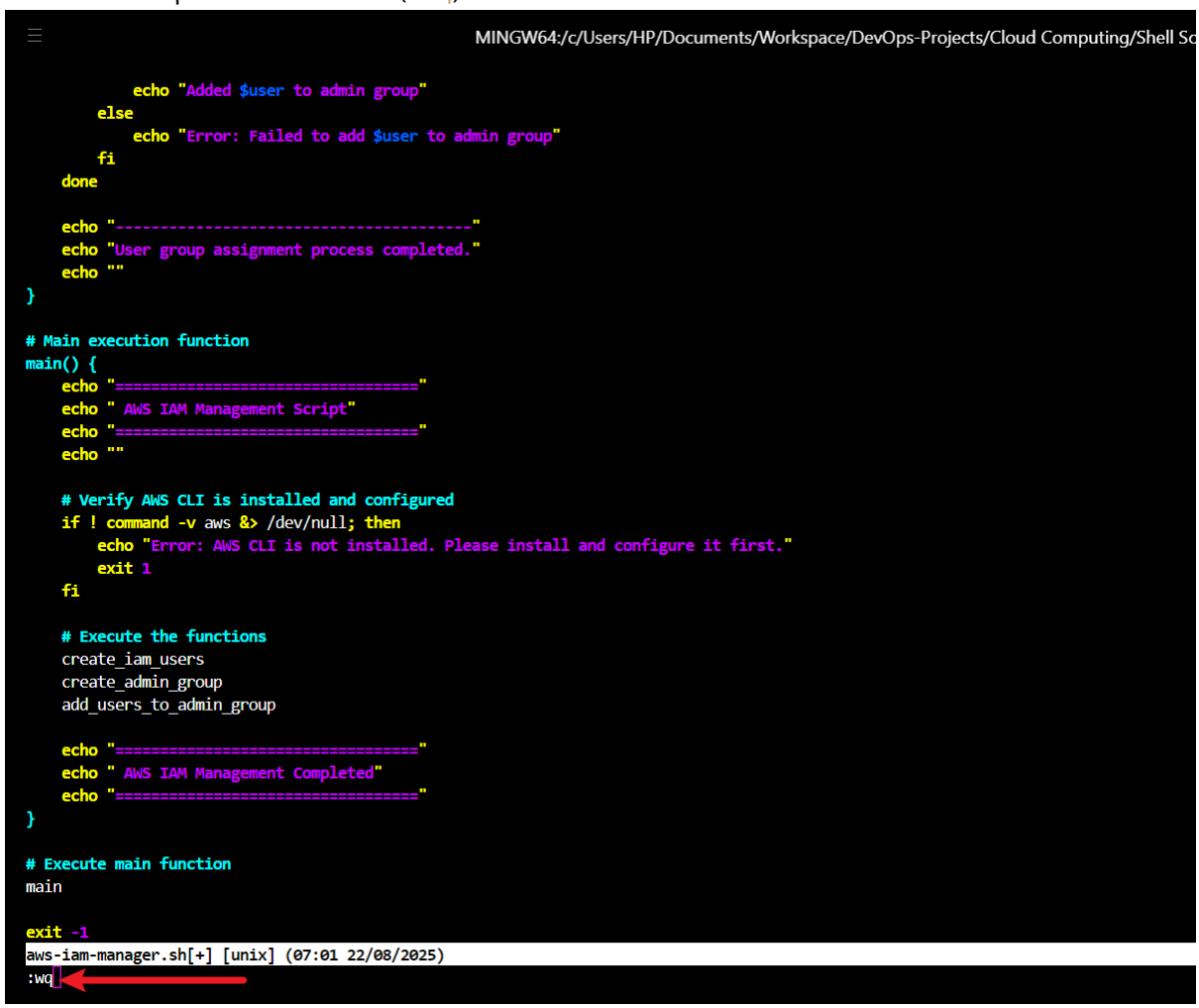
- 


30. **Implement User Assignment Function**:
    - Wrote the `add_users_to_admin_group` function to assign users to the "admin" group.
    - 


31. **Add Main Execution Function**:
    - Implemented the `main` function to orchestrate tasks and check for AWS CLI.
    - 


32. **Save and Exit Vim**:

- Saved the script and exited Vim (`: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

    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
```

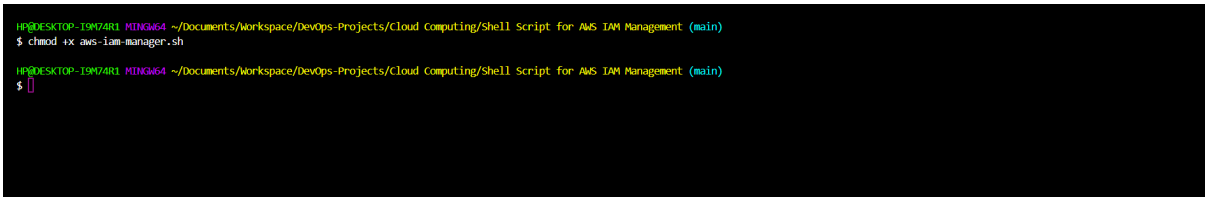
### Thought Process

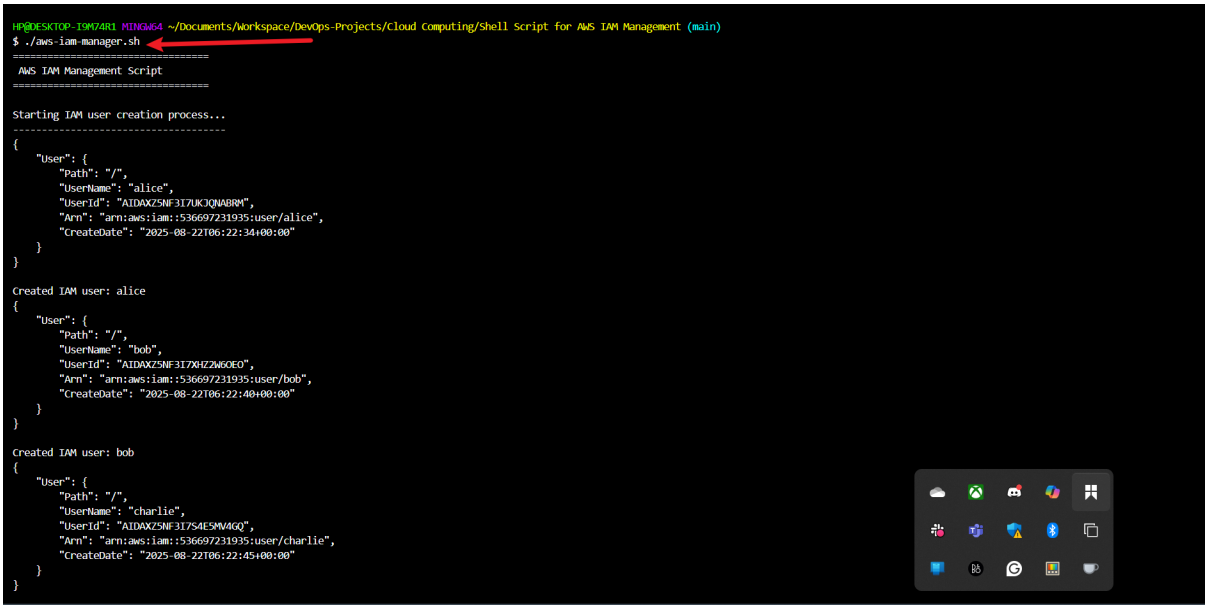
- **User Array**: Used three user names (`alice`, `bob`, `charlie`) as provided, aligning with the screenshots. To meet the five-user requirement, the array can be updated to include two additional users (e.g., `dave`, `eve`).
- **Modular Design**: Structured the script with separate functions for user 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.

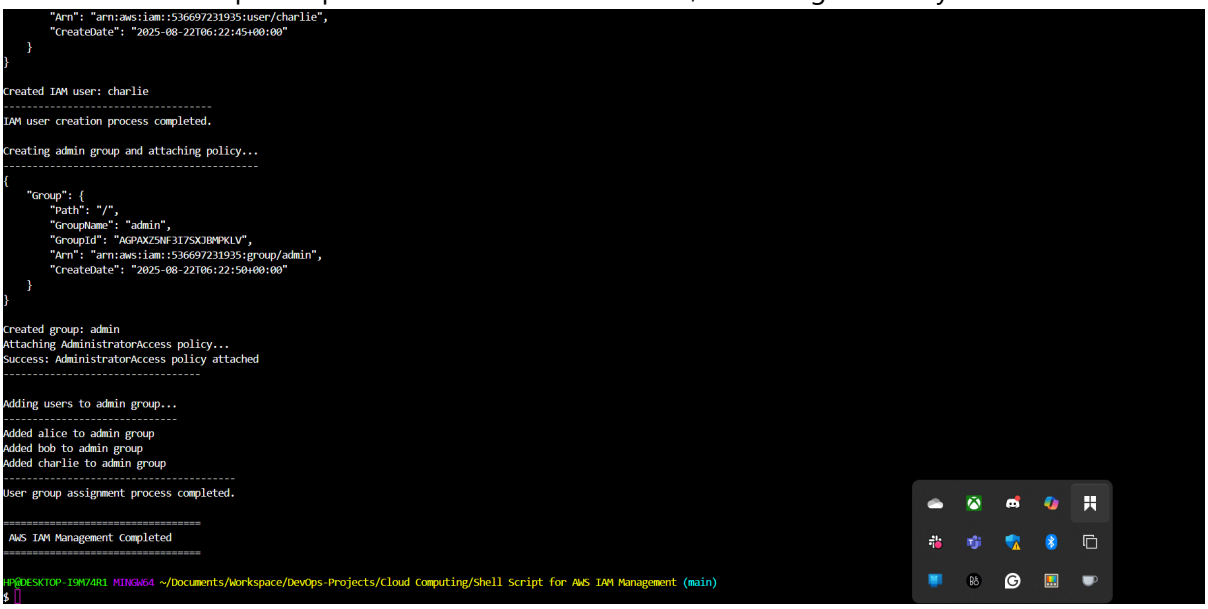
## 3. Script Execution

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.

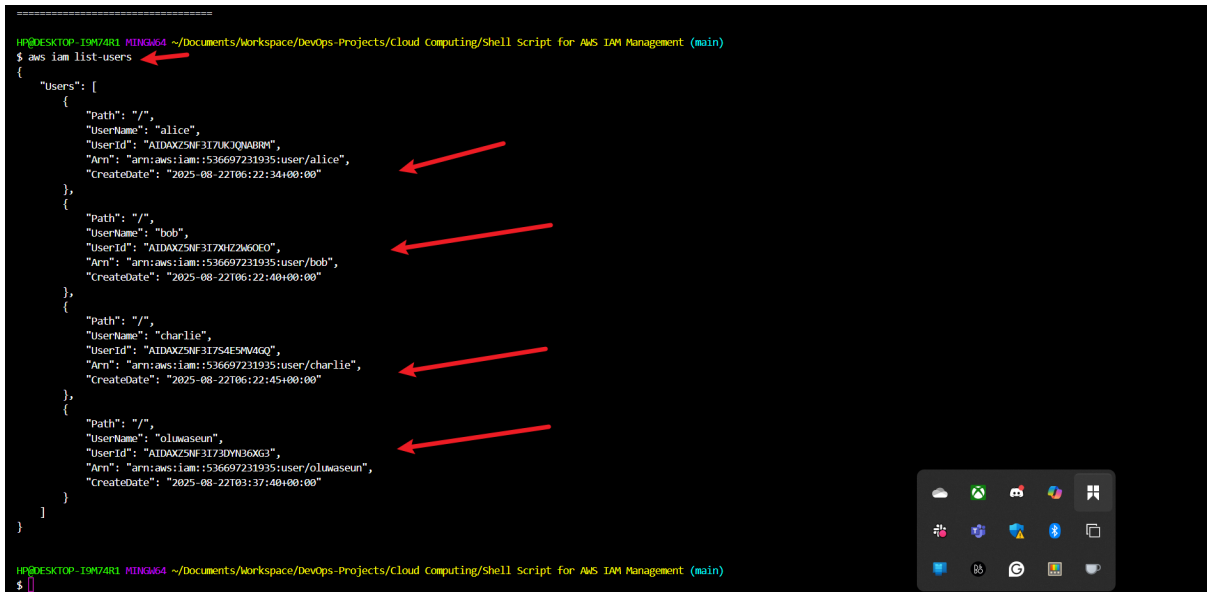
## Steps

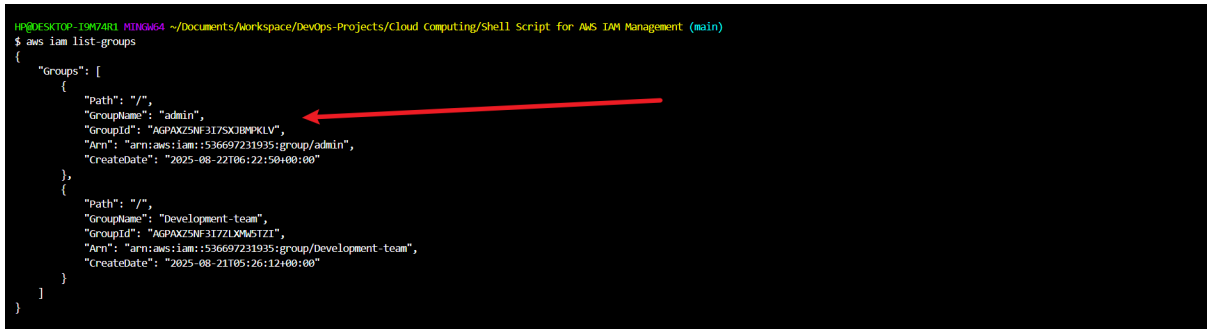
33. **Make Script Executable**:
    - Ran `chmod +x aws-iam-manager.sh` to make the script executable.
    - 

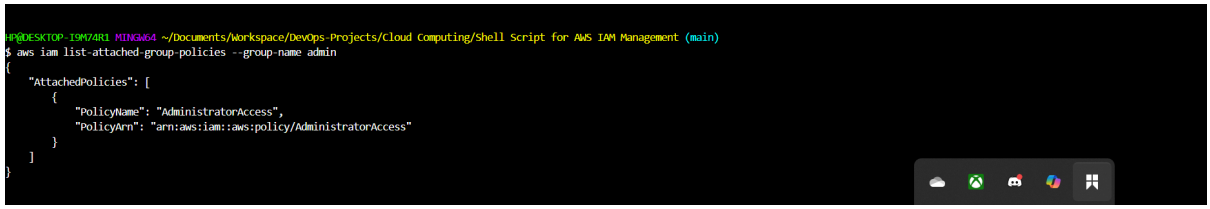
34. **Run the Script**:
    - Executed the script with `./aws-iam-manager.sh`, completing all tasks successfully.
    - 

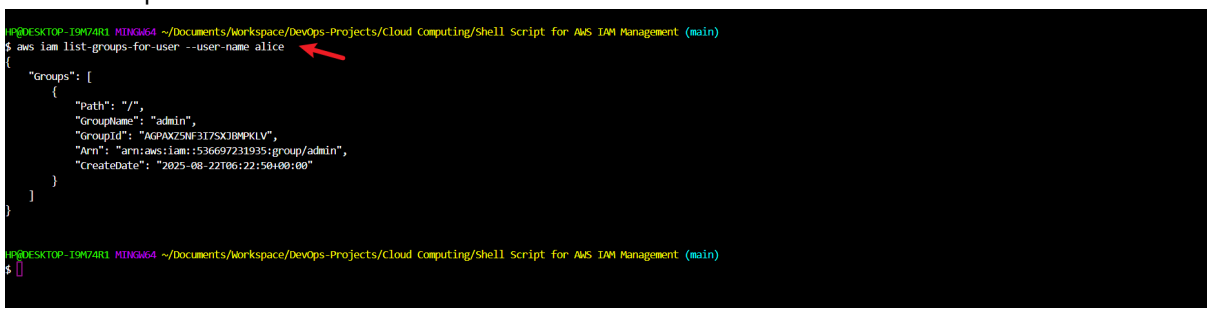
35. **Confirm Execution Time**:
    - Noted that the script completed in less than 30 seconds, indicating efficiency.
    - 

36. **Verify Users via CLI**:
    - Ran `aws iam list-users` to confirm the creation of users `alice`, `bob`, and `charlie`.

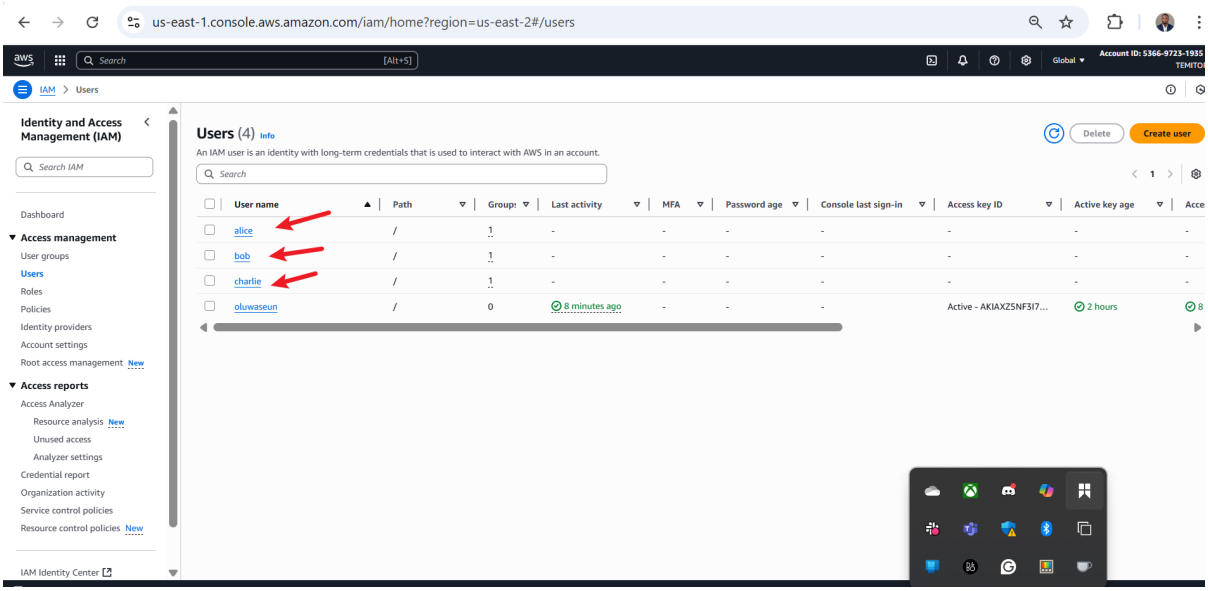
- 


37. **Verify Group Creation via CLI**:
    - Ran `aws iam list-groups` to confirm the "admin" group was created.
    - 

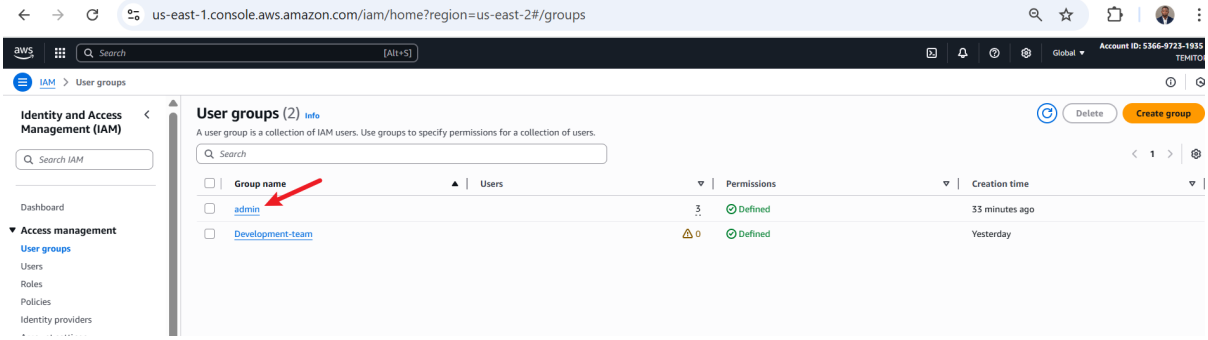

38. **Verify Policy Attachment via CLI**:
    - Ran `aws iam list-attached-group-policies --group-name admin` to confirm the `AdministratorAccess` policy was attached.
    - 


39. **Verify User Group Membership via CLI**:
    - Ran `aws iam list-groups-for-user --user-name alice` to confirm user group membership.
    - 

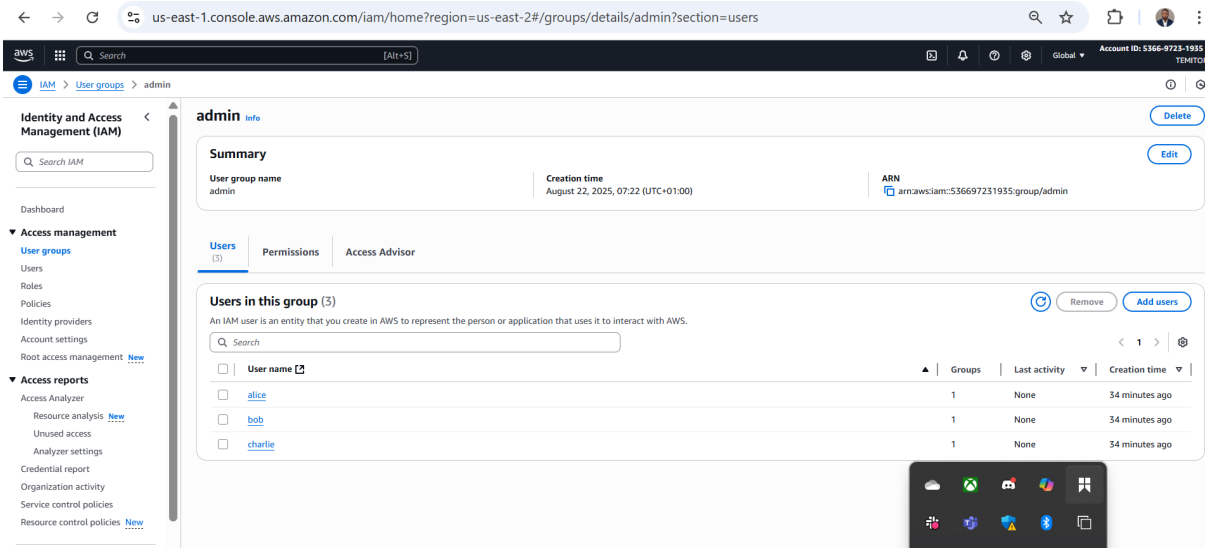

40. **Verify Users via AWS Console**:
    - Navigated to the IAM user list in the AWS Console to visually confirm the created users (`alice`, `bob`, `charlie`).



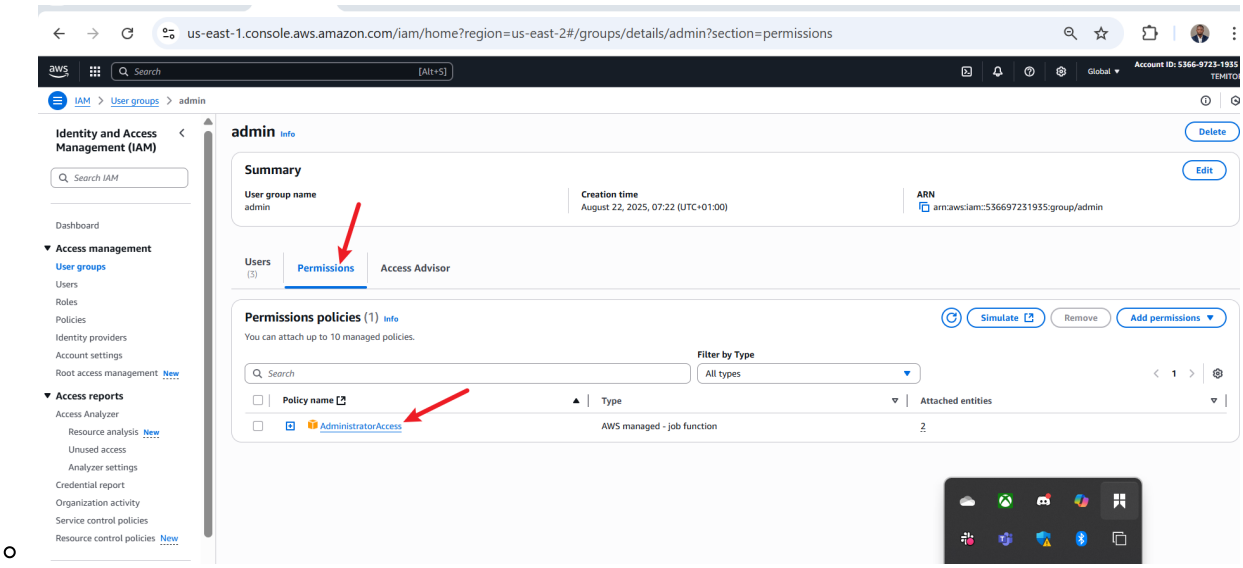

41. **Verify Group via AWS Console**:
    - Navigated to the IAM group list in the AWS Console to confirm the "admin" group.




42. **Verify Group Membership via AWS Console**:
    - Clicked on the "admin" group in the AWS Console to confirm user membership (`alice`, `bob`, `charlie`).




43. **Verify Policy via AWS Console**:
    - Checked the "admin" group permissions to verify the `AdministratorAccess` policy.




## 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 |
| --- | --- |
| `aws iam create-user --user-name "$user"` | Creates an IAM user. |
| `aws iam get-user --user-name "$user"` | Checks if a user exists. |
| `aws iam create-group --group-name admin` | Creates the "admin" group. |

| Command | Purpose |
| --- | --- |
| `aws iam attach-group-policy --group-name admin --policy-arn arn:aws:iam::aws:policy/AdministratorAccess` | Attaches the admin policy. |
| `aws iam add-user-to-group --user-name "$user" --group-name admin` | 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.