

# Mastering Environment Variables and Infrastructure Environments in Cloud Scripting

---

(Two-Phase Implementation: Beginner → Advanced)

**Author:** Oluwaseun Osunsola

**Environment and Tools:** AWS, Ubuntu VM, Nano

**Project Link:** <https://github.com/Oluwaseunoa/DevOps-Projects/tree/main>

## Introduction

In modern software development and cloud operations, the ability to **dynamically configure applications across multiple environments** is a foundational skill. This mini project, titled "**Mastering Environment Variables and Infrastructure Environments in Cloud Scripting**", explores the **critical distinction** between **infrastructure environments** (distinct deployment stages such as local, testing, and production) and **environment variables** (dynamic key-value pairs used to configure behavior without code changes).

Using a **real-world FinTech product scenario**, this project demonstrates how a single Bash script — `aws_cloud_manager.sh` — can securely and flexibly manage AWS resources across **three isolated environments**:

- **Local (Development):** VMware Workstation + Ubuntu
- **Testing (AWS Account 1):** Simulated pre-production
- **Production (AWS Account 2):** Live customer-facing deployment

The implementation progresses in **two phases**:

1. **Beginner Phase:** Focus on scripting fundamentals — conditionals, `export`, hard-coding pitfalls, and positional parameters
2. **Advanced Phase:** Full AWS integration using IAM users, CLI profiles, and real EC2 provisioning

Every step is **visually documented** with descriptive screenshots, ensuring **reproducibility, clarity, and alignment with DevOps best practices**.

---

## Definition of Terms

Term	Definition
<b>Infrastructure Environment</b>	A fully isolated deployment stage (e.g., local, testing, production) with dedicated compute, network, and data resources to prevent interference and ensure safe progression through the software lifecycle.
<b>Environment Variable</b>	A named value (e.g., <code>DB_URL=localhost</code> ) injected into a process at runtime to control configuration. Enables the same code to behave differently across environments.
<b>Positional Parameter</b>	Command-line arguments passed to a script ( <code>\$1, \$2</code> , etc.) that provide input at execution time, replacing hard-coded values for greater flexibility.

Term	Definition
<b>Hard-Coding</b>	Embedding fixed values directly in code (e.g., <code>ENVIRONMENT="testing"</code> ). An <b>anti-pattern</b> that reduces reusability and increases maintenance risk.
<b>AWS CLI Profile</b>	A named configuration in <code>~/.aws/credentials</code> and <code>~/.aws/config</code> that stores access keys, region, and output format for secure, account-specific operations.
<b>IAM User</b>	An AWS identity with granular permissions. Used to generate access keys for programmatic access (e.g., via AWS CLI) without using root credentials.
<b>Exit Code</b>	An integer ( <code>0</code> = success, <code>1+</code> = error) returned by a script to indicate execution outcome. Enables robust error handling and automation.
<b>Least Privilege</b>	Security principle granting only the permissions required for a task (e.g., <code>AmazonEC2FullAccess</code> instead of admin rights).

## Objectives

This project was designed to achieve the following **learning and technical objectives**:

1. **Clarify the difference** between infrastructure environments and environment variables
2. **Demonstrate progression** from hard-coded anti-patterns to dynamic configuration using `export` and positional parameters
3. **Implement robust input validation** and error handling with meaningful exit codes
4. **Create secure IAM users** with least-privilege policies in **two separate AWS accounts**
5. **Configure AWS CLI profiles** (`testing`, `production`) for isolated, secure access
6. **Develop a modular Bash script** that:
  - Accepts `<environment> <number_of_instances>`
  - Validates arguments
  - Checks AWS CLI and profile availability
  - Provisions real EC2 instances using `aws ec2 run-instances --profile`
  - Simulates operations locally via log file creation
7. **Verify all operations** in the AWS Management Console
8. **Apply DevOps principles**: reusability, security, automation, documentation, and incremental development

---

## Project Overview

This comprehensive mini project demonstrates the **evolution from beginner-level environment variable handling to advanced AWS multi-account automation**. The implementation progresses through two distinct phases:

### Phase 1: Beginner (No Users - Basic Scripting)

- Basic environment variable concepts
- Positional parameters and validation
- Local testing with mock operations
- Script structure and error handling fundamentals

## Phase 2: Advanced (IAM Users + AWS Integration)

- Secure IAM user creation with least-privilege policies
- AWS CLI profile configuration for multi-account management
- Real EC2 instance provisioning across testing/production
- Environment-specific resource configurations

The final script `aws_cloud_manager.sh` accepts two arguments: `<environment>` and `<number_of_instances>`, supporting local log file creation or AWS EC2 provisioning via dedicated profiles.

---

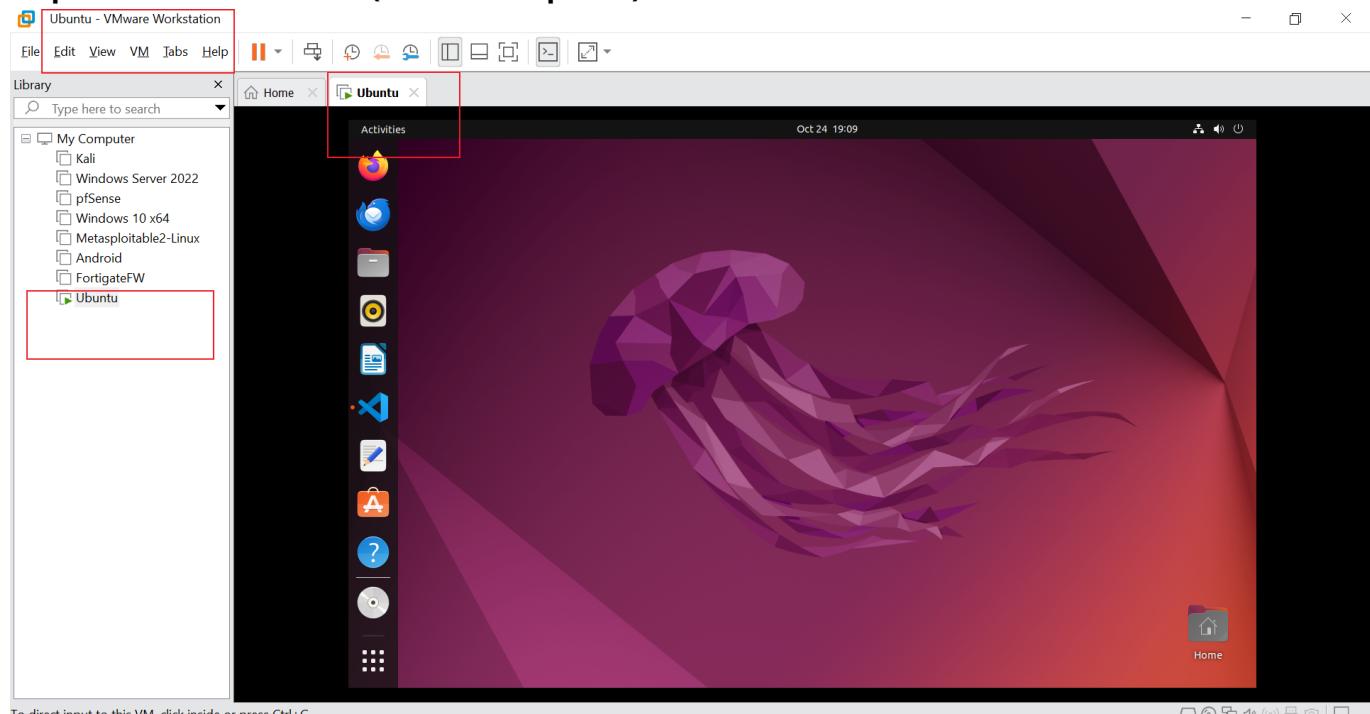
## Phase 1: Beginner Implementation (No AWS Users Required)

This phase focuses on **scripting fundamentals** using **environment variables**, **conditionals**, **positional parameters**, and **input validation** — all without requiring AWS credentials or IAM users.

We begin by setting up isolated **infrastructure environments** and evolve a simple script from **static logic** to **dynamic, reusable configuration**.

### Section 1: Infrastructure Environments Setup

#### Step 1: Launch Ubuntu VM (Local Development)



*Ubuntu 22.04 VM — our development sandbox*

## Step 2: Launch Testing EC2 (AWS Account 1)

The screenshot shows the AWS EC2 Instances page. The left sidebar is collapsed. The main area displays a table titled 'Instances (1/1) Info' with one row. The row contains a checkbox checked for the 'Testing Server' instance, which is highlighted with a red arrow. The instance details are as follows:

- Instance ID:** i-0996004ee9624f570
- Instance state:** Running
- Instance type:** t2.micro
- Status check:** Initializing
- Public IPv4 DNS:** us-east-2a
- Public IPv4 address:** (link to open address)
- Private IP4 addresses:** (link to view)
- Public DNS:** -

*Isolated testing environment — no customer impact*

**Note:** The existing EC2 server will be deleted and a new one will be created by the automation script.

## Step 3: Launch Production EC2 (AWS Account 2)

The screenshot shows the AWS EC2 Instances page. The left sidebar is collapsed. The main area displays a table titled 'Instances (1/1) Info' with one row. The row contains a checkbox checked for the 'Production Server' instance, which is highlighted with a red arrow. The instance details are as follows:

- Instance ID:** i-0996004ee9624f570
- Instance state:** Running
- Instance type:** t2.micro
- Status check:** Initializing
- Public IPv4 DNS:** us-east-2a
- Public IPv4 address:** (link to open address)
- Private IP4 addresses:** (link to view)
- Public DNS:** -

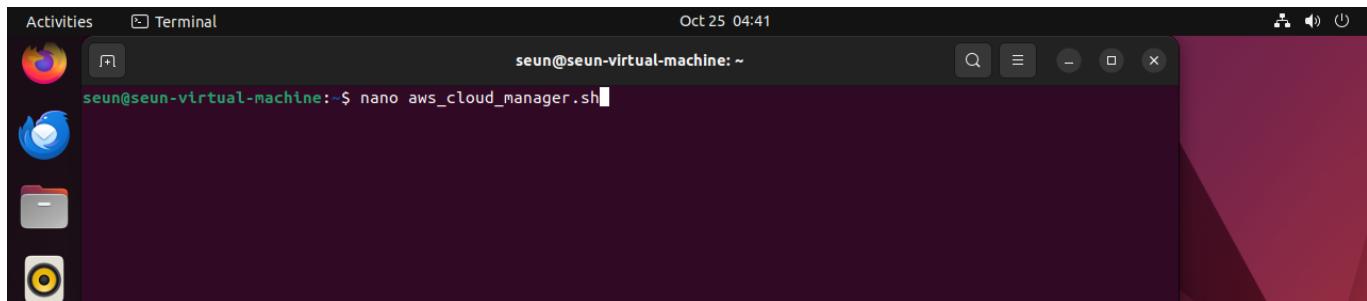
*Live environment — separate account for security* **Note:** The existing EC2 server will be deleted and a new one will be created by the automation script.

## 1.2 Creating the Initial Script

### 2.1 Script Creation

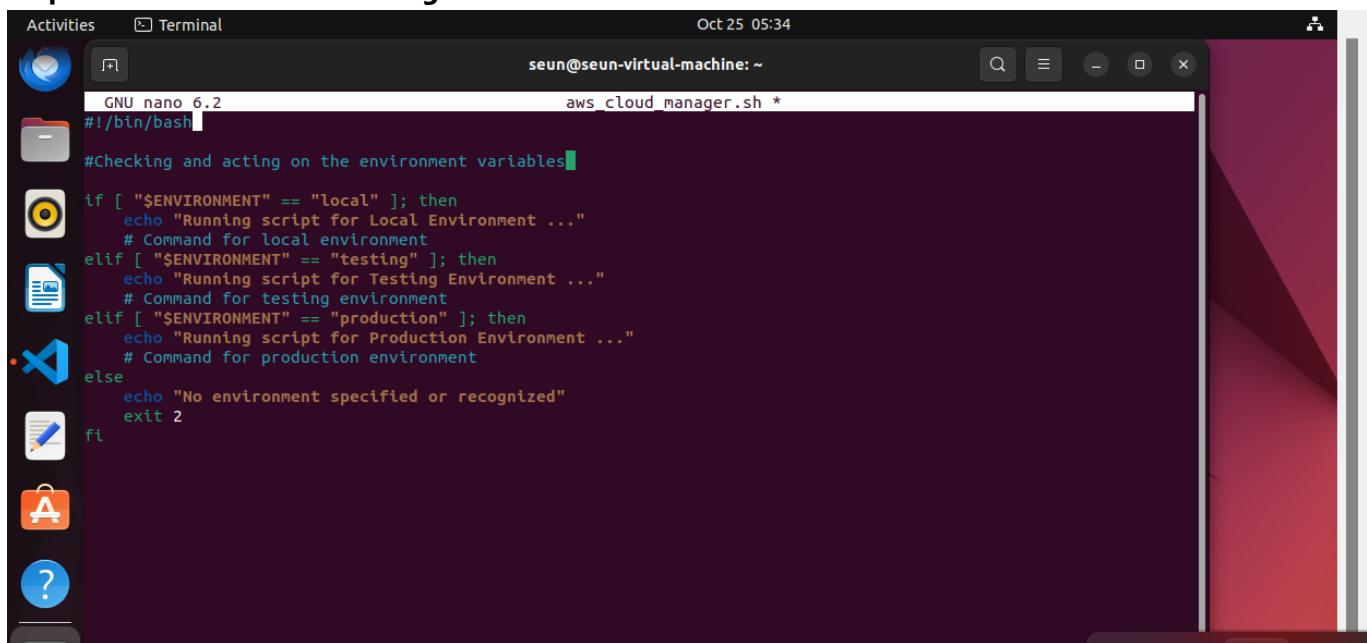
#### Step 1: Open **nano** to create the script

```
nano aws_cloud_manager.sh
```



```
Activities Terminal Oct 25 04:41
seun@seun-virtual-machine:~$ nano aws_cloud_manager.sh
```

## Step 2: Add basic conditional logic



```
Activities Terminal Oct 25 05:34
GNU nano 6.2          aws cloud manager.sh *
#!/bin/bash
#Checking and acting on the environment variables
if [ "$ENVIRONMENT" == "local" ]; then
    echo "Running script for Local Environment ..."
    # Command for local environment
elif [ "$ENVIRONMENT" == "testing" ]; then
    echo "Running script for Testing Environment ..."
    # Command for testing environment
elif [ "$ENVIRONMENT" == "production" ]; then
    echo "Running script for Production Environment ..."
    # Command for production environment
else
    echo "No environment specified or recognized"
    exit 2
fi
```

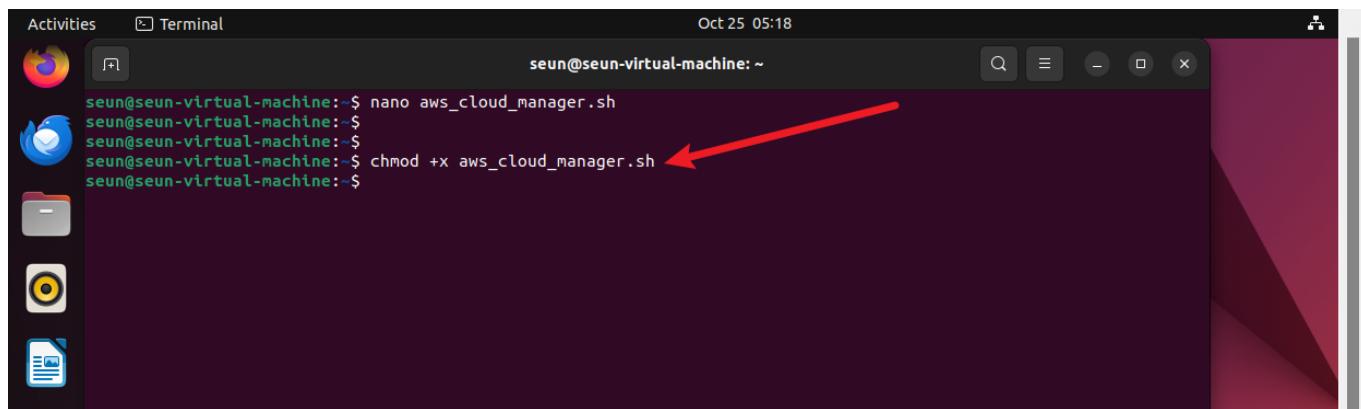
```
#!/bin/bash
# Checking and acting on the environment variable
if [ "$ENVIRONMENT" == "local" ]; then
    echo "Running script for Local Environment..."
elif [ "$ENVIRONMENT" == "testing" ]; then
    echo "Running script for Testing Environment..."
elif [ "$ENVIRONMENT" == "production" ]; then
    echo "Running script for Production Environment..."
else
    echo "No environment specified or recognized."
    exit 2
fi
```

**Note:** This version relies on the **global environment variable \$ENVIRONMENT**.

## 1.3 Making the Script Executable

**Step:** Grant execution permissions

```
sudo chmod +x aws_cloud_manager.sh
```

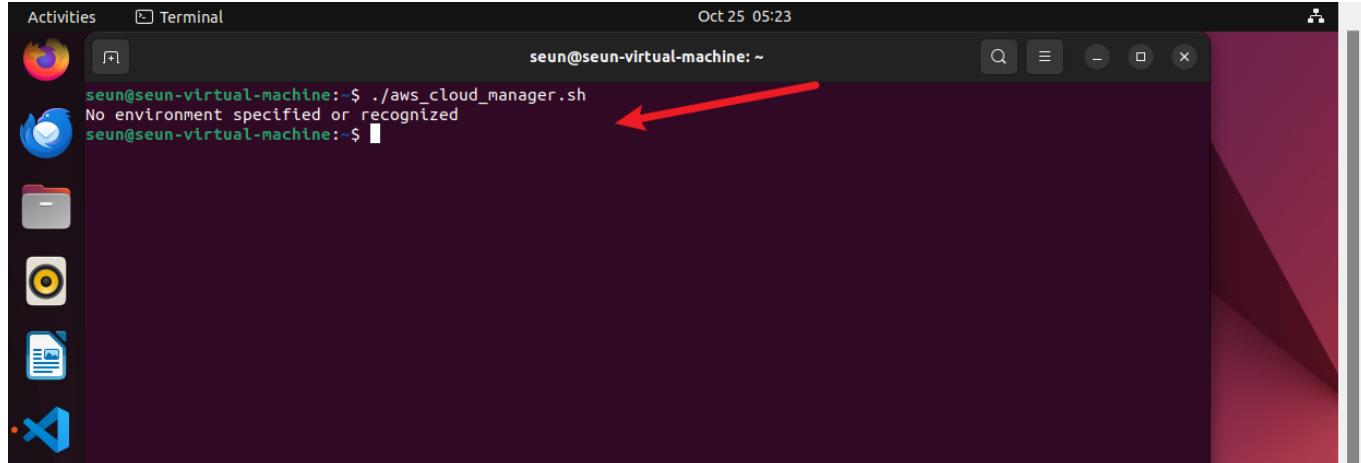


```
Activities Terminal seun@seun-virtual-machine: ~ Oct 25 05:18
seun@seun-virtual-machine:~$ nano aws_cloud_manager.sh
seun@seun-virtual-machine:~$ 
seun@seun-virtual-machine:~$ chmod +x aws_cloud_manager.sh ↗
seun@seun-virtual-machine:~$
```

## 1.4 Testing with Environment Variables

### 1.4.1 Run Without Setting ENVIRONMENT

```
./aws_cloud_manager.sh
```



```
Activities Terminal seun@seun-virtual-machine: ~ Oct 25 05:23
seun@seun-virtual-machine:~$ ./aws_cloud_manager.sh ↗
No environment specified or recognized
seun@seun-virtual-machine:~$
```

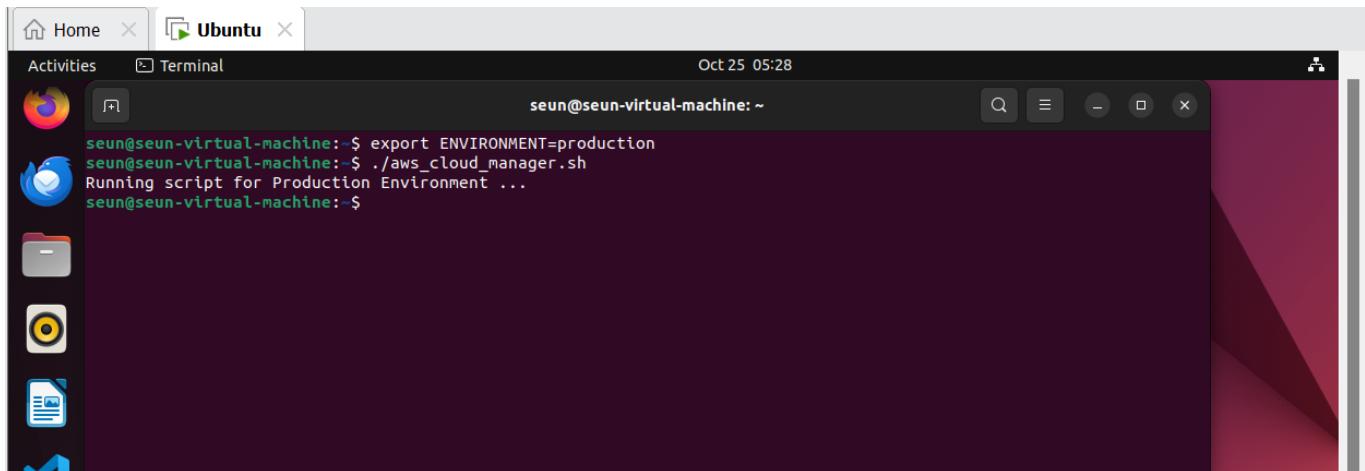
#### Output:

```
No environment specified or recognized.
```

**Exit Code:** 2 → Script terminates in `else` block

### 1.4.2 Use `export` to Set Environment Dynamically

```
export ENVIRONMENT=production
./aws_cloud_manager.sh
```

A screenshot of an Ubuntu desktop environment. A terminal window titled "Ubuntu" is open, showing the command line interface. The terminal window has a dark background with white text. At the top of the terminal, it says "seun@seun-virtual-machine: ~". Below that, the user has run the command "export ENVIRONMENT=production" followed by "./aws\_cloud\_manager.sh". The script outputs "Running script for Production Environment ...". The terminal window has a standard Linux-style title bar with icons for search, minimize, maximize, and close.

## Output:

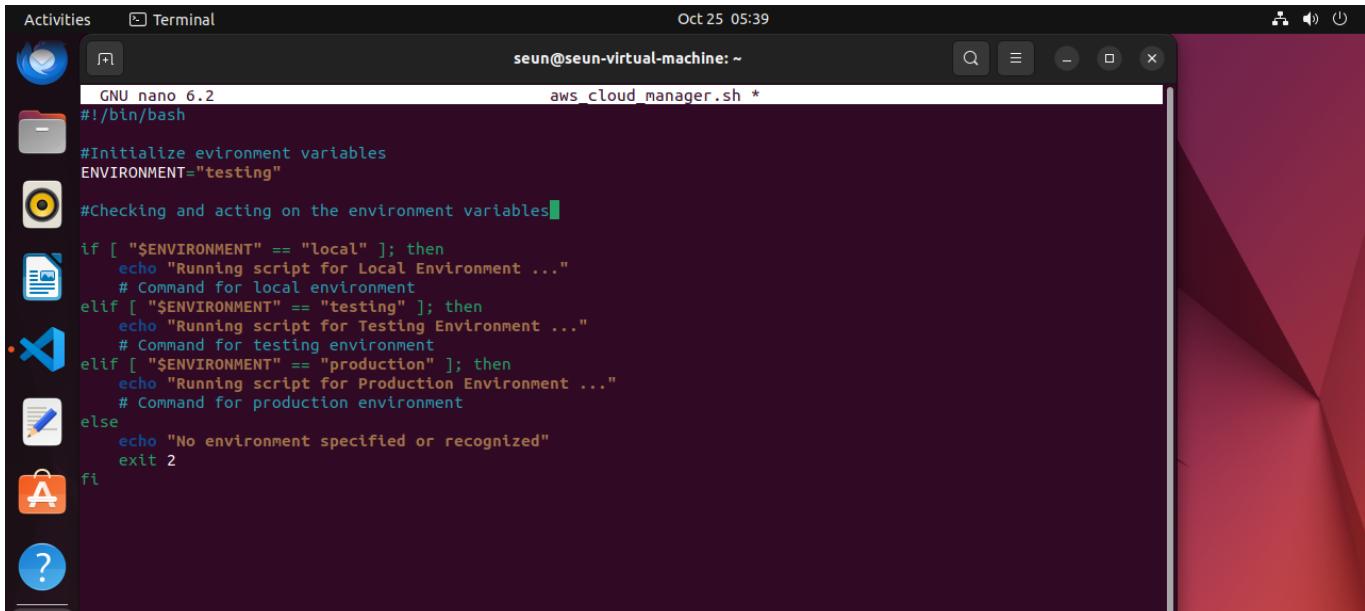
```
Running script for Production Environment...
```

### Key Insight:

`export` makes the variable available to child processes (like our script). This enables **dynamic configuration** without code changes.

## 1.5 Hard-Coded Version (Educational Anti-Pattern)

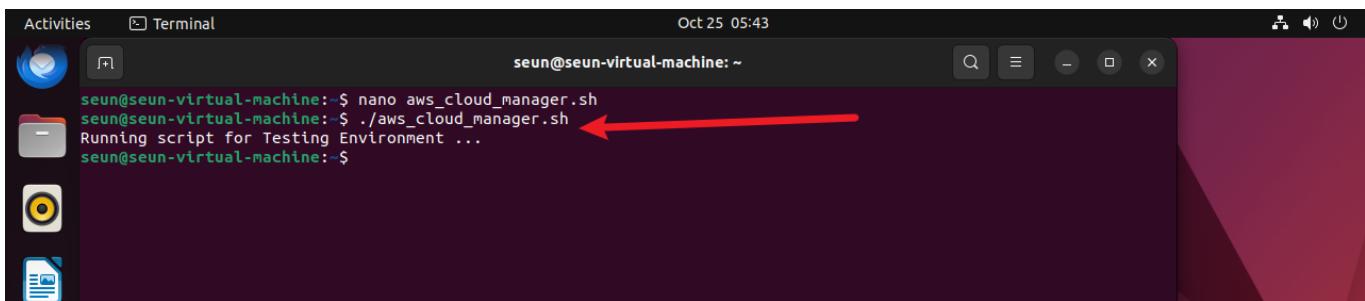
### Step: Modify script to hard-code the environment

A screenshot of an Ubuntu desktop environment. A terminal window titled "Terminal" is open, showing the command line interface. The terminal window has a dark background with white text. At the top of the terminal, it says "seun@seun-virtual-machine: ~". Below that, the user has run the command "GNU nano 6.2 aws\_cloud\_manager.sh \*". The terminal shows the contents of a file named "aws\_cloud\_manager.sh". The script starts with "#!/bin/bash" and "# Initialize environment variables". It then sets the variable "ENVIRONMENT" to "testing". The script continues with "# Checking and acting on the environment variables". It contains a series of if-then-else statements to check the value of "\$ENVIRONMENT". If it's "local", it echoes "Running script for Local Environment ...". If it's "testing", it echoes "Running script for Testing Environment ...". If it's "production", it echoes "Running script for Production Environment ...". If none of these conditions are met, it prints "No environment specified or recognized" and exits with status 2. The terminal window has a standard Linux-style title bar with icons for search, minimize, maximize, and close.

```
ENVIRONMENT="testing" # Hard-coded – not flexible
```

### Execute:

```
./aws_cloud_manager.sh
```



```
Activities Terminal Oct 25 05:43
seun@seun-virtual-machine:~$ nano aws_cloud_manager.sh
seun@seun-virtual-machine:~$ ./aws_cloud_manager.sh
Running script for Testing Environment ...
seun@seun-virtual-machine:~$
```

## Output:

```
Running script for Testing Environment...
```

### Problem:

- No flexibility
- Requires code edit per environment
- Violates **reusability** and **separation of config from code**

### Best Practice: Use **environment variables** or **command-line arguments**

## 1.6 Introducing Positional Parameters

### Goal: Replace hard-coding with **runtime arguments**

```
./aws_cloud_manager.sh testing 5
```

### 1.6.1 Add Argument Count Validation

```

Activities Terminal Oct 25 11:16
seun@seun-virtual-machine: ~ aws_cloud_manager.sh *

GNU nano 6.2
#!/bin/bash

# Checking the number of arguments
if [ "$#" -ne 1 ]; then
    echo "Usage: $0 <environment>"
    exit 1
fi

# Assessing the first argument
ENVIRONMENT=$1
NUMBER_OF_INSTANCES=2

# Accessing the first argument
if [ "$ENVIRONMENT" == "local" ]; then
    echo "Running script for Local Environment ..."
# Command for local environment
elif [ "$ENVIRONMENT" == "testing" ]; then
    echo "Running script for Testing Environment ..."
# Command for testing environment
elif [ "$ENVIRONMENT" == "production" ]; then
    echo "Running script for Production Environment ..."
# Command for production environment
else
    echo "Invalid environment specified. Please use 'local', 'testing', or 'production'. "
    exit 2
fi

```

```

if [ "$#" -ne 1 ]; then
    echo "Usage: $0 <environment>"
    exit 1
fi

ENVIRONMENT=$1

```

**\$#** = number of arguments  
**\$0** = script name  
**\$1** = first argument

---

## 1.6.2 Test Error Cases

### No arguments:

```
./aws_cloud_manager.sh
```

```

Activities Terminal Oct 25 06:02
seun@seun-virtual-machine: ~ ./aws_cloud_manager.sh
Usage: ./aws_cloud_manager.sh <environment>
seun@seun-virtual-machine: ~

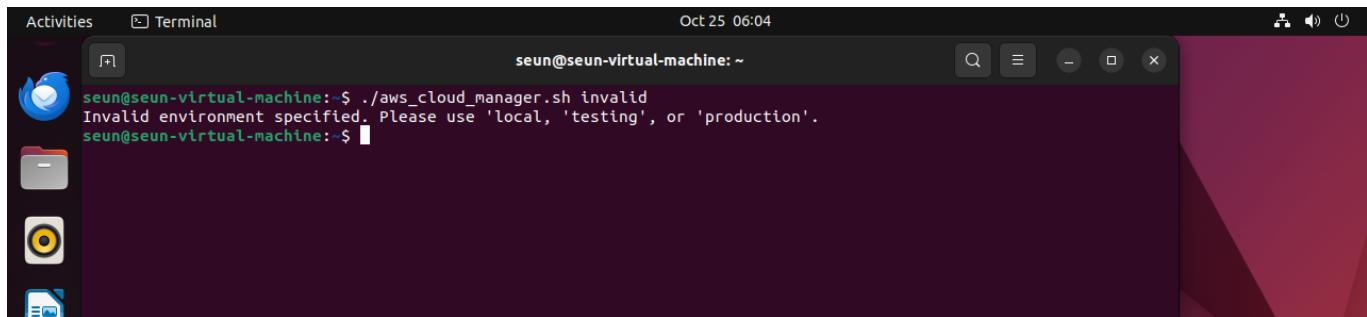
```

**Output:** Usage: ./aws\_cloud\_manager.sh <environment>

---

### Invalid environment:

```
./aws_cloud_manager.sh invalid
```

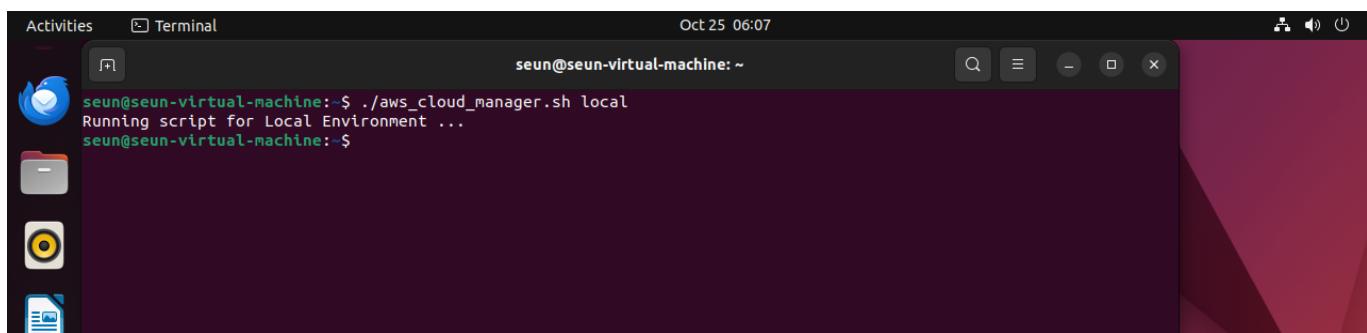


A screenshot of a Linux desktop environment showing a terminal window. The terminal title is "Terminal" and the date and time are "Oct 25 06:04". The command entered is "../aws\_cloud\_manager.sh invalid". The output is: "Invalid environment specified. Please use 'local', 'testing', or 'production'." The terminal window has a dark theme with a red gradient background.

**Output:** Invalid environment specified. Please use 'local', 'testing', or 'production'

### Valid environment (local):

```
./aws_cloud_manager.sh local
```

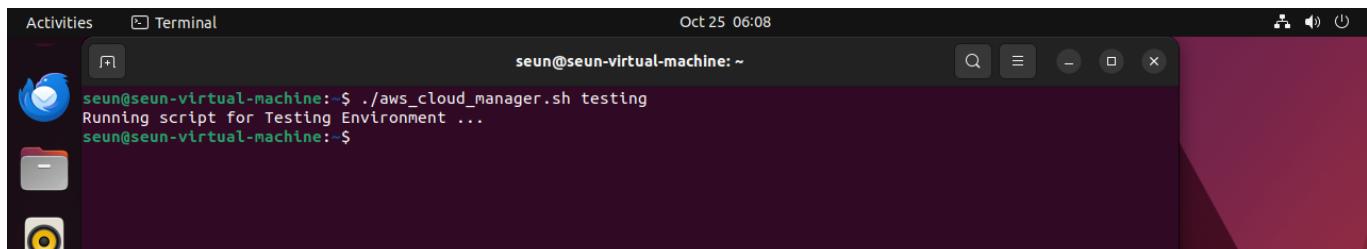


A screenshot of a Linux desktop environment showing a terminal window. The terminal title is "Terminal" and the date and time are "Oct 25 06:07". The command entered is "../aws\_cloud\_manager.sh local". The output is: "Running script for Local Environment ...". The terminal window has a dark theme with a red gradient background.

**Output:** Running script for Local Environment...

### Valid environment (testing):

```
./aws_cloud_manager.sh testing
```

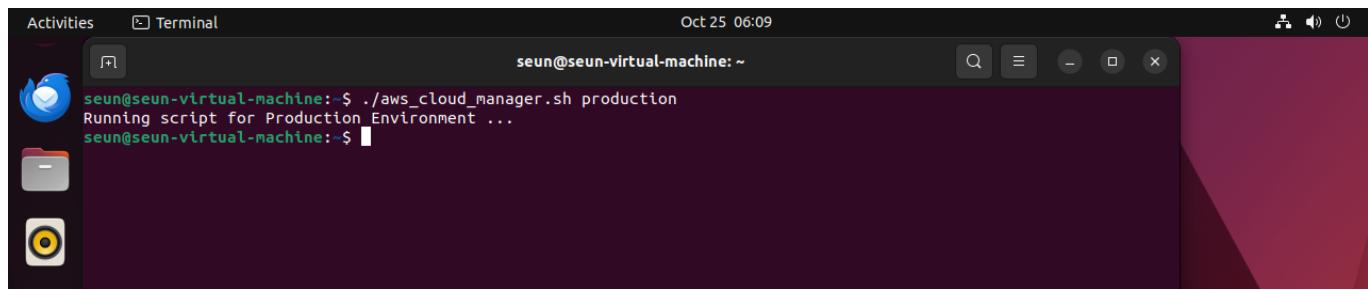


A screenshot of a Linux desktop environment showing a terminal window. The terminal title is "Terminal" and the date and time are "Oct 25 06:08". The command entered is "../aws\_cloud\_manager.sh testing". The output is: "Running script for Testing Environment ...". The terminal window has a dark theme with a red gradient background.

**Output:** Running script for Testing Environment...

### Valid environment (production):

```
./aws_cloud_manager.sh production
```



The screenshot shows a terminal window titled "Terminal" with the command "seun@seun-virtual-machine:~\$ ./aws\_cloud\_manager.sh production" entered. The output is "Running script for Production Environment ...". The terminal is part of a desktop environment with icons for Activities, Home, and Applications visible.

**Output:** Running script for Testing Environment...

### 1.6.3 Final Beginner Script (With Validation)

```
#!/bin/bash

# Validate number of arguments
if [ "$#" -ne 1 ]; then
    echo "Usage: $0 <environment>"
    exit 1
fi

ENVIRONMENT=$1

# Act based on environment
if [ "$ENVIRONMENT" == "local" ]; then
    echo "Running script for Local Environment..."
elif [ "$ENVIRONMENT" == "testing" ]; then
    echo "Running script for Testing Environment..."
elif [ "$ENVIRONMENT" == "production" ]; then
    echo "Running script for Production Environment..."
else
    echo "Invalid environment specified. Please use 'local', 'testing', or
'production'."
    exit 2
fi
```

## Phase 1 Summary

Concept	Learned
<b>Infrastructure Environments</b>	Isolated stages (local, testing, production)
<b>Environment Variables</b>	Dynamic config via <code>export</code>
<b>Hard-Coding</b>	Anti-pattern — reduces reusability
<b>Positional Parameters</b>	<code>\$1, \$#</code> for runtime input
<b>Input Validation</b>	Prevent bugs with argument checks

Concept	Learned
Exit Codes	<code>exit 1</code> (usage), <code>exit 2</code> (invalid input)

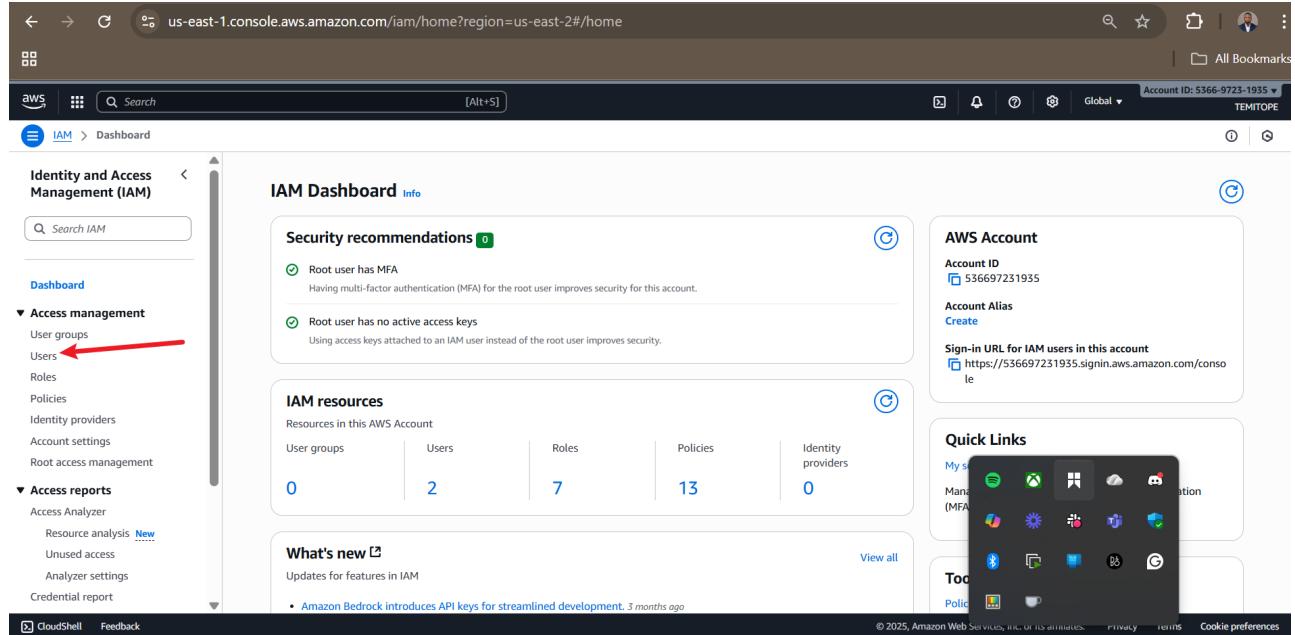
## Phase 2: Advanced Implementation (IAM Users + AWS Integration)

## Section 2: Secure AWS Account Setup

### **3.1 Testing Account IAM User Creation (AWS Account 1)**

## **Step 1: Navigate to IAM Dashboard and Click Users**

- Login to AWS Account 1 (Another account you have), navigate to IAM dashboard and click on users



## 2. Click Create User

- On Users page click on create users

The screenshot shows the AWS IAM 'Users' page. On the left, there's a navigation sidebar with 'Identity and Access Management (IAM)' selected. Under 'Access management', 'Users' is also selected. The main area displays a table of users with columns for User name, Path, Group, Last activity, MFA, Password age, Console last sign-in, and Access key ID. Two users are listed: 'oluwaeseun' (last active 3 days ago) and 'automation-testing-user' (last active 15 hours ago). In the top right corner of the table, there's a 'Create user' button, which is highlighted with a red arrow.

### 3. Enter User Name:

- Enter User Name and Click Next

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 heading 'Specify user details' and a 'User details' section. It includes a 'User name' input field containing 'automation-testing-user', a note about character restrictions, and a checkbox for 'Provide user access to the AWS Management Console - optional'. Below these is a note about generating access keys. At the bottom right, there are 'Cancel' and 'Next Step' buttons, with a red arrow pointing to the 'Next Step' button.

### 1. Attach Policies Directly

- Select "Attach Policies Directly", search and select AmazonEC2FullAccess

us-east-1.console.aws.amazon.com/iam/home?region=us-east-2#/users/details/automation-testing-user/add-permissions

**Add permissions**

Step 1: Add permissions Step 2: Review

**Permissions options**

- Add user to group
- Copy permissions
- Attach policies directly** (selected)

**Permissions policies (2/1412)**

Filter by Type: All types, 1 match

Policy name	Type	Attached entities
AmazonEC2FullAccess	AWS managed	1

Cancel Next

- Also Search and select AmazonSSMManagedInstanceCore

us-east-1.console.aws.amazon.com/iam/home?region=us-east-2#/users/details/automation-testing-user/add-permissions

**Add permissions**

Step 1: Add permissions Step 2: Review

**Permissions options**

- Add user to group
- Copy permissions
- Attach policies directly** (selected)

**Permissions policies (2/1412)**

Filter by Type: All types, 1 match

Policy name	Type	Attached entities
AmazonSSMManagedInstanceCore	AWS managed	0

Cancel Next

## 2. Review & Create

- Review all the details set and click create user if everything is perfect.

**Review**

The following policies will be attached to this user. [Learn more](#)

**User details**

User name: automation-testing-user

**Permissions summary (2)**

Name	Type	Used as
AmazonEC2FullAccess	AWS managed	Permissions policy
AmazonSSMManagedInstanceCore	AWS managed	Permissions policy

## 3. User Created Successfully

- automation-testing-user created successfully click on automation-testing-user in the Users list.

The screenshot shows the AWS IAM 'Users' list. There are two users listed:

- automation-testing-user**: Created on October 25, 2025, at 08:33 (UTC+01:00). Last console sign-in was 3 days ago. Access key ID: AKIAJZ5NF317... (Active).
- oluwasueun**: Created on October 25, 2025, at 08:33 (UTC+01:00). Last console sign-in was 3 days ago. Access key ID: AKIAJZ5NF317... (Active).

## 1. Click create Access Key

- On automation-testing-user dashboard, click create access key.

The screenshot shows the 'automation-testing-user' details page under the 'Permissions' tab. The user has one permission policy attached. The 'Create access key' button is highlighted with a red arrow.

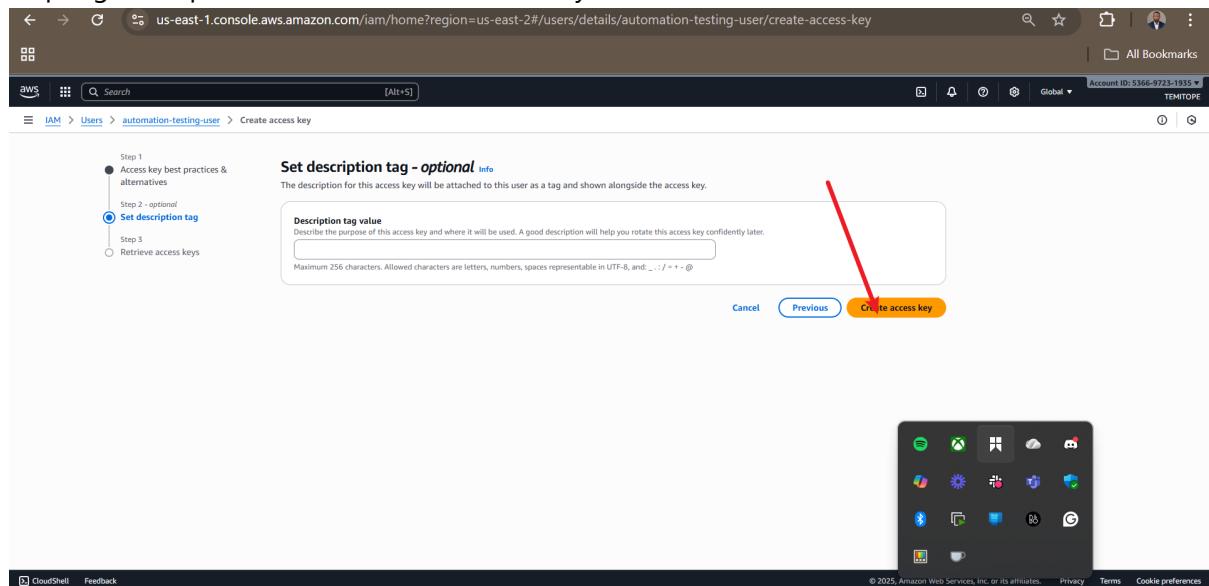
- Select Command Line Interface (CLI) and check the box to acknowledge understanding the recommendation and click next.

The screenshot shows the 'Access key best practices & alternatives' step of the 'Create access key' wizard. It lists several use cases for access keys:

- Command Line Interface (CLI)** (selected): You plan to use this access key to enable the AWS CLI to access your AWS account.
- Local code**: You plan to use this access key to enable application code in a local development environment to access your AWS account.
- Application running on an AWS compute service**: You plan to use this access key to enable application code running on an AWS compute service like Amazon EC2, Amazon Lambda, or AWS Lambda to access your AWS account.
- Third-party service**: You plan to use this access key to enable access for a third-party application or service that monitors or manages your AWS resources.
- Application running outside AWS**: You plan to use this access key to authenticate workloads running in your data center or other infrastructure outside of AWS that needs to access your AWS resources.
- Other**: Your use case is not listed here.

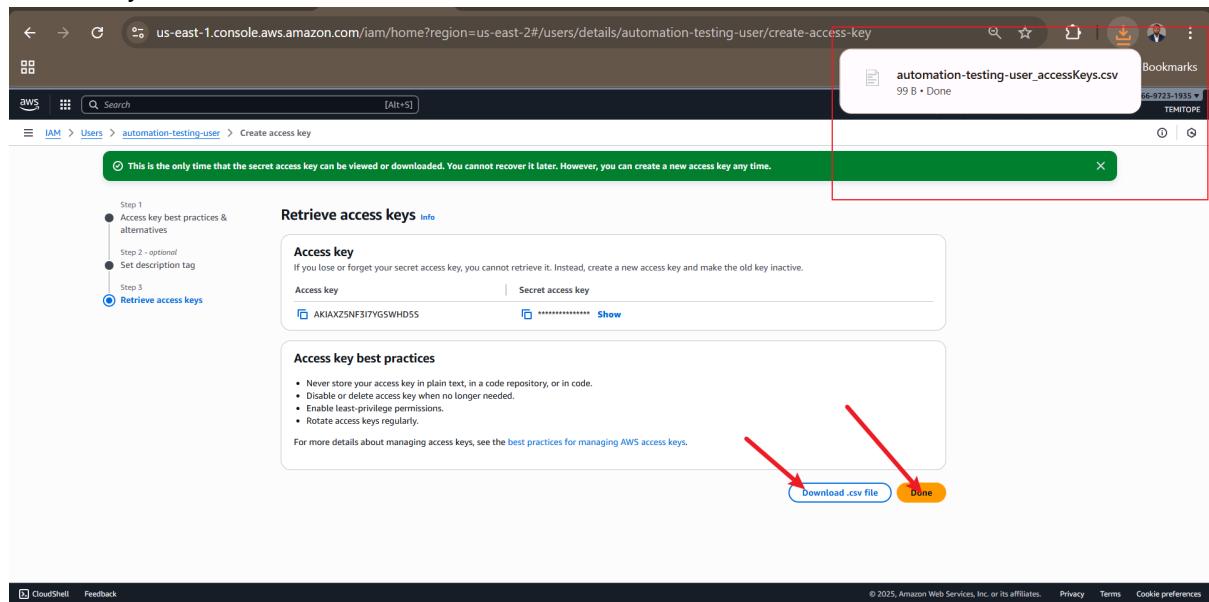
A yellow box highlights the 'AWS CLI (CloudShell)' alternative, which is recommended for running commands. A checkbox at the bottom left is checked, and a red arrow points to the 'Next Step' button at the bottom right.

- Skip tag description and click create access key.



## 2. Download Credentials

- Access key created now download and click, Done.



- Notice on the dashboard that access key is now created but not used.

**automation-testing-user**

**Summary**

ARN: arn:aws:iam::536697231935:user/automation-testing-user

Console access: Disabled

Created: October 25, 2025, 08:33 (UTC-01:00)

Last console sign-in:

**Access key 1**  
AKIAZ25NF3I7YGSWHDSS - Active  
Never used. Created today.

**Access key 2**  
Create access key

**Console sign-in**

Console sign-in link: https://S36697231935.sigin.aws.amazon.com/console

Console password: Not enabled

**Multi-factor authentication (MFA) (0)**

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

**Assign MFA device**

## 3.2 Production Account IAM User Creation (AWS Account 2)

Same process for **automation-production-user**:

- Switch to AWS Account 2 (Another account you have), navigate to IAM dashboard and click on users

**IAM Dashboard**

**Security recommendations**

- Root user has MFA
- Root user has no active access keys

**IAM resources**

User groups	Users	Roles	Policies	Identity providers
0	0	3	0	0

**What's new**

Updates for features in IAM

- Amazon Bedrock introduces API keys for streamlined development. 3 months ago

**AWS Account**

Account ID: 832959958705

Account Alias: Create

Sign-in URL for IAM users in this account: https://832959958705.sigin.aws.amazon.com/console

**Quick Links**

My security credentials, Policy simulator

**Tools**

Policy simulator

- On users dashboard, click on create user.

The screenshot shows the AWS IAM Users page. On the right side, there is a large orange button labeled "Create user". A red arrow points from the top right towards this button. The page title is "Users (0) Info". The left sidebar shows navigation options like "Identity and Access Management (IAM)", "Access management", and "Access reports". The main content area displays a table header for "User name", "Path", "Group", "Last activity", "MFA", "Password age", and "Console last sign-in". Below the table, it says "No resources to display". At the bottom right of the page, there is a "Create user" button.

- Name user automation-production-user and click next

The screenshot shows the "Specify user details" step of the IAM User creation wizard. The user name field contains "automation-production-user". A red arrow points from the bottom right towards the "Next" button. The page title is "Step 1 Specify user details". The left sidebar shows steps: Step 1 (selected), Step 2 (Set permissions), Step 3 (Review and create). The main content area includes a "User details" section with a "User name" input field and a note about valid characters (A-Z, a-z, 0-9, +, ., @, -, \_). It also includes an optional checkbox for "Provide user access to the AWS Management Console - optional" and a note about generating programmatic access keys. At the bottom right, there are "Cancel" and "Next" buttons.

- Select "Attach Policies Directly", search and select AmazonEC2FullAccess

**Add permissions**

Step 1: Add permissions

Step 2: Review

**Permissions options**

- Add user to group
- Copy permissions
- Attach policies directly

**Permissions policies (1/1399)**

Filter by Type: All types, 1 match

Policy name	Type	Attached entities
<input checked="" type="checkbox"/> AmazonEC2FullAccess	AWS managed	1

Cancel Next

- Also Search and select AmazonSSMManagedInstanceCore then click next

**Add permissions**

Step 1: Add permissions

Step 2: Review

**Permissions options**

- Add user to group
- Copy permissions
- Attach policies directly

**Permissions policies (2/1399)**

Filter by Type: All types, 1 match

Policy name	Type	Attached entities
<input checked="" type="checkbox"/> AmazonSSMManagedInstanceCore	AWS managed	0

Cancel Next

- Review all the details set and click create user if everything is perfect.

**Review**

The following policies will be attached to this user. [Learn more](#)

**User details**

User name: automation-production-user

**Permissions summary (2)**

Name	Type	Used as
AmazonEC2FullAccess	AWS managed	Permissions policy
AmazonSSMManagedInstanceCore	AWS managed	Permissions policy

Create user

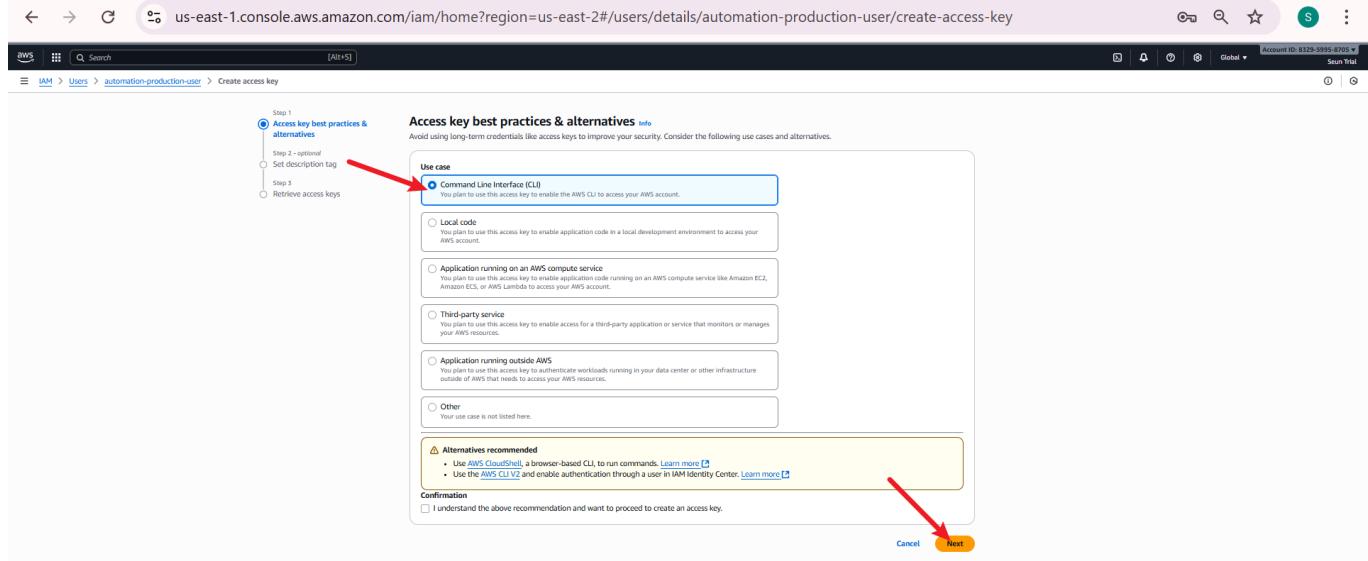
- automation-production-user created successfully click on automation-production-user in the Users list.

The screenshot shows the AWS IAM 'Users' list. A green success message box at the top states: 'User created successfully' and 'You can view and download the user's password and email instructions for signing in to the AWS Management Console.' Below this, the 'Users (1)' table lists one item: 'automation-production-user'. A red arrow points to the 'automation-production-user' link in the 'User name' column. The table includes columns for User name, Path, Group, Last activity, MFA, Password age, Console last sign-in, and Access key ID.

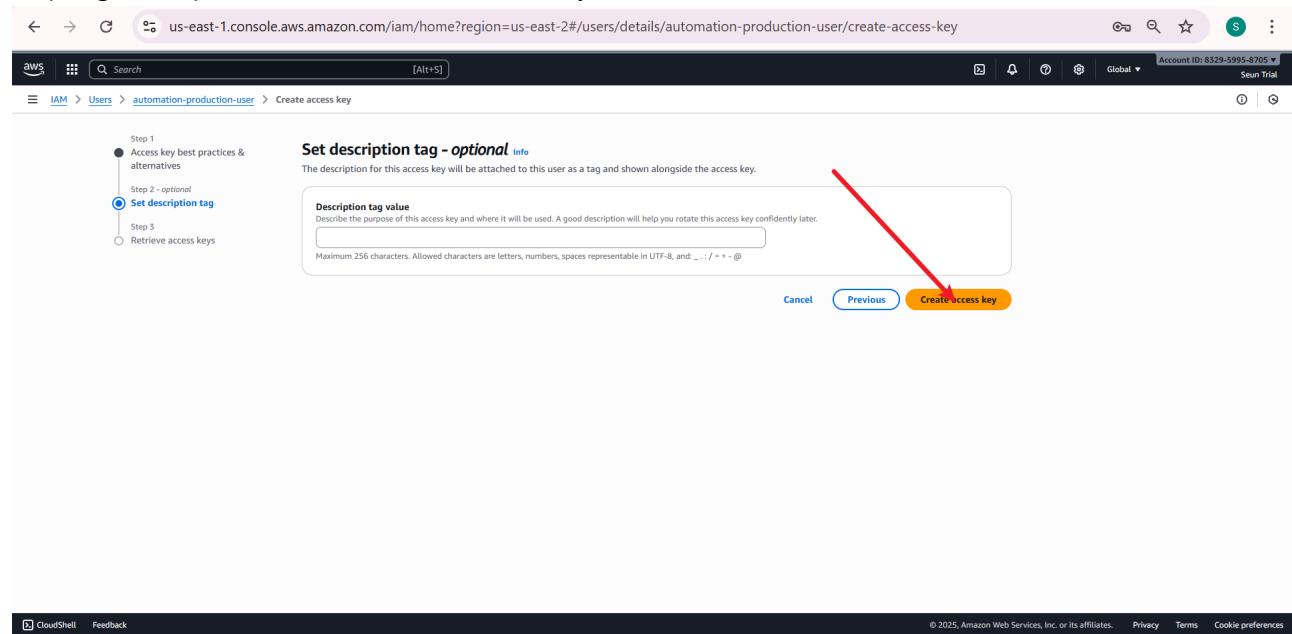
- On automation-production-user click create access key

The screenshot shows the 'automation-production-user' details page. In the 'Summary' section, there is a callout pointing to the 'Create access key' button, which is highlighted in blue. The 'Permissions' tab is selected. Other tabs include Groups, Tags, Security credentials, and Last Accessed.

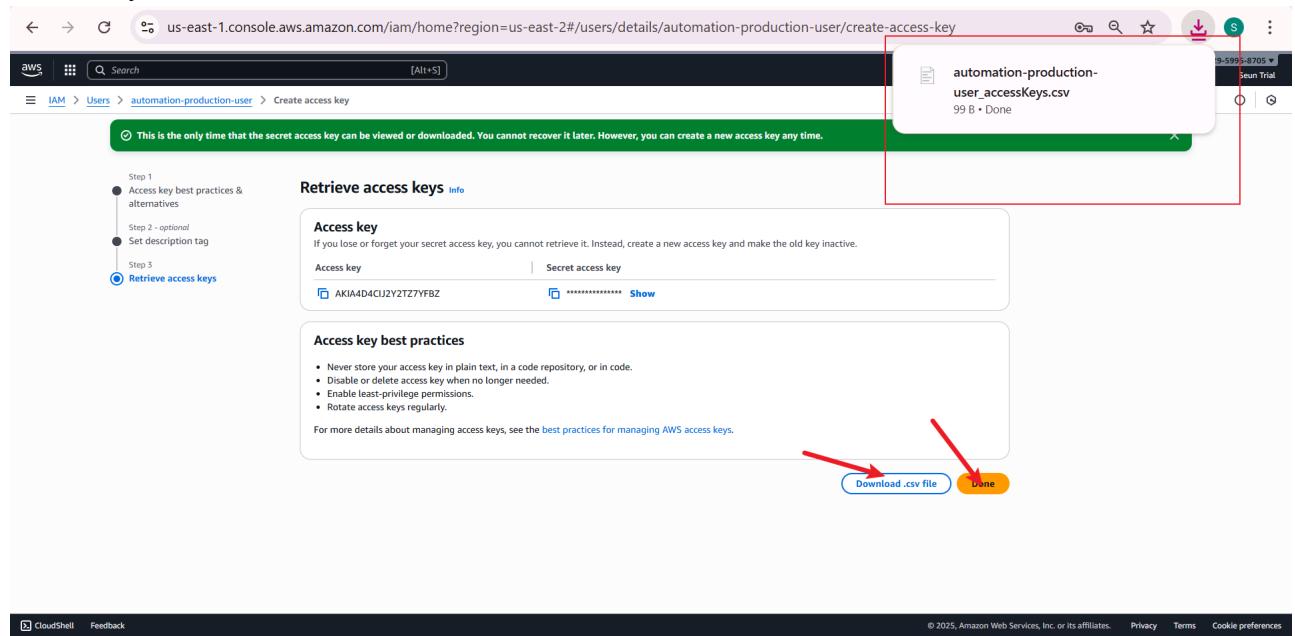
- Select Command Line Interface (CLI) and check the box to acknowledge understanding the recommendation and click next.



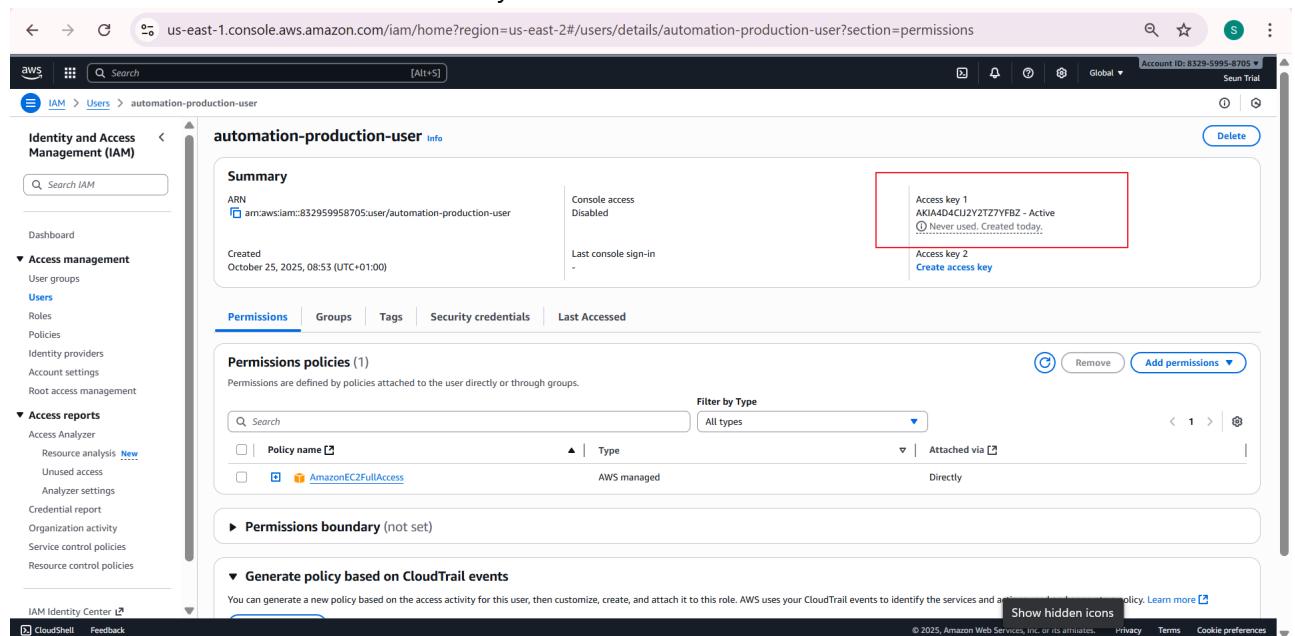
- Skip tag description and click create access key.



- Access key created now download and click, Done.



- Notice on the dashboard that access key is now created but not used.



### 3.3 AWS CLI Profile Configuration

#### Testing Profile:

```
aws configure --profile testing
```

- The response will be a prompt asking for the create Access key and Secret Key as well as default region and default format. Once everything

```
Activities Terminal Oct 25 10:47
seun@seun-virtual-machine: ~
seun@seun-virtual-machine: $ aws configure --profile testing
AWS Access Key ID [None]: AKIAZ5NF3I7YGSWHDSS
AWS Secret Access Key [None]: DrI0u92mLkkq3cesC4QzxXk4pfPyuHwY/Ebed0QQ
Default region name [None]: us-east-2
Default output format [None]: json
seun@seun-virtual-machine: $
```

## Production Profile:

```
aws configure --profile production
```

- The response will be a prompt asking for the create Access key and Secret Key as well as default region and default format. Once everything

```
Activities Terminal Oct 25 10:47
seun@seun-virtual-machine: ~
seun@seun-virtual-machine: $ aws configure --profile production
AWS Access Key ID [None]: AKIA4D4CIJ2Y2TZ7YFBZ
AWS Secret Access Key [None]: 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
Default region name [None]: us-east-2
Default output format [None]: json
seun@seun-virtual-machine: $
```

## Verify Configuration:

```
cat ~/.aws/credentials
```

```
Activities Terminal Oct 25 10:56
seun@seun-virtual-machine: ~
seun@seun-virtual-machine: $ cat .aws/credentials
[testing]
aws_access_key_id = AKIAZ5NF3I7YGSWHDSS
aws_secret_access_key = DrI0u92mLkkq3cesC4QzxXk4pfPyuHwY/Ebed0QQ
[production]
aws_access_key_id = AKIA4D4CIJ2Y2TZ7YFBZ
aws_secret_access_key = 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
seun@seun-virtual-machine: $
```

## Section 3: Advanced Script Implementation

### 4.1 Enhanced Script Development and Explanation

**Core Functions:** The script has now been enhanced with functions. Some logic that previously wasn't organized into functions has now been refactored into separate ones. Hence, we have the following functions:

```
check_num_of_args
activate_infra_environment
```

```
check_aws_cli
check_aws_profile
create_local_testing_or_prod_resources
```

```
seun@seun-virtual-machine: ~
GNU nano 6.2
#!/bin/bash

# Environment variables
ENVIRONMENT=$1
NUMBER_OF_INSTANCES=$2

check_num_of_args() {
    # Checking the number of arguments
    if [ "$#" -ne 2 ]; then
        echo "Usage: $0 <environment> <number_of_instances>"
        exit 1
    fi
}

activate_infra_environment() {
    # Acting based on the argument value
    if [ "$ENVIRONMENT" == "local" ]; then
        echo "Running script for Local Environment..."
    elif [ "$ENVIRONMENT" == "testing" ]; then
        echo "Running script for Testing Environment..."
    elif [ "$ENVIRONMENT" == "production" ]; then
        echo "Running script for Production Environment..."
    else
        echo "Invalid environment specified. Please use 'local', 'testing', or 'production'." 
        exit 2
    fi
}
```

**Note:** Functions that have not yet been utilized or described will be explained in subsequent parts of this section.

## AWS CLI & Profile Checks:

```
check_aws_cli() {
    if ! command -v aws &> /dev/null; then
        echo "AWS CLI is not installed. Please install it before proceeding."
        exit 1
    fi
}

# Function to check if AWS profile is set
check_aws_profile() {
    if [ -z "$AWS_PROFILE" ] && [ "$ENVIRONMENT" != "local" ]; then
        echo "AWS profile environment variable is not set for $ENVIRONMENT environment."
        exit 1
    fi
}
```

**Note** The above code is not the complete script.

**check\_aws\_cli():** Verifies that the AWS CLI is installed on the system. If not found, it displays an error message and exits the script.

**check\_aws\_profile():** Ensures the `AWS_PROFILE` environment variable is set for non-local environments. If missing, it alerts the user and exits to prevent unauthorized or failed AWS operations.

```

}
# Function to check if AWS CLI is installed
check_aws_cli() {
    if ! command -v aws &> /dev/null; then
        echo "AWS CLI is not installed. Please install it before proceeding."
        exit 1
    fi
}

# Function to check if AWS profile is set
check_aws_profile() {
    if [ -z "$AWS_PROFILE" ] && [ "$ENVIRONMENT" != "local" ]; then
        echo "AWS profile environment variable is not set for $ENVIRONMENT environment."
        exit 1
    fi
}

```

## Resource Creation Logic:

```

# Function to create local log files or EC2 instances
create_local_testing_or_prod_resources() {
    # Validate number_of_instances is a positive integer
    if ! [[ "$NUMBER_OF_INSTANCES" =~ ^[1-9][0-9]*$ ]]; then
        echo "Invalid number_of_instances: Must be a positive integer."
        exit 2
    fi

    if [ "$ENVIRONMENT" == "local" ]; then
        echo "Creating $NUMBER_OF_INSTANCES sample log files locally..."
        for ((i=1; i<=$NUMBER_OF_INSTANCES; i++)); do
            echo "Log entry $i for local simulation" > "log_file_$i.txt"
            if [ $? -eq 0 ]; then
                echo "✓ Created log_file_$i.txt successfully."
            else
                echo "✗ Failed to create log_file_$i.txt."
            fi
        done
    else
        # Common parameters
        ...
    fi
}

```

**Note** The above code snippet is incomplete. It won't work

create\_local\_testing\_or\_prod\_resources(): This function creates resources based on the selected environment.

- It first checks that **NUMBER\_OF\_INSTANCES** is a valid positive integer.
- If the environment is **local**, it simulates resource creation by generating sample log files (**log\_file\_1.txt**, **log\_file\_2.txt**, etc.).
- If the environment is **testing** or **production**, it prepares configuration parameters (like instance type, AMI ID, region, and key pairs) needed to launch EC2 instances using AWS CLI.
- It ensures the setup process matches the environment — lightweight for local testing and actual cloud provisioning for AWS environments.

The screenshot shows a terminal window titled "seun@seun-virtual-machine: ~". The file being edited is "aws\_cloud\_manager.sh". The script contains functions for creating local log files or EC2 instances, validating instance counts, and launching EC2 instances with specific parameters like instance type, AMI ID, and security groups. The terminal includes a menu bar with standard Linux keyboard shortcuts.

```

GNU nano 6.2                               aws_cloud_manager.sh

# Function to create local log files or EC2 instances
create_local_testing_or_prod_resources() {
    # Validate number_of_instances is a positive integer
    if ! [[ "$NUMBER_OF_INSTANCES" =~ ^[1-9][0-9]*$ ]]; then
        echo "Invalid number_of_instances: Must be a positive integer."
        exit 2
    fi

    if [ "$ENVIRONMENT" == "local" ]; then
        echo "Creating $NUMBER_OF_INSTANCES sample log files locally..."
        for ((i=1; i<=$NUMBER_OF_INSTANCES; i++)); do
            echo "Log entry $i for local simulation" > "log_file_$i.txt"
            if [ $? -eq 0 ]; then
                echo "✓ Created log_file_$i.txt successfully."
            else
                echo "✗ Failed to create log_file_$i.txt."
            fi
        done
    else
        # Common parameters
        instance_type="t2.micro"      # for testing environment
        instance_type2="t3.micro"     #for production environment
        ami_id="resolve:ssm:/aws/service/ami-amazon-linux-latest/al2023-ami-kernel-default-x86_64"
        region="us-east-2"
        # No explicit subnet assignment (let AWS choose default)
        subnet_id=None   # or simply omit this parameter when creating the instance
        # Create a new Security Group instead of using an existing one
        create_security_group=True # flag to indicate a new SG should be created
        security_group_name="auto-created-sg"
        security_group_description="Security group automatically created for this instance"
        key_name="my-key-new-pair" # for testing environment
        key_name2="MyKeyPair1"      #for production environment

        echo "Launching $NUMBER_OF_INSTANCES EC2 instances in $ENVIRONMENT subnet: $subnet_id..."
    fi
}

echo "Launching $NUMBER_OF_INSTANCES EC2 instances in $ENVIRONMENT subnet: $subnet_id..."

^G Help      ^O Write Out      ^W Where Is      ^K Cut      ^T Execute      ^C Location      M-U Undo      M-A Set M
^X Exit      ^R Read File      ^A Replace      ^U Paste      ^J Justify      ^L Go To Line      M-E Redo      M-6 Copy

```

## Input Validation:

```

if ! [[ "$NUMBER_OF_INSTANCES" =~ ^[1-9][0-9]*$ ]]; then
    echo "Invalid number_of_instances: Must be a positive integer."
    exit 2
fi

```

**Note** The above code snippet is incomplete.

This block checks whether `NUMBER_OF_INSTANCES` is a **positive integer**.

- The expression `[[ "$NUMBER_OF_INSTANCES" =~ ^[1-9][0-9]*$ ]]` uses a **regular expression** to match numbers starting from 1 and containing only digits.
- The `!` negates the test, meaning “*if the value does **not** match this pattern.*”
- If invalid, it prints an error message and exits the script with **status code 2** to stop execution.

The screenshot shows a terminal window titled "seun@seun-virtual-machine: ~". The file being edited is "aws\_cloud\_manager.sh". The script contains functions for creating local log files or EC2 instances, validating instance counts, and launching EC2 instances with specific parameters like instance type, AMI ID, and security groups. The terminal includes a menu bar with standard Linux keyboard shortcuts.

```

Activities Terminal Oct 25 13:03
seun@seun-virtual-machine: ~
GNU nano 6.2                               aws_cloud_manager.sh *

# Function to create local log files or EC2 instances
create_local_testing_or_prod_resources() {
    # Validate number_of_instances is a positive integer
    if ! [[ "$NUMBER_OF_INSTANCES" =~ ^[1-9][0-9]*$ ]]; then
        echo "Invalid number_of_instances: Must be a positive integer."
        exit 2
    fi
}

```

## Local Environment:

```

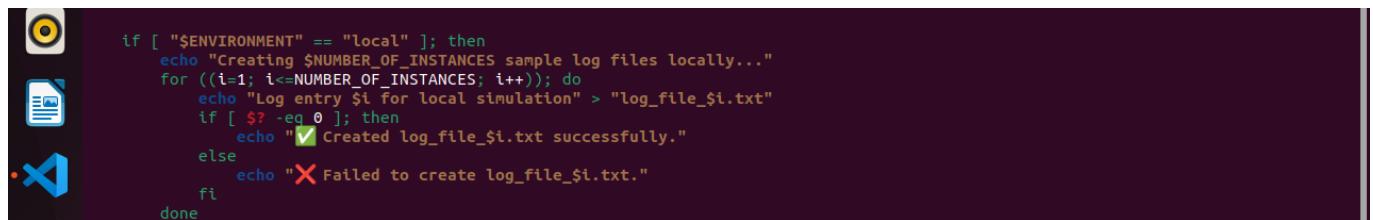
if [ "$ENVIRONMENT" == "local" ]; then
    echo "Creating $NUMBER_OF_INSTANCES sample log files locally..."
    for ((i=1; i<=NUMBER_OF_INSTANCES; i++)); do
        echo "Log entry $i for local simulation" > "log_file_$i.txt"
        if [ $? -eq 0 ]; then
            echo "✓ Created log_file_$i.txt successfully."
        else
            echo "✗ Failed to create log_file_$i.txt."
        fi
    done
else

```

**Note** The above code snippet is incomplete. It won't work.

This block handles the **local environment setup**.

- It checks if **ENVIRONMENT** equals "**local**".
- If true, it creates the specified number of sample log files (**log\_file\_1.txt**, **log\_file\_2.txt**, etc.).
- Each file contains a simple line of text for simulation ("**Log entry i for local simulation**").
- After each creation, it checks the command's exit status (**\$?**) — printing a success **✓** or failure **✗** message accordingly.
- If the environment isn't local, control passes to the **else** block for cloud (AWS) resource creation.



```

if [ "$ENVIRONMENT" == "local" ]; then
    echo "Creating $NUMBER_OF_INSTANCES sample log files locally..."
    for ((i=1; i<=NUMBER_OF_INSTANCES; i++)); do
        echo "Log entry $i for local simulation" > "log_file_$i.txt"
        if [ $? -eq 0 ]; then
            echo "✓ Created log_file_$i.txt successfully."
        else
            echo "✗ Failed to create log_file_$i.txt."
        fi
    done
else

```

## AWS Parameters:

```

# Common parameters
instance_type="t2.micro"      # for testing environment
instance_type2="t3.micro"      #for production environment
ami_id="resolve:ssm:/aws/service/ami-amazon-linux-latest/a12023-ami-
kernel-default-x86_64"
region="us-east-2"
# No explicit subnet assignment (let AWS choose default)
subnet_id=None    # or simply omit this parameter when creating the
instance
# Create a new Security Group instead of using an existing one
create_security_group=True   # flag to indicate a new SG should be created
security_group_name="auto-created-sg"
security_group_description="Security group automatically created for this
instance"
key_name="my-key-new-pair" # for testing environment
key_name2="MyKeyPair1"       #for production environment

```

This section defines **common configuration parameters** for launching AWS EC2 instances.

- `instance_type` and `instance_type2` specify instance sizes — `t2.micro` for **testing** and `t3.micro` for **production**.
- `ami_id` references the **latest Amazon Linux 2023 image** via AWS Systems Manager (SSM).
- `region` sets the deployment location (`us-east-2`).
- `subnet_id` is set to `None`, letting AWS automatically choose a default subnet.
- `create_security_group=True` indicates a new **security group** should be created with a name and description provided.
- `key_name` and `key_name2` define the **SSH key pairs** for connecting to instances in testing and production respectively.

In short, these variables standardize AWS deployment settings across environments.

```
# Common parameters
instance_type="t2.micro"      # for testing environment
instance_type2="t3.micro"     #for production environment
ami_id="resolve:ssm:/aws/service/ami-amazon-linux-latest/al2023-ami-kernel-default-x86_64"
region="us-east-2"
# No explicit subnet assignment (let AWS choose default)
subnet_id=None   # or simply omit this parameter when creating the instance
# Create a new Security Group instead of using an existing one
create_security_group=True  # flag to indicate a new SG should be created
security_group_name="auto-created-sg"
security_group_description="Security group automatically created for this instance"
key_name="my-key-new-pair" # for testing environment
key_name2="MyKeyPair1"       #for production environment

echo "Launching $NUMBER_OF_INSTANCES EC2 instances in $ENVIRONMENT subnet: $subnet_id..."
```

## Testing EC2 Launch:

```
echo "Launching $NUMBER_OF_INSTANCES EC2 instances in $ENVIRONMENT subnet:
$subnet_id..."

# Launch EC2 instances with profile based on environment
if [ "$ENVIRONMENT" == "testing" ]; then
    aws ec2 run-instances --profile testing \
        --image-id "$ami_id" \
        --instance-type "$instance_type" \
        --count "$NUMBER_OF_INSTANCES" \
        --key-name "$key_name" \
        --region "$region" \
        --associate-public-ip-address \
        --tag-specifications "ResourceType=instance,Tags=
[ {Key=Environment,Value=$ENVIRONMENT} ]"
```

**Note** The above code snippet is incomplete. It won't work.

This section launches EC2 instances based on the selected environment. It uses parameters like `--profile` to choose the correct AWS credentials, `--image-id` for the base AMI, `--instance-type` for hardware specs, `--count` for the number of instances, `--key-name` for SSH access, `--region` for deployment location, `--associate-public-ip-address` to enable internet access, and `--tag-specifications` to label instances with environment metadata.

```
# Launch EC2 instances with profile based on environment
if [ "$ENVIRONMENT" == "testing" ]; then
    aws ec2 run-instances --profile testing \
        --image-id "$ami_id" \
        --instance-type "$instance_type" \
        --count "$NUMBER_OF_INSTANCES" \
        --key-name "$key_name" \
        --region "$region" \
        --network-interfaces "DeviceIndex=0,SubnetId=$subnet_id,AssociatePublicIpAddress=true,Groups=$security_group" \
        --tag-specifications "ResourceType=instance,Tags=[{Key=Environment,Value=testing}]" 2>&1
```

## Production EC2 Launch:

```
elif [ "$ENVIRONMENT" == "production" ]; then
    aws ec2 run-instances --profile production \
        --image-id "$ami_id" \
        --instance-type "$instance_type2" \
        --count "$NUMBER_OF_INSTANCES" \
        --key-name "$key_name2" \
        --region "$region" \
        --associate-public-ip-address \
        --tag-specifications "ResourceType=instance,Tags=
[ {Key=Environment,Value=$ENVIRONMENT} ]"
fi
```

**Note** The above code snippet is incomplete. It won't work. This block handles **production environment deployment**.

- The `elif` checks if the environment is **production**.
- It runs `aws ec2 run-instances` with the **production profile** to ensure the correct AWS account and permissions are used.
- It specifies parameters such as:
  - `--image-id`: defines the AMI to use, same as testing.
  - `--instance-type`: uses `t3.micro`, slightly more powerful than `t2.micro`.
  - `--count`: number of instances to create.
  - `--key-name`: production key pair for secure access.
  - `--region`: AWS region for deployment.
  - `--associate-public-ip-address`: assigns public IPs for external access.
  - `--tag-specifications`: labels instances for easy identification by environment.

```
elif [ "$ENVIRONMENT" == "production" ]; then
    aws ec2 run-instances --profile production \
        --image-id "$ami_id" \
        --instance-type "$instance_type" \
        --count "$NUMBER_OF_INSTANCES" \
        --key-name "$key_name" \
        --region "$region" \
        --network-interfaces "DeviceIndex=0,SubnetId=$subnet_id,AssociatePublicIpAddress=true,Groups=$security_group" \
        --tag-specifications "ResourceType=instance,Tags=[{Key=Environment,Value=production}]" 2>&1
fi
```

## Error Handling:

```

if [ $? -eq 0 ]; then
    echo "✅ $NUMBER_OF_INSTANCES EC2 instances created successfully in
$ENVIRONMENT subnet $subnet_id."
else
    echo "✗ Failed to create EC2 instances in $ENVIRONMENT."
    exit
fi
fi
}

```

**Note** The above code snippet is incomplete. It won't work

This block **verifies whether the EC2 instance creation command succeeded or failed**:

- `$?` stores the exit status of the last executed command (`0` means success).
- If the status equals `0`, it prints a success message showing how many instances were created and in which environment/subnet.
- If not, it prints an error message indicating failure and exits the script to stop further execution.

```

if [ $? -eq 0 ]; then
    echo "✅ $NUMBER_OF_INSTANCES EC2 instances created successfully in $ENVIRONMENT subnet $subnet_id."
else
    echo "✗ Failed to create EC2 instances in $ENVIRONMENT."
    exit
fi
}

```

## Function Execution:

```

check_num_of_args "$@"
activate_infra_environment
check_aws_cli
check_aws_profile
create_local_testing_or_prod_resources

```

**Note** The above code snippet is incomplete. It won't work

This sequence defines the **main execution flow** of the script — it runs each function in order to ensure everything is set before creating resources:

1. `check_num_of_args "$@"` – Verifies that the correct number of command-line arguments were passed (e.g., environment name, number of instances).
2. `activate_infra_environment` – Sets up or activates the environment (local, testing, or production) before resource creation.
3. `check_aws_cli` – Ensures that AWS CLI is installed on the system.
4. `check_aws_profile` – Confirms that a valid AWS profile is configured (unless in local mode).
5. `create_local_testing_or_prod_resources` – Executes the main task: creating local files (for local mode) or launching EC2 instances (for testing/production).

```
# Execute functions
check_num_of_args "$@"
activate_infra_environment
check_aws_cli
check_aws_profile
create_local_testing_or_prod_resources
```

## Full Script:

```
#!/bin/bash

# Environment variables
ENVIRONMENT=$1
NUMBER_OF_INSTANCES=$2

check_num_of_args() {
    # Checking the number of arguments
    if [ "$#" -ne 2 ]; then
        echo "Usage: $0 <environment> <number_of_instances>"
        exit 1
    fi
}

activate_infra_environment() {
    # Acting based on the argument value
    if [ "$ENVIRONMENT" == "local" ]; then
        echo "Running script for Local Environment..."
    elif [ "$ENVIRONMENT" == "testing" ]; then
        echo "Running script for Testing Environment..."
    elif [ "$ENVIRONMENT" == "production" ]; then
        echo "Running script for Production Environment..."
    else
        echo "Invalid environment specified. Please use 'local', 'testing', or 'production'.".
        exit 2
    fi
}

# Function to check if AWS CLI is installed
check_aws_cli() {
    if ! command -v aws &> /dev/null; then
        echo "AWS CLI is not installed. Please install it before proceeding."
        exit 1
    fi
}

# Function to check if AWS profile is set
check_aws_profile() {
    if [ -z "$AWS_PROFILE" ] && [ "$ENVIRONMENT" != "local" ]; then
        echo "AWS profile environment variable is not set for $ENVIRONMENT environment."
    fi
}
```

```
        exit 1
    fi
}

# Function to create local log files or EC2 instances
create_local_testing_or_prod_resources() {
    # Validate number_of_instances is a positive integer
    if ! [[ "$NUMBER_OF_INSTANCES" =~ ^[1-9][0-9]*$ ]]; then
        echo "Invalid number_of_instances: Must be a positive integer."
        exit 2
    fi

    if [ "$ENVIRONMENT" == "local" ]; then
        echo "Creating $NUMBER_OF_INSTANCES sample log files locally..."
        for ((i=1; i<=NUMBER_OF_INSTANCES; i++)); do
            echo "Log entry $i for local simulation" > "log_file_$i.txt"
            if [ $? -eq 0 ]; then
                echo "✓ Created log_file_$i.txt successfully."
            else
                echo "✗ Failed to create log_file_$i.txt."
            fi
        done
    else

        # Common parameters
        instance_type="t2.micro"      # for testing environment
        instance_type2="t3.micro"     #for production environment
        ami_id="resolve:ssm:/aws/service/ami-amazon-linux-latest/al2023-ami-kernel-default-x86_64"
        region="us-east-2"
        # No explicit subnet assignment (let AWS choose default)
        subnet_id=None   # or simply omit this parameter when creating the
instance
        # Create a new Security Group instead of using an existing one
        create_security_group=True  # flag to indicate a new SG should be created
        security_group_name="auto-created-sg"
        security_group_description="Security group automatically created for this
instance"
        key_name="my-key-new-pair" # for testing environment
        key_name2="MyKeyValuePair1"      #for production environment

        echo "Launching $NUMBER_OF_INSTANCES EC2 instances in $ENVIRONMENT subnet:
$subnet_id..."

        # Launch EC2 instances with profile based on environment
        if [ "$ENVIRONMENT" == "testing" ]; then
            aws ec2 run-instances --profile testing \
                --image-id "$ami_id" \
                --instance-type "$instance_type" \
                --count "$NUMBER_OF_INSTANCES" \
                --key-name "$key_name" \
                --region "$region" \
                --associate-public-ip-address \
                --tag-specifications "ResourceType=instance,Tags="
```

```
[{Key=Environment,Value=$ENVIRONMENT}]"
    elif [ "$ENVIRONMENT" == "production" ]; then
        aws ec2 run-instances --profile production \
            --image-id "$ami_id" \
            --instance-type "$instance_type2" \
            --count "$NUMBER_OF_INSTANCES" \
            --key-name "$key_name2" \
            --region "$region" \
            --associate-public-ip-address \
            --tag-specifications "ResourceType=instance,Tags=
[{Key=Environment,Value=$ENVIRONMENT}]"
    fi

    if [ $? -eq 0 ]; then
        echo "✅ $NUMBER_OF_INSTANCES EC2 instances created successfully in
$ENVIRONMENT subnet $subnet_id."
    else
        echo "❌ Failed to create EC2 instances in $ENVIRONMENT."
        exit
    fi
fi
}

# Execute functions
check_num_of_args "$@"
activate_infra_environment
check_aws_cli
check_aws_profile
create_local_testing_or_prod_resources
```

```

Activities Terminal Oct 25 12:38
seun@seun-virtual-machine: ~
GNU nano 6.2 aws_cloud_manager.sh

#!/bin/bash

# Environment variables
ENVIRONMENT=$1
NUMBER_OF_INSTANCES=$2

check_num_of_args() {
    # Checking the number of arguments
    if [ "$#" -ne 2 ]; then
        echo "Usage: $0 <environment> <number_of_instances>"
        exit 1
    fi
}

activate_infra_environment() {
    # Acting based on the argument value
    if [ "$ENVIRONMENT" == "local" ]; then
        echo "Running script for Local Environment..."
    elif [ "$ENVIRONMENT" == "testing" ]; then
        echo "Running script for Testing Environment..."
    elif [ "$ENVIRONMENT" == "production" ]; then
        echo "Running script for Production Environment..."
    else
        echo "Invalid environment specified. Please use 'local', 'testing', or 'production'."
        exit 1
    fi
}

# Function to check if AWS CLI is installed
check_aws_cli() {
    if ! command -v aws &> /dev/null; then
        echo "AWS CLI is not installed. Please install it before proceeding."
        exit 1
    fi
}

AG Help ^O Write Out ^W Where Is ^K Cut ^T Execute
^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify ^C Location M-U Undo
M-, M-^ Go To Line M-E Redo M-.

```

## Section 4: Comprehensive Testing

### 5.1 Local Environment Test

```
./aws_cloud_manager.sh local 2
```

```

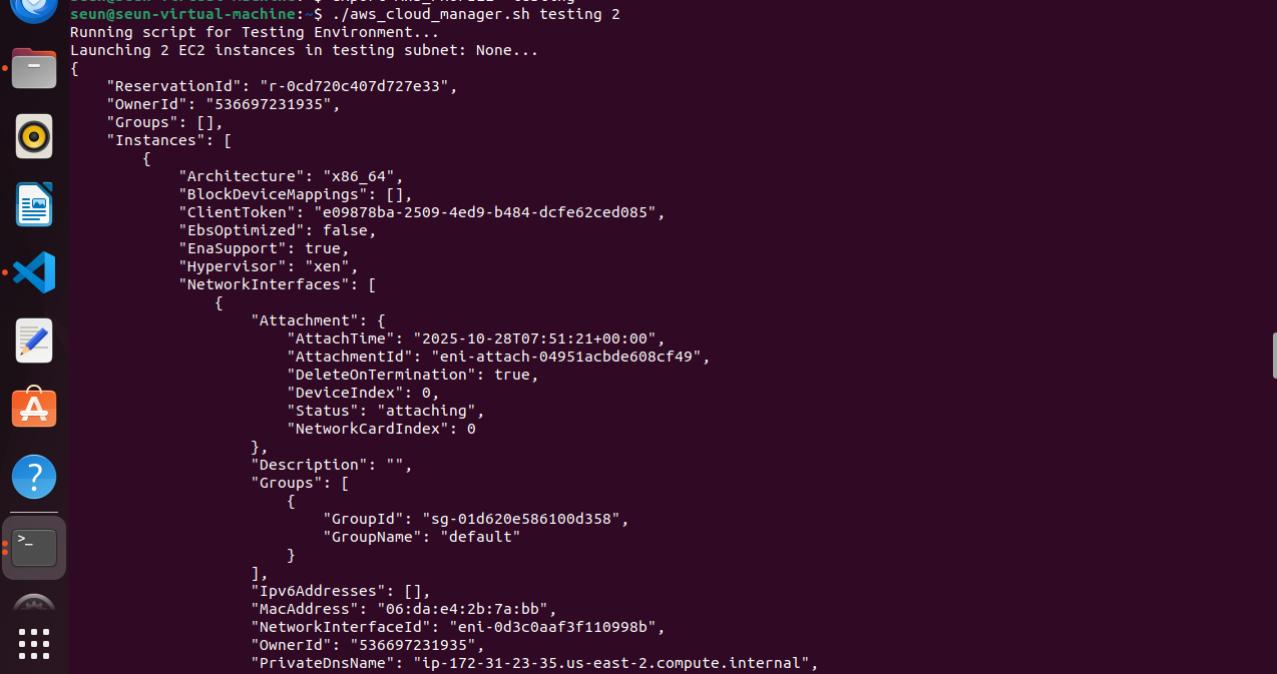
seun@seun-virtual-machine:~$ ./aws_cloud_manager.sh local 2
Running script for Local Environment...
Creating 2 sample log files locally...
✓ Created log_file_1.txt successfully.
✓ Created log_file_2.txt successfully.
seun@seun-virtual-machine:~$ 
seun@seun-virtual-machine:~$ ls log_file_1.txt log_file_2.txt
log_file_1.txt  log_file_2.txt
seun@seun-virtual-machine:~$ 

```

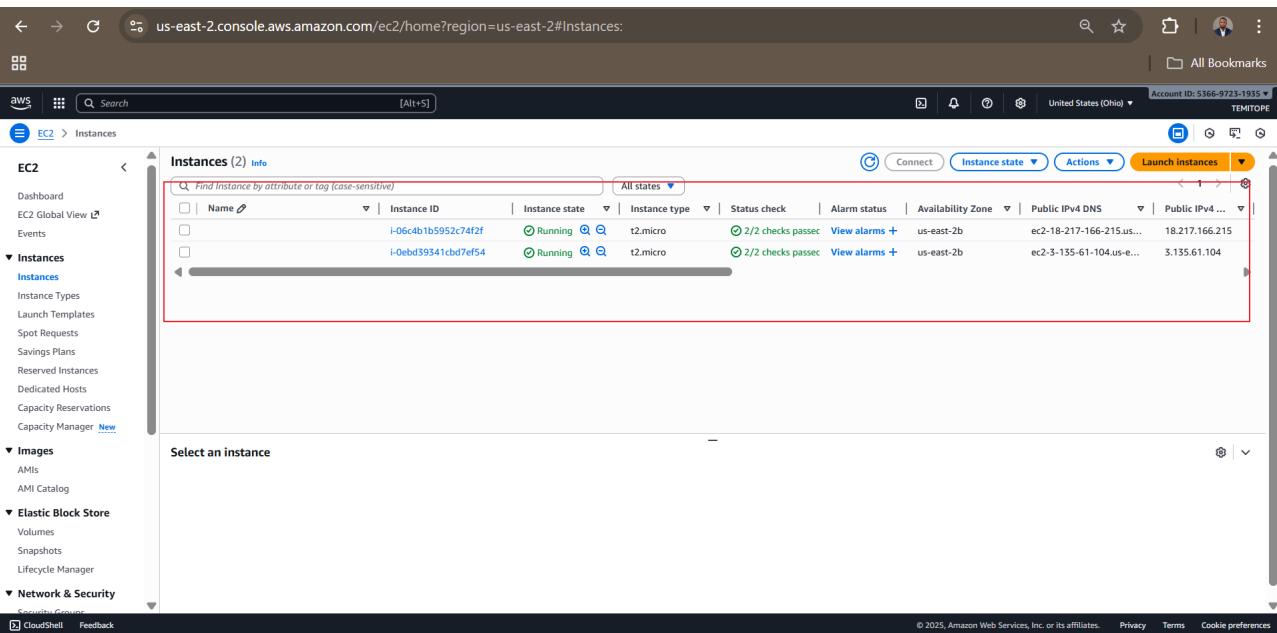
### 5.2 Testing Environment Test

```
./aws_cloud_manager.sh testing 2
```

- After execution, two EC2 Instances were successfully created in testing environment (AWS Account 1) as instructed by the script.



```
seun@seun-virtual-machine: ~ $ ./aws_cloud_manager.sh testing 2
Running script for Testing Environment...
Launching 2 EC2 instances in testing subnet: None...
{
    "ReservationId": "r-0cd720c407d727e33",
    "OwnerId": "536697231935",
    "Groups": [],
    "Instances": [
        {
            "Architecture": "x86_64",
            "BlockDeviceMappings": [],
            "ClientToken": "e09878ba-2509-4ed9-b484-dcfe62ced085",
            "EbsOptimized": false,
            "EnaSupport": true,
            "Hypervisor": "xen",
            "NetworkInterfaces": [
                {
                    "Attachment": {
                        "AttachTime": "2025-10-28T07:51:21+00:00",
                        "AttachmentId": "eni-attach-04951acbde608cf49",
                        "DeleteOnTermination": true,
                        "DeviceIndex": 0,
                        "Status": "attaching",
                        "NetworkCardIndex": 0
                    },
                    "Description": "",
                    "Groups": [
                        {
                            "GroupId": "sg-01d620e586100d358",
                            "GroupName": "default"
                        }
                    ],
                    "Ipv6Addresses": [],
                    "MacAddress": "06:da:e4:2b:7a:bb",
                    "NetworkInterfaceId": "eni-0d3c0aaaf3f110998b",
                    "OwnerId": "536697231935",
                    "PrivateDnsName": "ip-172-31-23-35.us-east-2.compute.internal",
                    "PrivateIpAddress": "172.31.23.35"
                }
            ]
        }
    ]
}
```



The screenshot shows the AWS CloudWatch Metrics console with a chart titled 'CPU Utilization' for two instances. The chart displays two data series: one for each instance, showing utilization levels fluctuating between 0% and 100% over a period of approximately 24 hours.

### 5.3 Production Environment Test

```
./aws_cloud_manager.sh production 1
```

- After execution, two EC2 Instances were successfully created in production environment (AWS Account 2) as instructed by the script.

```
Activities Terminal Oct 28 08:29
seun@seun-virtual-machine: ~
seun@seun-virtual-machine:~$ ./aws_cloud_manager.sh production 2
Running script for Production Environment...
Launching 2 EC2 instances in production subnet: None...
{
  "ReservationId": "r-064cea5dafd264eb1",
  "OwnerId": "832959958705",
  "Groups": [],
  "Instances": [
    {
      "Architecture": "x86_64",
      "BlockDeviceMappings": [],
      "ClientToken": "1e4953ea-5e40-4e61-8f91-7c718e9c77eb",
      "EbsOptimized": false,
      "EnaSupport": true,
      "Hypervisor": "xen",
      "NetworkInterfaces": [
        {
          "Attachment": {
            "AttachTime": "2025-10-28T08:29:18+00:00",
            "AttachmentId": "eni-attach-092fffc9354e44c7a6",
            "DeleteOnTermination": true,
            "DeviceIndex": 0,
            "Status": "attaching",
            "NetworkCardIndex": 0
          },
          "Description": "",
          "Groups": [
            {
              "GroupId": "sg-0e3ece17a71b13d4b",
              "GroupName": "default"
            }
          ],
          "Ipv6Addresses": [],
          "MacAddress": "06:d3:19:71:10:43",
          "NetworkInterfaceId": "eni-02f90c674593c20fd",
          "OwnerId": "832959958705",
          "PrivateDnsName": "ip-172-31-29-65.us-east-2.compute.internal",
          "PrivateIpAddress": "172.31.29.65",
          "PrivateIpAddresses": []
        }
      ]
    }
  ]
}
```

us-east-2.console.aws.amazon.com/ec2/home?region=us-east-2#Instances:instanceState=running

Instances (2) Info

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS
i-0efad5b13e57fa757	Running	t3.micro	3/3 checks passed	<a href="#">View alarms</a>	us-east-2b	ec2-18-117-120-199	
i-01cf3ca7096e1f0b3	Running	t3.micro	3/3 checks passed	<a href="#">View alarms</a>	us-east-2b	ec2-3-23-94-209.us-east-2	

## Key Learnings & Feedback Addressed

Original Feedback	How Addressed
No AWS CLI Integration	Full <code>run-instances</code> implementation with profiles
Missing NUMBER_OF_INSTANCES	\$2 parameter with regex validation
Placeholder Logic	Real EC2 provisioning + local file operations
Error Handling	Exit codes 1-3, input validation, AWS checks
Testing Gaps	Comprehensive testing across all environments

## Conclusion

This project successfully demonstrated the **full lifecycle of environment-aware cloud scripting**, from conceptual understanding to production-grade automation. By progressing through **two structured phases**, we evolved a basic conditional script into a **secure, reusable, and robust cloud management tool**.

Key achievements include:

- **Eliminated hard-coding** in favor of positional parameters and AWS CLI profiles
- **Implemented multi-account security** using dedicated IAM users and isolated profiles
- **Automated real EC2 provisioning** with environment-specific configurations (`t2.micro` for testing, `t3.micro` for production)
- **Added comprehensive validation** for arguments, AWS CLI, and profile existence
- **Verified functionality** across **local, testing, and production environments** with AWS Console evidence

The final script `aws_cloud_manager.sh` is **modular, well-documented**, and **ready for integration** into CI/CD pipelines or infrastructure-as-code frameworks. This implementation serves as a **strong portfolio piece** showcasing **Bash scripting, AWS automation, security best practices**, and **DevOps methodology**.

---

## Recommendations

To extend this project into a **production-ready automation framework**, the following enhancements are recommended:

### 1. Adopt Terraform or AWS CDK

Replace imperative `aws ec2 run-instances` calls with declarative infrastructure-as-code for idempotency and version control.

### 2. Integrate with CI/CD

Trigger the script via **GitHub Actions** on code merge to testing/production branches.

### 3. Add Cleanup Functionality

Implement a `--destroy` flag to terminate instances and delete log files for cost control.

### 4. Use AWS Systems Manager Parameter Store

Store `NUMBER_OF_INSTANCES`, AMIs, and subnet IDs securely instead of hard-coding.

### 5. Enhance Logging

Redirect output to structured logs (e.g., JSON) and send to CloudWatch.

### 6. Add Dry-Run Mode

Simulate actions without executing AWS API calls for safer testing.

### 7. Implement Idempotency

Check for existing resources (by tag) before creating new ones.

### 8. Containerize the Script

Package in a Docker image for consistent execution across environments.

These improvements would transform this educational project into a **scalable, enterprise-grade automation tool**.

**Let's connect on LinkedIn to discuss DevOps, automation, and cloud engineering!**

[linkedin.com](https://linkedin.com)