

AWS Secure Authentication Setup for API Automation

Author: Oluwaseun Osunsola

Environment: AWS

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

Introduction

In the rapidly evolving world of cloud computing, securing access to AWS services is crucial for developers, DevOps engineers, and organizations aiming to automate infrastructure tasks efficiently. This comprehensive guide walks you through setting up secure programmatic authentication to AWS APIs using IAM roles, custom policies, dedicated users, and the AWS CLI. By implementing these practices, you'll enable automated operations like creating EC2 instances and managing S3 buckets via shell scripts, all while minimizing security risks such as credential leaks or unauthorized access.

This tutorial is designed for beginners and intermediate users looking to integrate AWS automation into their workflows, such as CI/CD pipelines or custom scripts. Emphasizing best practices like the principle of least privilege and multi-factor authentication (MFA), it ensures your setup is robust, scalable, and compliant with industry standards.

Objectives

The main goals of this project are to:

- Establish secure authentication mechanisms for programmatic access to AWS APIs.
- Create and configure IAM components specifically for EC2 and S3 automation.
- Install and set up the AWS CLI on Ubuntu for command-line interactions.
- Develop and troubleshoot a shell script for resource automation, including permission fixes.
- Promote cost efficiency through resource cleanup and best practices.

Definition of Terms

To ensure clarity, here are key terms used throughout this guide:

- **AWS (Amazon Web Services):** A comprehensive cloud computing platform providing services like computing power, storage, and databases.
- **IAM (Identity and Access Management):** An AWS service that helps manage secure access to AWS resources by defining users, roles, and permissions.
- **EC2 (Elastic Compute Cloud):** An AWS service offering scalable virtual servers (instances) in the cloud.
- **S3 (Simple Storage Service):** An AWS object storage service for storing and retrieving data at any scale.
- **AWS CLI (Command Line Interface):** A unified tool to manage AWS services from the terminal using commands.
- **MFA (Multi-Factor Authentication):** A security process requiring two or more verification methods to access an account.
- **Access Keys:** Credentials consisting of an Access Key ID and Secret Access Key used for programmatic access to AWS.

- **SSM (Systems Manager)**: An AWS service for managing and automating operational tasks across resources.
- **Shell Script**: A text file containing a sequence of commands for a Unix-based shell, used here for automation.

Prerequisites

Before starting, ensure you have:

- An AWS account with MFA enabled.
- Ubuntu Linux 22.04 (or a compatible distribution).
- Basic knowledge of terminal commands.
- A code editor like VS Code or nano.
- Access to a web browser for AWS documentation and console.

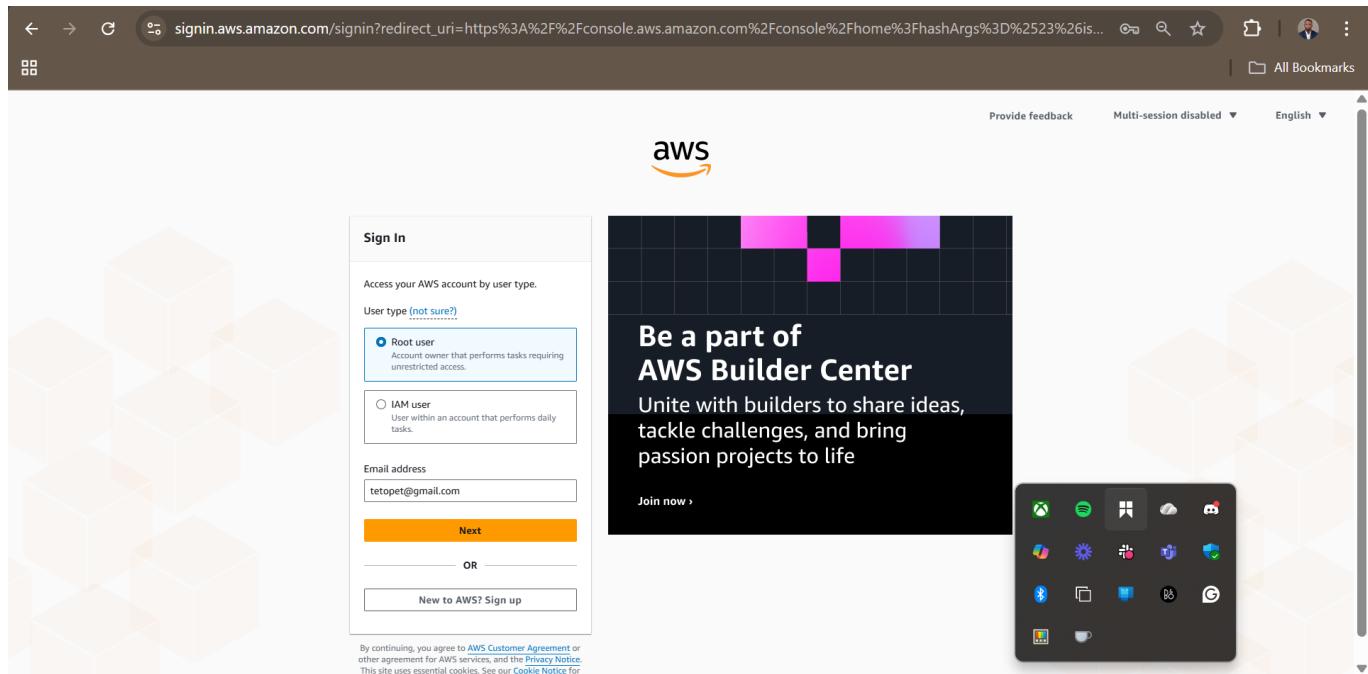
🚀 Step-by-Step Implementation

Phase 1: AWS Console Authentication

Step 1: Sign In to AWS Account with MFA

- Enter AWS account login credentials
- Complete Multi-Factor Authentication (MFA) verification

Purpose: Secure console access with 2FA



Step 2: Successful Console Login

- AWS Management Console dashboard loaded

Status: Ready for IAM configuration

The screenshot shows the AWS Console Home page. On the left, there's a sidebar titled "Recently visited" with icons for EC2, VPC, S3, IAM, IAM Identity Center, CodeBuild, Secrets Manager, and CodePipeline. Below this is a "View all services" link. To the right, there are several widgets: "Applications" (0), "AWS Health" (0 open issues, 0 scheduled changes), and "Cost and usage". A large "Welcome to AWS" box on the left side of the main area contains a "Getting started with AWS" section. At the bottom, there are links for CloudShell, Feedback, and a footer with copyright information.

Phase 2: IAM Policy Creation

Step 3: Navigate to IAM Service

- Use search bar: type "IAM"
- Select IAM from services results

Purpose: Access Identity and Access Management console

The screenshot shows the AWS Console Home page with a red arrow pointing to the search bar at the top left, which contains the text "iam". The search results are displayed in the main pane, showing the "Services" section with the "IAM" service highlighted. Other services listed include IAM Identity Center and Resource Access Manager. Below the services are sections for "Features", "Groups", and "Roles". A sidebar on the left provides links for Services, Features, Documentation, and Knowledge articles. At the bottom, there are links for CloudShell, Feedback, and a footer with copyright information.

Step 4: Access Policies Section

- Click Policies in left sidebar

Purpose: Manage custom permission policies

The screenshot shows the AWS IAM Dashboard. On the left sidebar, under the 'Access management' section, the 'Policies' option is highlighted with a red arrow. The main content area displays 'Security recommendations' (Root user has MFA, Root user has no active access keys), 'IAM resources' (User groups: 0, Users: 1, Roles: 6, Policies: 12, Identity providers: 0), 'What's new' (recent updates like Amazon Bedrock API keys, AWS Service Reference annotations, and AWS IAM MFA enforcement), and 'Quick Links' and 'Tools' sections.

Step 5: Initiate Policy Creation

- Click Create policy button

Purpose: Define custom EC2+S3 permissions

The screenshot shows the 'Policies' page in the AWS IAM console. The 'Policies' section lists 1410 policies. A red arrow points to the 'Create policy' button at the top right of the table. The table columns include Policy name, Type, Used as, and Description. The 'Used as' column shows that many policies are associated with 'Permissions policy (1)'.

Policy name	Type	Used as	Description
AccessAnalyzerServiceRolePolicy	AWS managed	None	Allow Access Analyzer to analyze resou...
AdministratorAccess	AWS managed - job function	Permissions policy (1)	Provides full access to AWS services an...
AdministratorAccess-Amplify	AWS managed	None	Grants account administrative permis...
AdministratorAccess-AWSElasticBeanstalk	AWS managed	None	Grants account administrative permis...
AIOpsAssistantIncidentReportPolicy	AWS managed	None	Provides permissions required by the A...
AIOpsAssistantPolicy	AWS managed	None	Provides ReadOnly permissions requir...
AIOpsConsoleAdminPolicy	AWS managed	None	Grants full access to Amazon AI Opera...
AIOpsOperatorAccess	AWS managed	None	The Amazon AI Opera...
AIOpsReadOnlyAccess	AWS managed	None	permissions to the A...
AlexaForBusinessDeviceSetup	AWS managed	None	up access to AlexaFo...
AlexaForBusinessFullAccess	AWS managed	None	to AlexaForBusiness ...
AlexaForBusinessGatewayExecution	AWS managed	None	execution access to A...
AlexaForBusinessLifesizeDelegatedAccessPolicy	AWS managed	None	Lifesize AVS devices

Step 6: JSON Policy Editor

- Select JSON tab

Purpose: Manual policy definition for precision

The screenshot shows the AWS IAM 'Create policy' wizard at Step 1: 'Specify permissions'. The 'JSON' tab is selected in the top navigation bar. The main area displays a JSON editor with a single statement:

```
1 Version: "2012-10-17",  
2 Statement: [  
3     {  
4         Sid: "Statement1",  
5         Effect: "Allow",  
6         Action: [],  
7         Resource: []  
8     }  
9 ]  
10 }  
11 ]
```

To the right of the editor, there's a 'Select a statement' dropdown menu with icons for various services like Lambda, S3, and CloudWatch. A red arrow points to the 'JSON' tab in the top bar.

Step 7: Define EC2+S3 Full Access Policy

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "ec2:*",  
                "s3:/*"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

- Replace default JSON with EC2+S3 permissions
- Click Next

Specify permissions Info
Add permissions by selecting services, actions, resources, and conditions. Build permission statements using the JSON editor.

```

1 "Version": "2012-10-17",
2 "Statement": [
3     {
4         "Effect": "Allow",
5         "Action": "ec2:DescribeInstances",
6         "Resource": "*"
7     }
8 ]
9
10
11
12
13
14

```

+ Add new statement

JSON Line 14, Col 0
Security: 0 Errors: 0 Warnings: 0 Suggestions: 0
6046 of 6144 characters remaining

Cancel Next

Step 8: Name and Create Policy

- Name: EC2S3FullAccessPolicy
- Click Create policy

Result: Custom policy available for attachment

Review and create Info
Review the permissions, specify details, and tags.

Policy details

Policy name
Enter a meaningful name to identify this policy.
EC2S3FullAccessPolicy
Maximum 128 characters. Use alphanumeric and +,-,_,=,` characters.

Description - optional
Add a short explanation for this policy.
Policy granting full access to EC2 and S3
Maximum 1,000 characters. Use alphanumeric and +,-,_,=,` characters.

Permissions defined in this policy Info
Permissions defined in this policy document specify which actions are allowed or denied. To define permissions for an IAM identity (user, user group, or role), attach a policy to it.

Service	Access level	Resource	Request condition
EC2	Full access	All resources	None
S3	Full access	All resources	None

Add tags - optional Info
Tags are key-value pairs that you can add to AWS resources to help identify, organize, or search for resources.
No tags associated with the resource.
Add new tag
You can add up to 50 more tags.

Cancel Previous Create policy

Step 9: Verify Policy Creation

- Policy appears in Policies list
- Navigate to Roles section

Policies (1411) info

A policy is an object in AWS that defines permissions.

Policy name	Type	Used as	Description
AccessAnalyzerServiceRolePolicy	AWS managed	None	Allow Access Analyzer to analyze resou...
AdministratorAccess	AWS managed - job function	Permissions policy (1)	Provides full access to AWS services an...
AdministratorAccess-AWSFleet-Amplify	AWS managed	None	Grants account administrative permisi...
AdministratorAccess-AWSFleetBeanstalk	AWS managed	None	Grants account administrative permisi...
AIOpsAssistantInCloudReportPolicy	AWS managed	None	Provides permissions required by the A...
AIOpsAssistantPolicy	AWS managed	None	Provides ReadOnly permissions require...
AIOpsConsoleAdminPolicy	AWS managed	None	Grants full access to Amazon AI Opera...
AIOpsOperatorAccess	AWS managed	None	Grants access to the Amazon AI Opera...
AIOpsReadOnlyAccess	AWS managed	None	Grants ReadOnly permissions to the A...
AlexaForBusinessDeviceSetup	AWS managed	None	Provide device setup access to AlexaFo...
AlexaForBusinessFullAccess	AWS managed	None	full access to AlexaForBusiness ...
AlexaForBusinessGatewayExecution	AWS managed	None	gateway execution access to Al...
AlexaForBusinessLifesizeDelegatedAccessPolicy	AWS managed	None	access to Lifesize AVS devices
AlexaForBusinessNetworkProfileServicePolicy	AWS managed	None	policy enables Alexa for Business ...
AlexaForBusinessPolyDelegatedAccessPolicy	AWS managed	None	access to Poly AVS devices
AlexaForBusinessReadOnlyAccess	AWS managed	None	read only access to AlexaForBu...
AmazonAPIGatewayAdministrator	AWS managed	None	full access to create/edit/delete...
AmazonAPIGatewayInvokeFullAccess	AWS managed	None	full access to invoke APIs in A...

Phase 3: IAM Role Creation

Step 10: Start Role Creation

- Click Create role

Purpose: Create service-trusted role for EC2

Roles (6) info

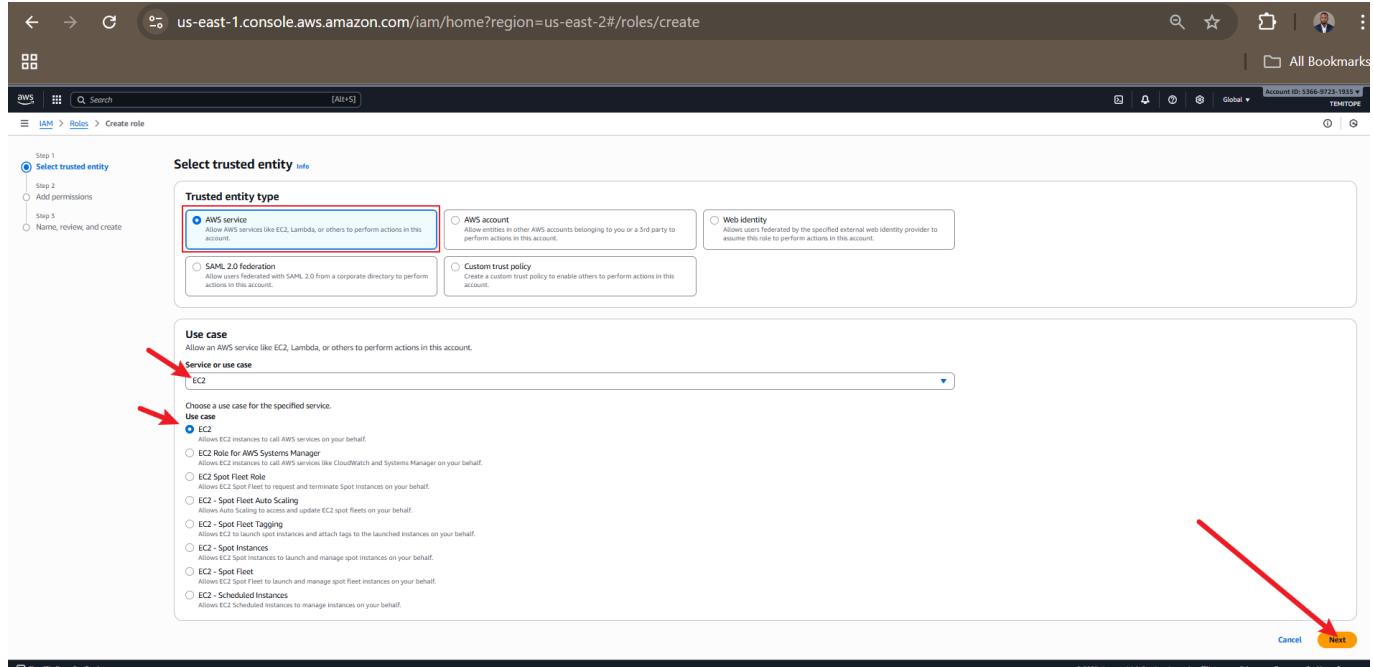
An IAM role is an identity you can create that has specific permissions with credentials that are valid for short durations. Roles can be assumed by entities that you trust.

Role name	Trusted entities	Last activity
AWSServiceRoleForAutoScaling	AWS Service: autoscaling	11 hours ago
AWSServiceRoleForElasticLoadBalancing	AWS Service: elasticloadbalancing	11 hours ago
AWSServiceRoleForResourceExplorer	AWS Service: resource-explorer-2	18 minutes ago
AWSServiceRoleForSupport	AWS Service: support	-
AWSServiceRoleForTrustedAdvisor	AWS Service: trustedadvisor	-
codebuild-JAVA-SaaS-service-role	AWS Service: codebuild	-

Step 11: Configure Trusted Entity

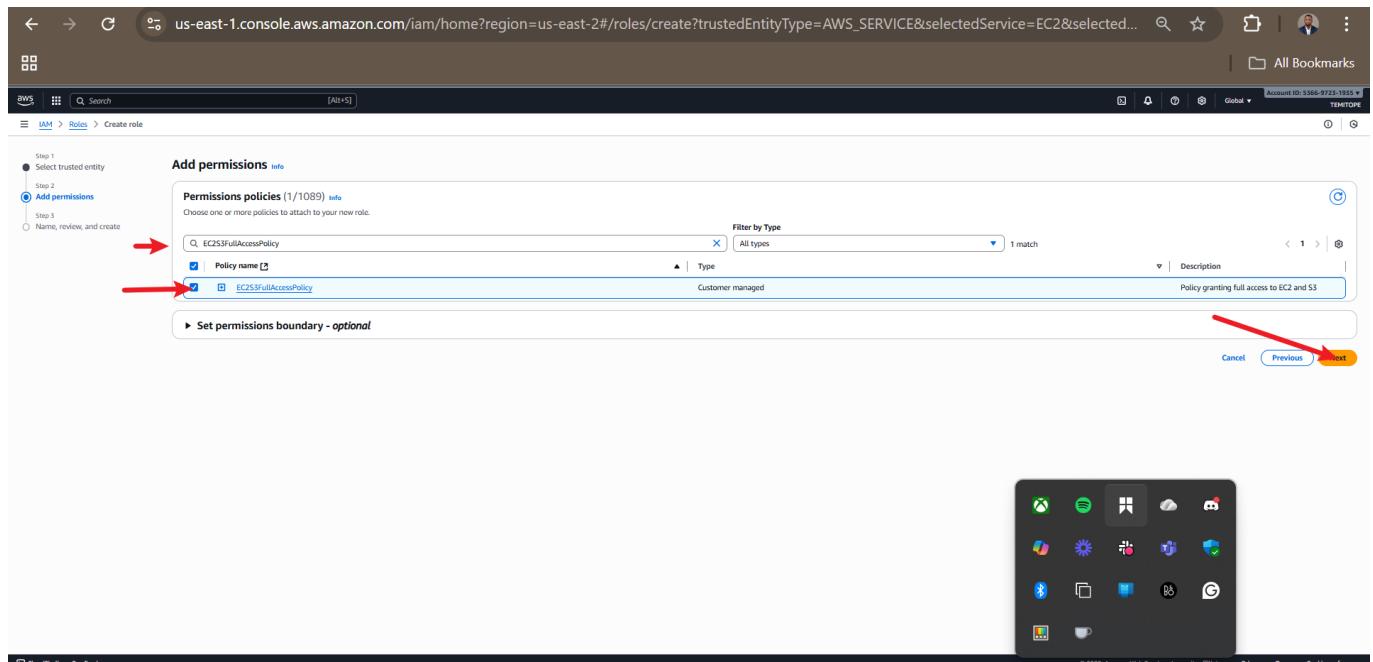
- Trusted entity: AWS service
- Use case: Search and select EC2
- Click Next

Purpose: Allow EC2 instances to assume this role



Step 12: Attach Policy to Role

- Search: EC2S3FullAccessPolicy
- Select policy checkbox
- Click Next



Step 13: Name the Role

- Role name: AutomationRole
- Scroll to review settings

Step 1: Select trusted entity

Name, review, and create

Role details

Role name: AutomationRole

Description: Allows EC2 instances to call AWS services on your behalf.

Step 1: Select trusted entities

Trust policy

```

1: [
2:   "version": "2012-10-17",
3:   "statement": [
4:     {
5:       "effect": "Allow",
6:       "action": "sts:AssumeRole",
7:       "resource": "*"
8:     },
9:     {
10:      "principal": [
11:        "ec2.amazonaws.com"
12:      ]
13:    }
14:  ]
15: ]
16: ]

```

Step 2: Add permissions

Permissions policy summary

Policy name	Type	Attached as	Permissions policy
EC2SSFullAccessPolicy	Customer managed		

Step 14: Create Role

- Review summary
- Click Create role

Result: AutomationRole ready

Step 1: Select trusted entities

Trust policy

```

1: [
2:   "version": "2012-10-17",
3:   "statement": [
4:     {
5:       "effect": "Allow",
6:       "action": "sts:AssumeRole",
7:       "resource": "*"
8:     },
9:     {
10:      "principal": [
11:        "ec2.amazonaws.com"
12:      ]
13:    }
14:  ]
15: ]
16: ]

```

Step 2: Add permissions

Permissions policy summary

Policy name	Type	Attached as	Permissions policy
EC2SSFullAccessPolicy	Customer managed		

Step 3: Add tags

Add tags - optional info

Tags are key-value pairs that you can add to AWS resources to help identify, organize, or search for resources.

No tags associated with the resource.

Add new tag

You can add up to 50 more tags.

Create role

Step 15: Role Dashboard Verification

- AutomationRole visible in roles list
- Navigate to Users

Roles (7) Info

An IAM role is an identity you can create that has specific permissions with credentials that are valid for short durations. Roles can be assumed by entities that you trust.

Role name	Trusted entities	Last activity
AutomationRole	AWS Service: ec2	-
AWSServiceRoleForAutoScaling	AWS Service: autoscaling (Service-Link)	15 hours ago
AWSServiceRoleForElasticLoadBalancing	AWS Service: elasticloadbalancing (Service-Link)	15 hours ago
AWSServiceRoleForResourceExplorer	AWS Service: resource-explorer-2 (Service-Link)	1 hour ago
AWSServiceRoleForSupport	AWS Service: support (Service-Link)	-
AWSServiceRoleForTrustedAdvisor	AWS Service: trustedadvisor (Service-Link)	-
codebuild-JAVA-SAST-service-role	AWS Service: codebuild	-

Roles Anywhere Info

Authenticate your non AWS workloads and securely provide access to AWS services.

Access AWS from your non AWS workloads

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

X.509 Standard

Use your own existing PKI infrastructure or use AWS Certificate Manager Private Certificate Authority to authenticate identities.

Phase 4: IAM User Creation

Step 16: Create Automation User

- Click Create user

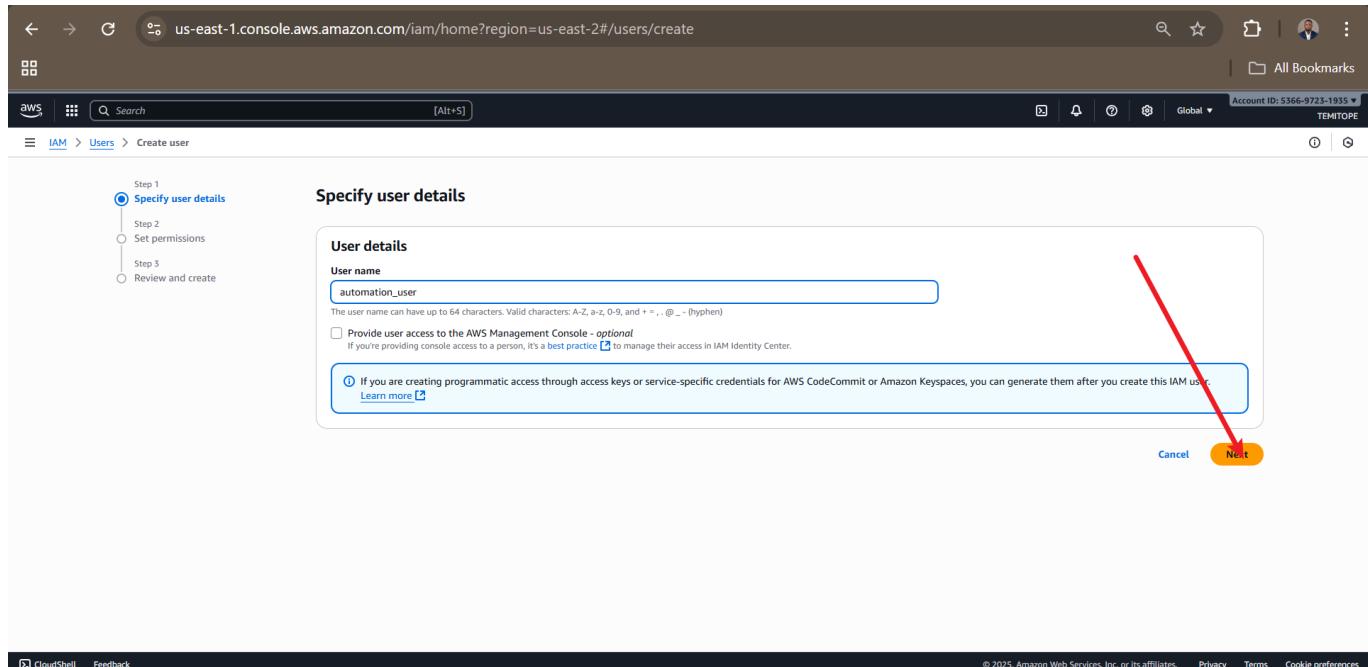
Users (1) Info

An IAM user is an identity with long-term credentials that is used to interact with AWS in an account.

User name	Path	Group	Last activity	MFA	Password age	Console last sign-in	Access key ID	Active key age	Access
oluwasenue	/	0	Yesterday	-	-	-	Active - AKIAZ5NF3...	2 days	2 days

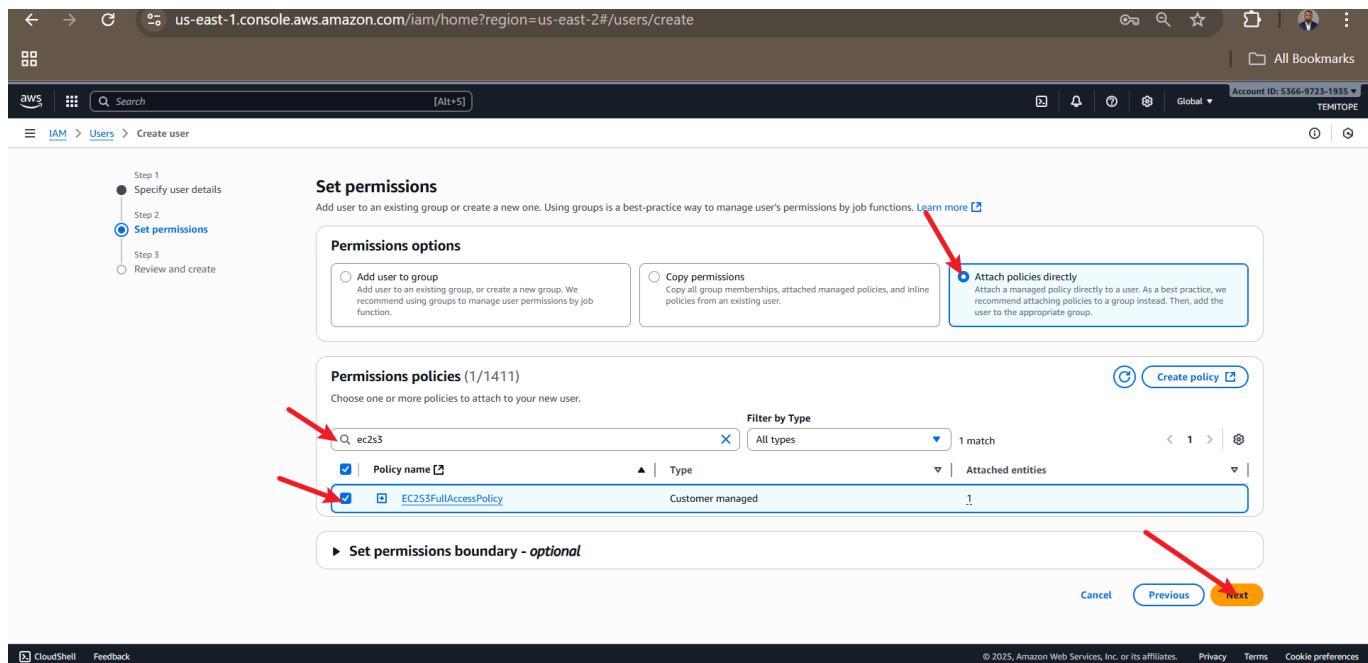
Step 17: Name the User

- User name: automation_user
- Click Next



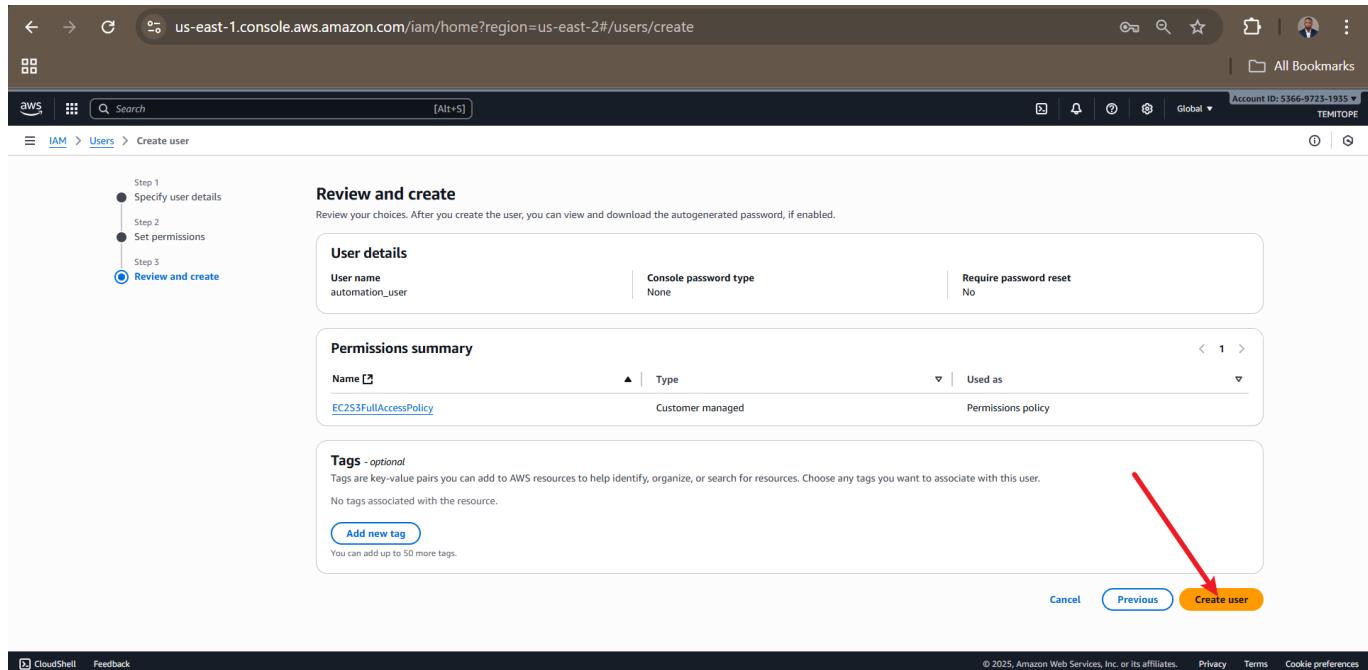
Step 18: Attach Policy to User

- Select Attach policies directly
- Search: EC2S3FullAccessPolicy
- Select and click Next



Step 19: Review and Create User

- Review user configuration
- Click Create user



Step 20: User Creation Success

- automation_user appears in Users list
- Click user for credential setup

User name	Path	Group	Last activity	MFA	Password age	Console last sign-in	Access key ID	Active key age	Access
automation_user	/	0	-	-	-	-	Active - AKIAZ25NF3i7...	2 days	
oluwasen	/	0	Yesterday	-	-	-	Active - AKIAZ25NF3i7...	2 days	

Phase 5: Generate Access Keys

Step 21: Access Security Credentials

- Click Security credentials tab
- Find Access keys section
- Click Create access key

The screenshot shows the AWS IAM User Details page for a user named 'automation_user'. The 'Security Credentials' tab is selected. In the 'Access keys' section, there is one existing access key labeled 'Access key 1' with a 'Create access key' button. Another red arrow points to this 'Create access key' button.

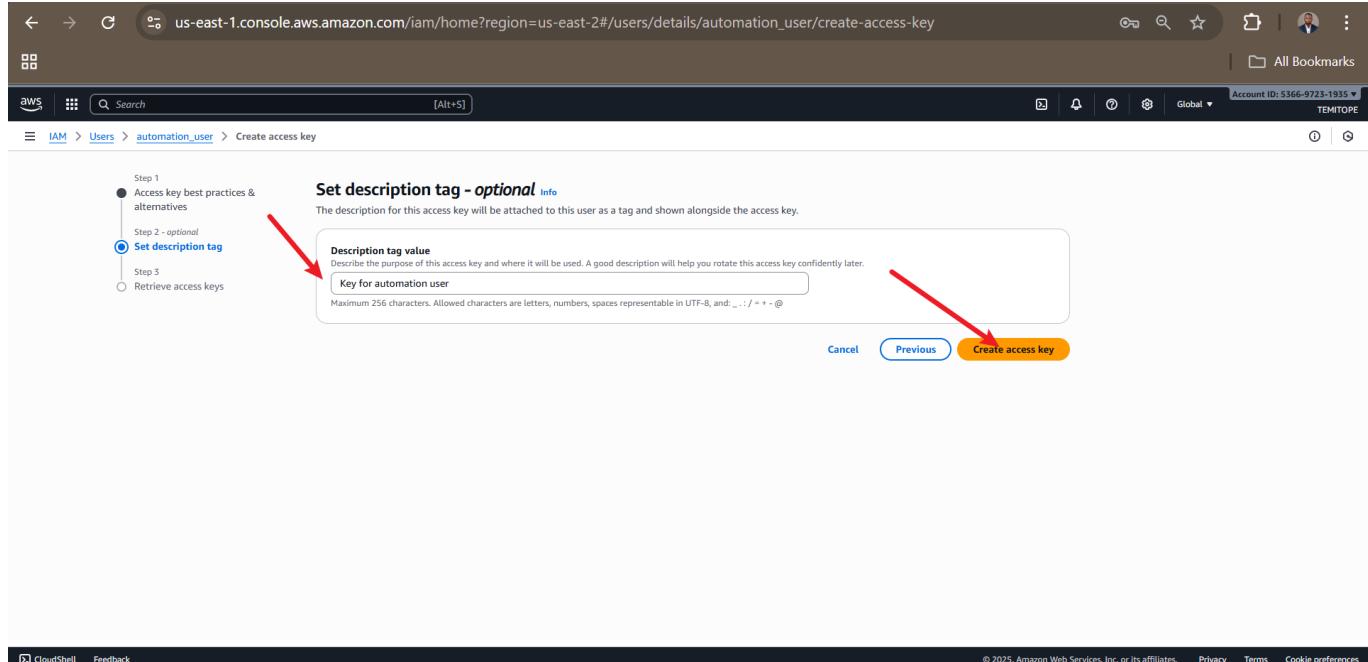
Step 22: Select CLI Use Case

- Use case: Command Line Interface (CLI)
- Check recommendation acknowledgment
- Click Next

The screenshot shows the 'Create access key' wizard, Step 1: Access key best practices & alternatives. It lists several use cases, with 'Command Line Interface (CLI)' selected. At the bottom, there is a 'Confirmation' section with a checkbox and a 'Next' button. Red arrows point to both the selected use case and the 'Next' button.

Step 23: Add Description (Optional)

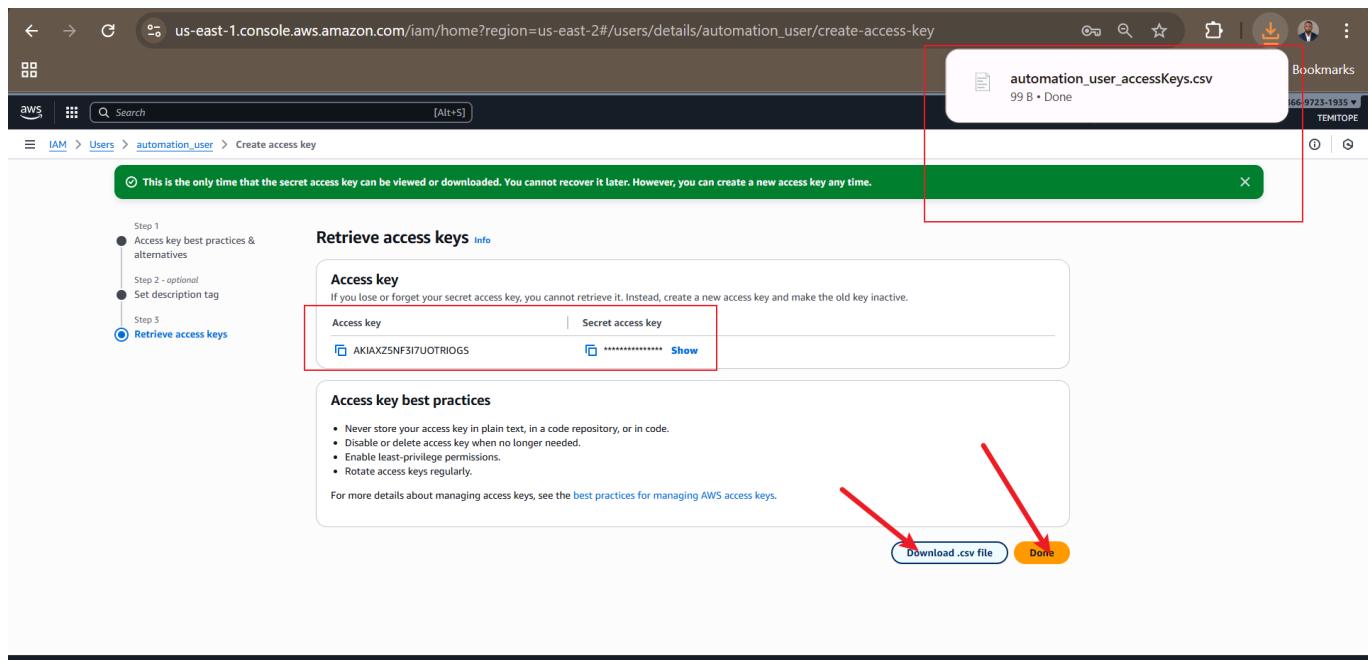
- Add optional description tag
- Click Create access key



Step 24: Download Credentials

Critical: Download .csv file immediately

- Contains Access Key ID and Secret Access Key
- Click Done



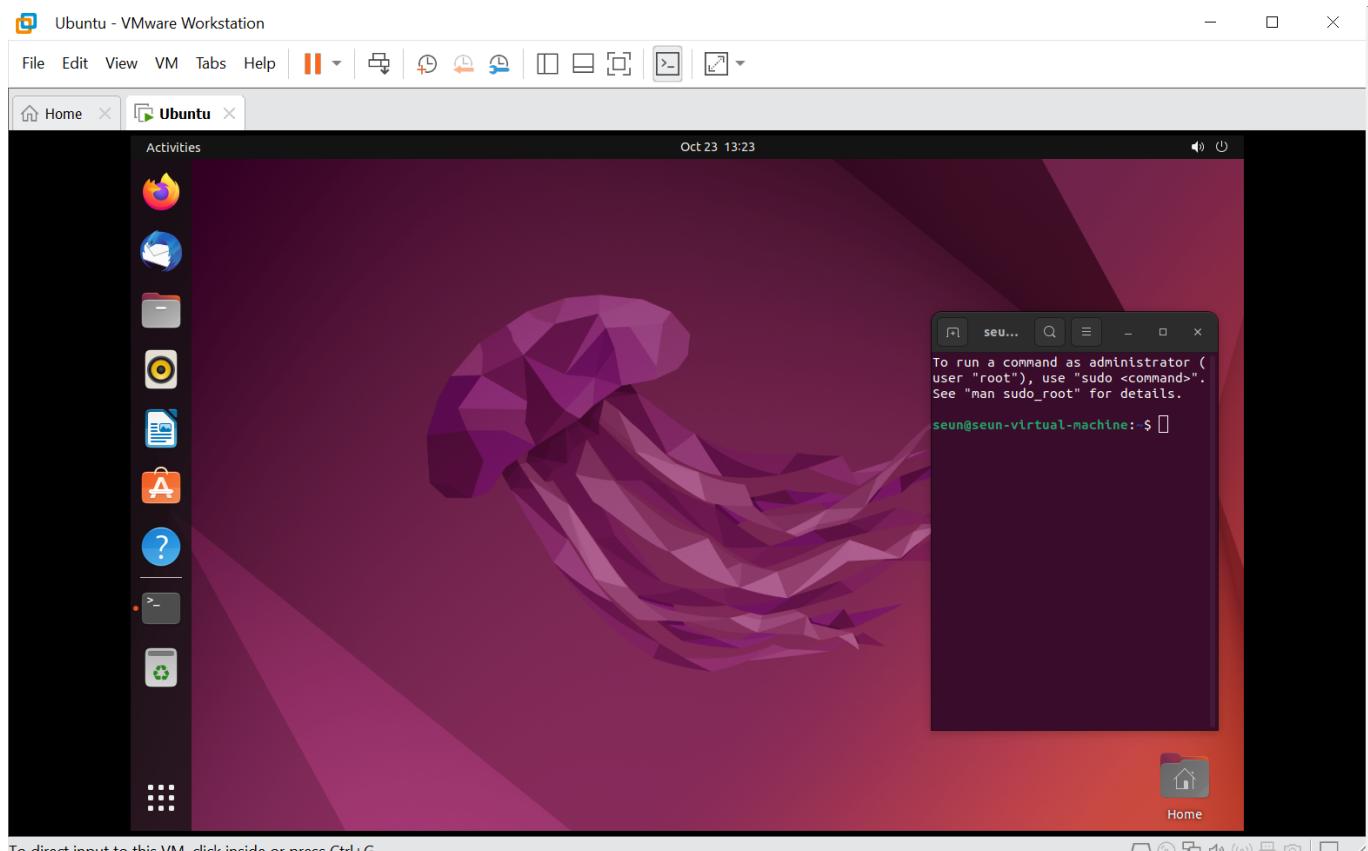
Step 25: Verify Key Status

- Access key status: Active
- Usage: Unused (expected)

Phase 6: AWS CLI Installation (Ubuntu)

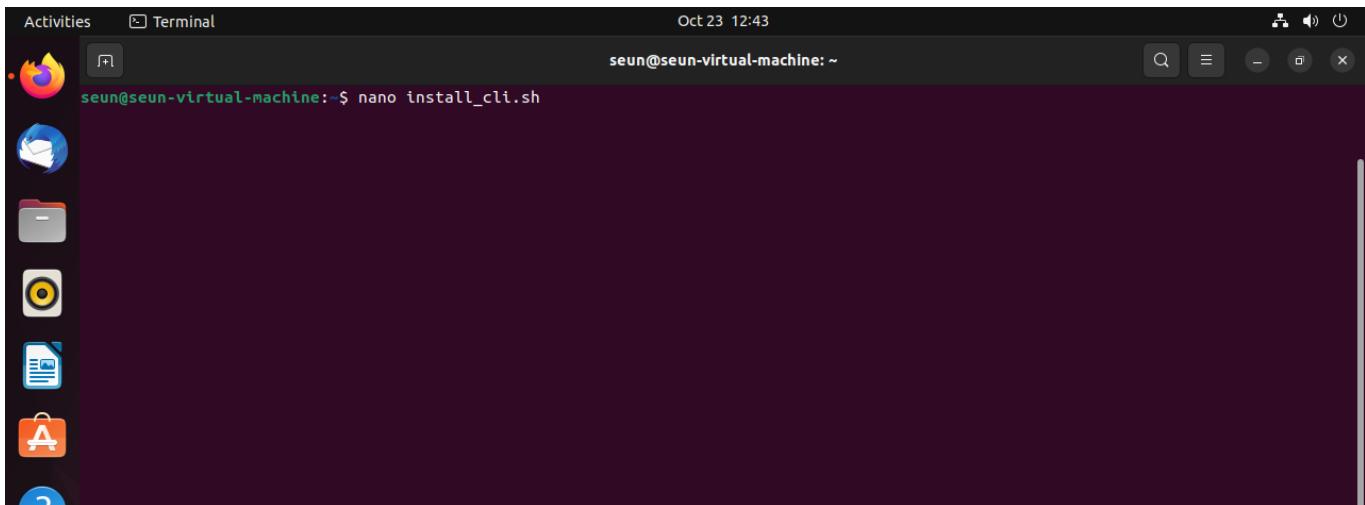
Step 26: Ubuntu Environment Setup

- OS: Ubuntu Linux 22.04
- Terminal ready for CLI installation



Step 27: Create Installation Script

- nano install_cli.sh

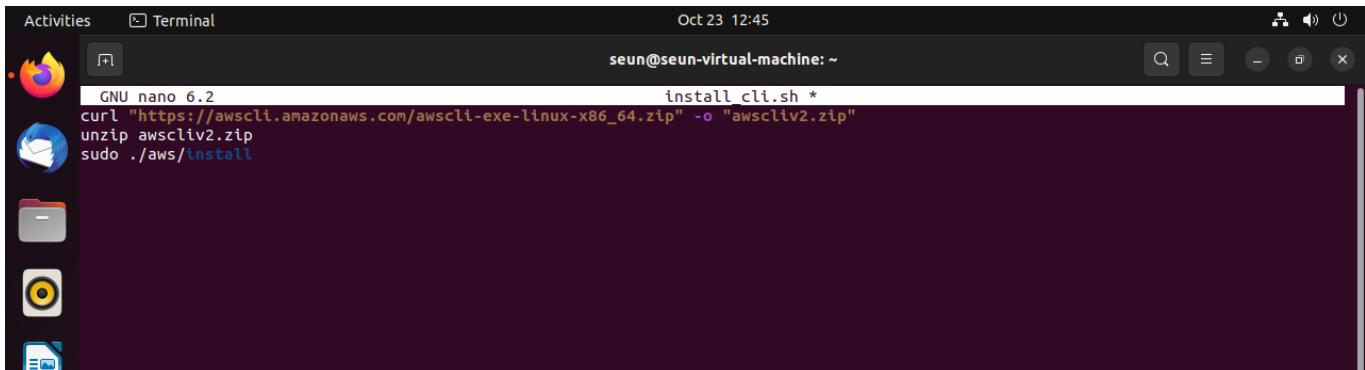


```
Activities Terminal seun@seun-virtual-machine: ~ Oct 23 12:43 seun@seun-virtual-machine: ~ seun@seun-virtual-machine: ~$ nano install_cli.sh
```

Step 28: Write Installation Commands

```
#!/bin/bash
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
```

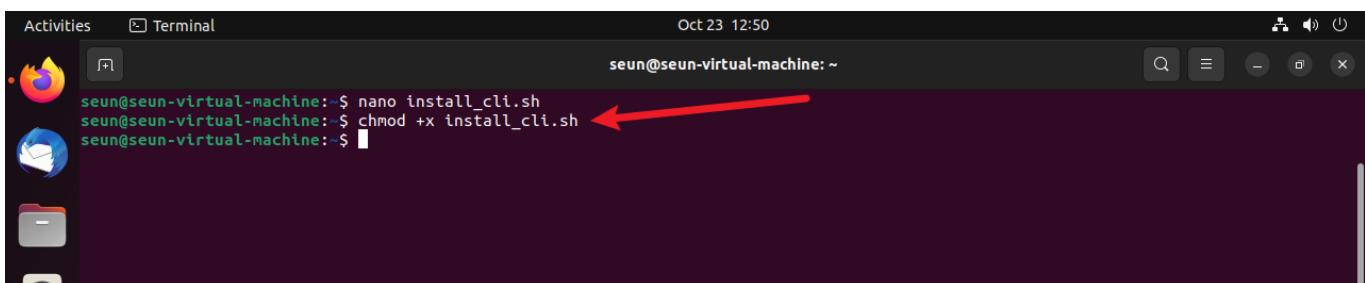
- Save: Ctrl+O, Enter, Ctrl+X



```
Activities Terminal seun@seun-virtual-machine: ~ Oct 23 12:45 seun@seun-virtual-machine: ~ seun@seun-virtual-machine: ~$ nano install_cli.sh *
GNU nano 6.2
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
```

Step 29: Make Script Executable

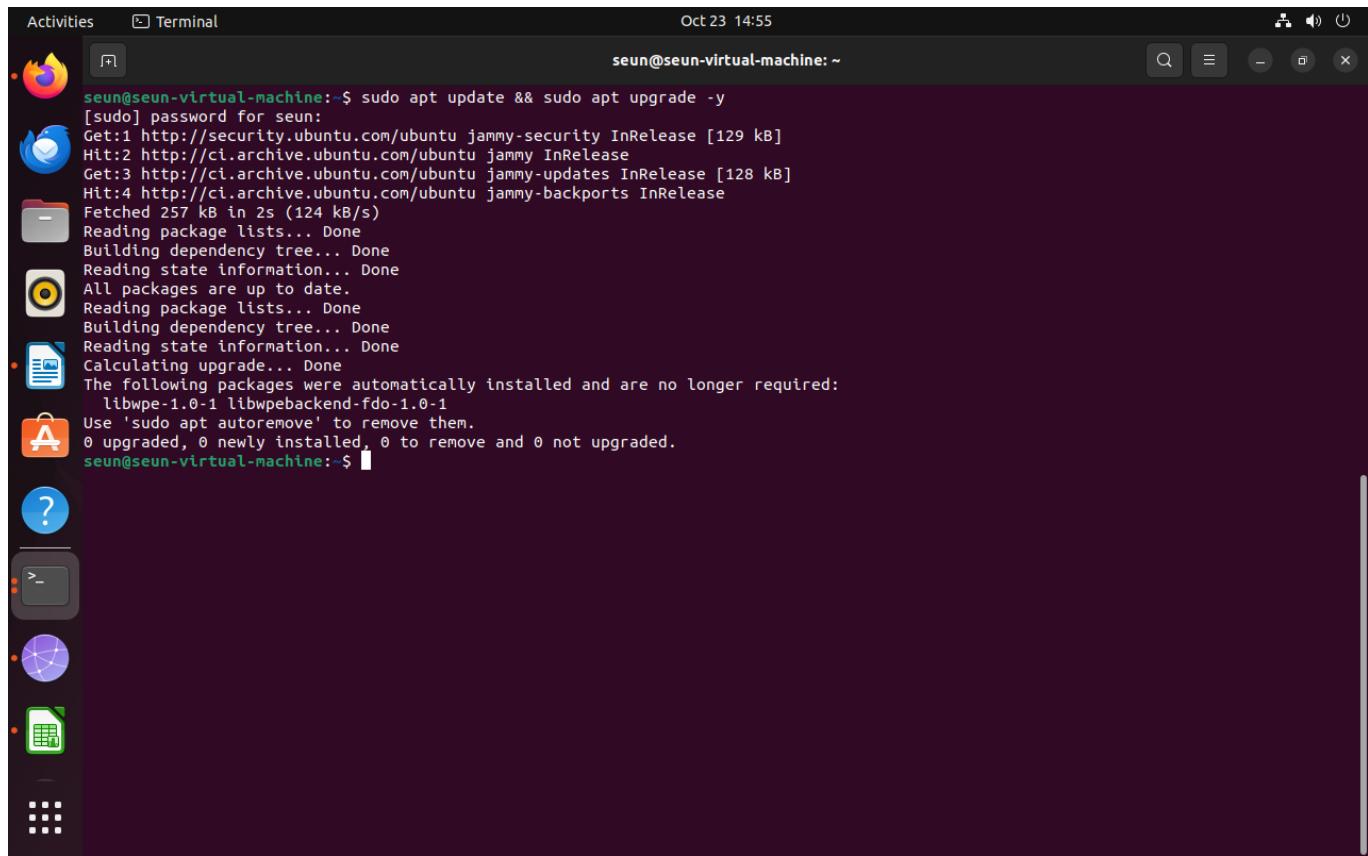
- `chmod +x install_cli.sh`



```
Activities Terminal seun@seun-virtual-machine: ~ Oct 23 12:50 seun@seun-virtual-machine: ~ seun@seun-virtual-machine: ~$ nano install_cli.sh
seun@seun-virtual-machine: ~$ chmod +x install_cli.sh ←
seun@seun-virtual-machine: ~$
```

Step 30: System Update

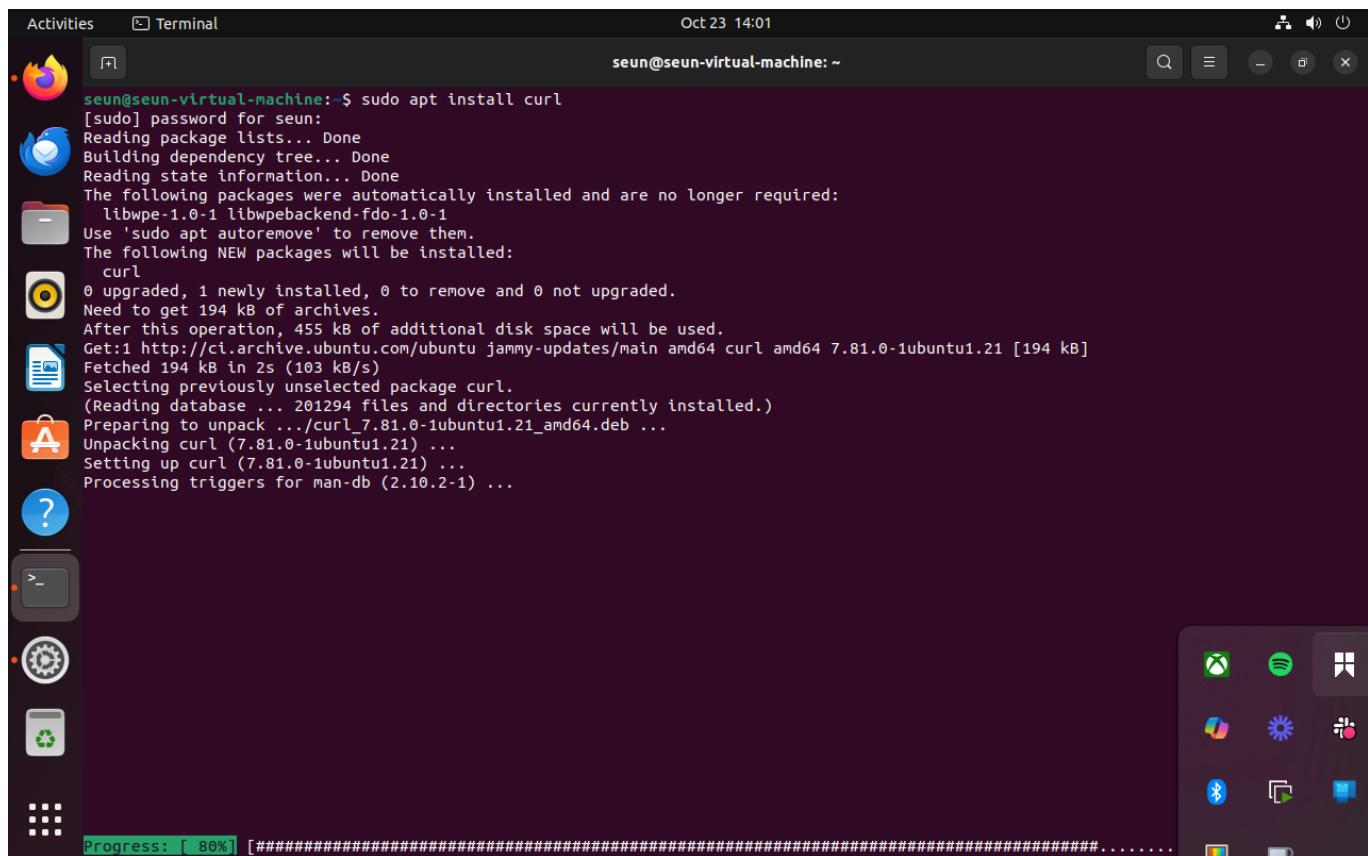
- `sudo apt update && sudo apt upgrade -y`



A screenshot of a Ubuntu desktop environment. A terminal window is open in the top right corner, showing the command `sudo apt update && sudo apt upgrade -y` being run. The output shows the system checking for updates from the security, jammy-security, and jammy-updates repositories, and then upgrading packages. It lists packages like libwpe-1.0-1 and libwpebackend-fdo-1.0-1 as automatically installed and no longer required, with options to remove them. The terminal window has a dark theme and is titled "Activities Terminal". The status bar at the bottom indicates "Oct 23 14:55".

Step 31: Install Curl

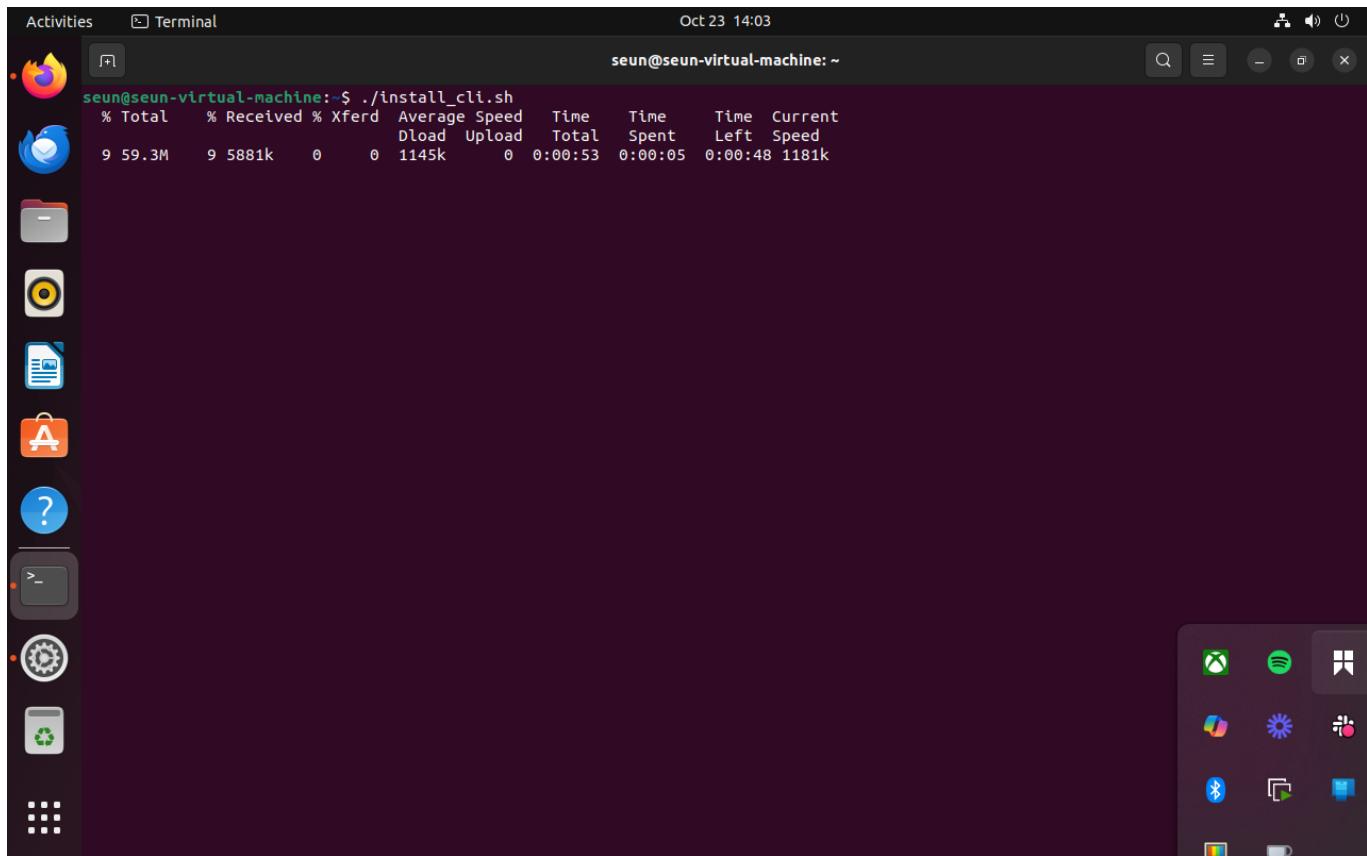
- `sudo apt install curl`



A screenshot of a Ubuntu desktop environment. A terminal window is open in the top right corner, showing the command `sudo apt install curl` being run. The output shows the system reading package lists, building dependency trees, and installing curl. It also lists packages like libwpe-1.0-1 and libwpebackend-fdo-1.0-1 as automatically installed and no longer required, with options to remove them. The terminal window has a dark theme and is titled "Activities Terminal". The status bar at the bottom indicates "Oct 23 14:01". A progress bar at the bottom of the terminal window shows "Progress: [80%] [#####.....]".

Step 32: Execute Installation Script

- `./install_cli.sh`

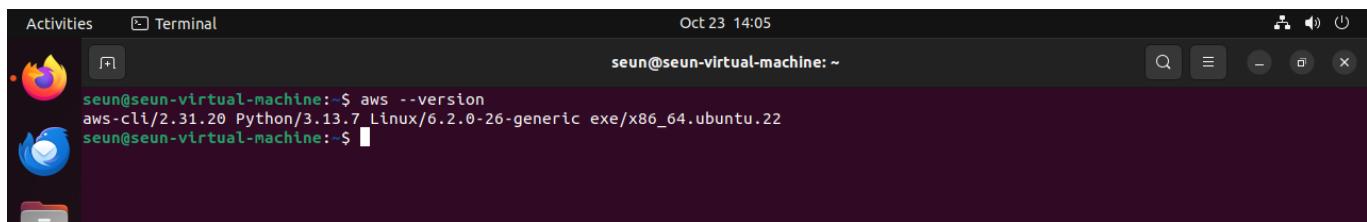


```
Activities Terminal Oct 23 14:03 seun@seun-virtual-machine: ~
seun@seun-virtual-machine:~$ ./install_cli.sh
% Total    % Received % Xferd  Average Speed   Time     Time     Time  Current
          Dload  Upload Total   Spent   Left Speed
9 59.3M    9 5881k    0      0  1145k      0  0:00:53  0:00:05  0:00:48 1181k
```

Step 33: Verify AWS CLI Installation

- `aws --version`

Expected Output: aws-cli/2.15.30 Python/3.11.6 Linux/...

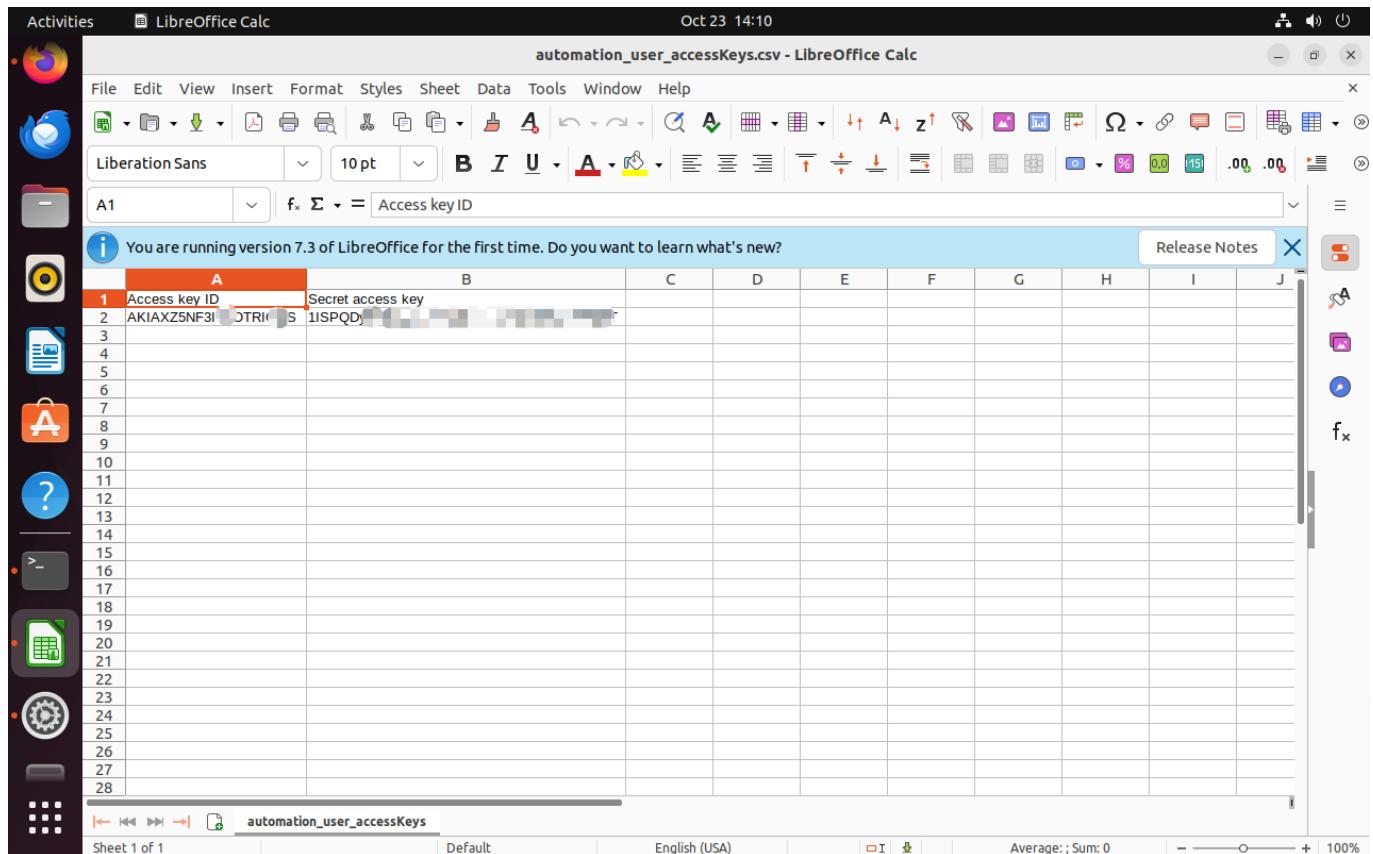


```
Activities Terminal Oct 23 14:05 seun@seun-virtual-machine: ~
seun@seun-virtual-machine:~$ aws --version
aws-cli/2.31.20 Python/3.13.7 Linux/6.2.0-26-generic exe/x86_64/ubuntu.22
seun@seun-virtual-machine:~$ █
```

Phase 7: AWS CLI Configuration & Testing

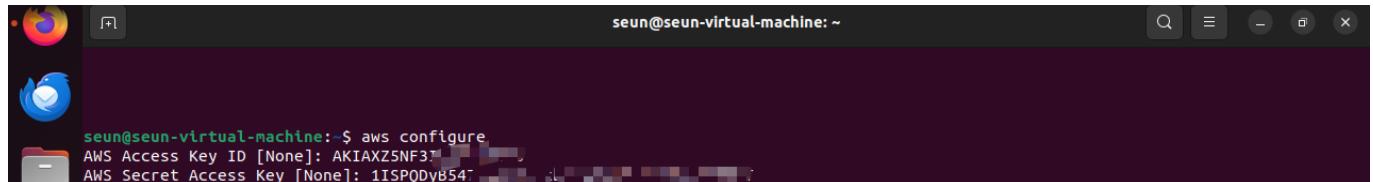
Step 34: Extract Credentials from CSV

- Open downloaded accesskeys.csv in LibreOffice Calc
- Copy Access Key ID and Secret Access Key



Step 35: Configure AWS CLI (Credentials)

- `aws configure`
- AWS Access Key ID [None]: AKIA...
- AWS Secret Access Key [None]: ...



Step 36: Configure Region & Output

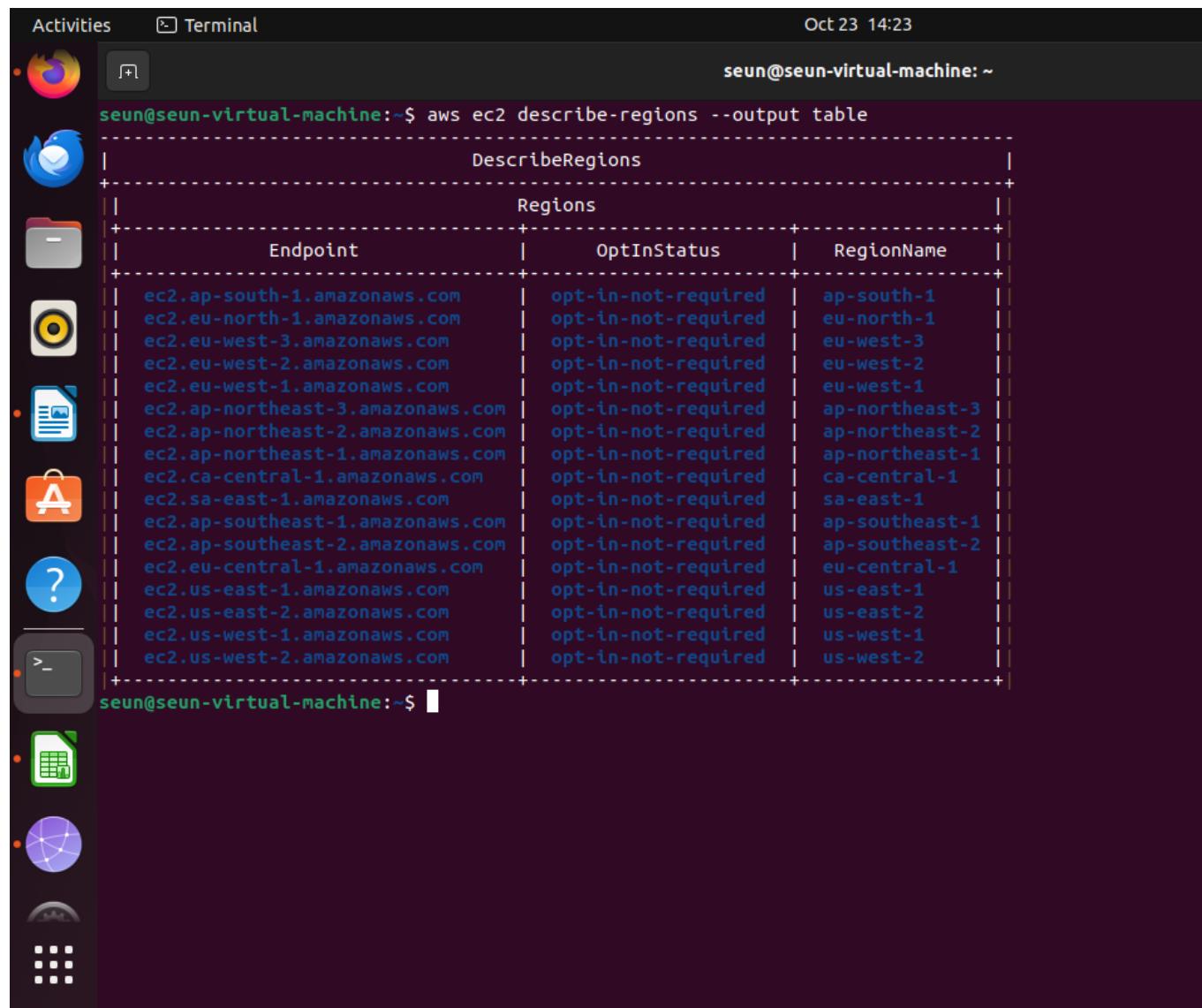
- Default region name [None]: us-east-1
- Default output format [None]: json



Step 37: Test API Connectivity

- `aws ec2 describe-regions --output table`

Success: Complete regions table displayed



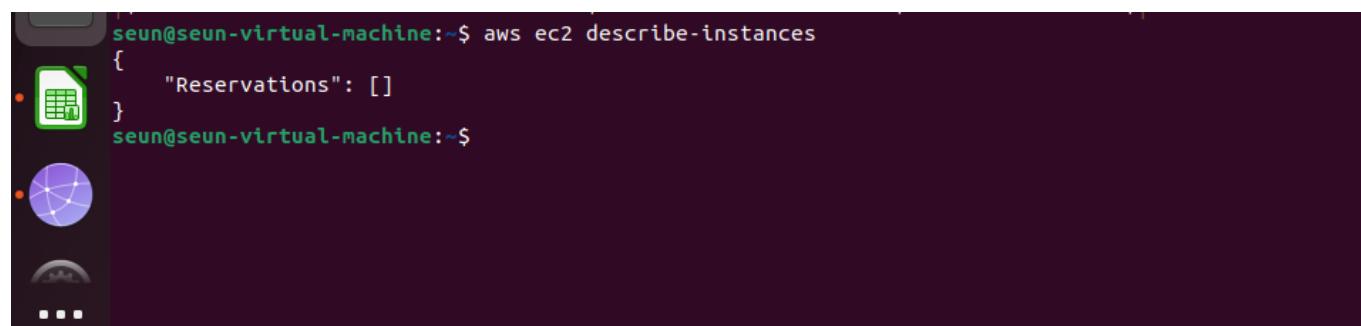
A screenshot of a Linux desktop environment (Ubuntu) showing a terminal window. The terminal window title is "Terminal" and the date is "Oct 23 14:23". The command run is "aws ec2 describe-regions --output table". The output is a table listing 18 AWS Regions:

DescribeRegions		
Regions		
Endpoint	OptInStatus	RegionName
ec2.ap-south-1.amazonaws.com	opt-in-not-required	ap-south-1
ec2.eu-north-1.amazonaws.com	opt-in-not-required	eu-north-1
ec2.eu-west-3.amazonaws.com	opt-in-not-required	eu-west-3
ec2.eu-west-2.amazonaws.com	opt-in-not-required	eu-west-2
ec2.eu-west-1.amazonaws.com	opt-in-not-required	eu-west-1
ec2.ap-northeast-3.amazonaws.com	opt-in-not-required	ap-northeast-3
ec2.ap-northeast-2.amazonaws.com	opt-in-not-required	ap-northeast-2
ec2.ap-northeast-1.amazonaws.com	opt-in-not-required	ap-northeast-1
ec2.ca-central-1.amazonaws.com	opt-in-not-required	ca-central-1
ec2.sa-east-1.amazonaws.com	opt-in-not-required	sa-east-1
ec2.ap-southeast-1.amazonaws.com	opt-in-not-required	ap-southeast-1
ec2.ap-southeast-2.amazonaws.com	opt-in-not-required	ap-southeast-2
ec2.eu-central-1.amazonaws.com	opt-in-not-required	eu-central-1
ec2.us-east-1.amazonaws.com	opt-in-not-required	us-east-1
ec2.us-east-2.amazonaws.com	opt-in-not-required	us-east-2
ec2.us-west-1.amazonaws.com	opt-in-not-required	us-west-1
ec2.us-west-2.amazonaws.com	opt-in-not-required	us-west-2

Step 38: Test EC2 Instance Listing

- aws ec2 describe-instances

Success: Empty response (no instances = correct)



A screenshot of a Linux desktop environment (Ubuntu) showing a terminal window. The terminal window title is "Terminal" and the date is "Oct 23 14:23". The command run is "aws ec2 describe-instances". The output is an empty JSON object:

```
seun@seun-virtual-machine:~$ aws ec2 describe-instances
{
    "Reservations": []
}
seun@seun-virtual-machine:~$
```

Phase 8: Prepare Automation Script and Resources

Step 39: Create Automation Script File

- Create the shell script file using `touch aws-resources.sh`

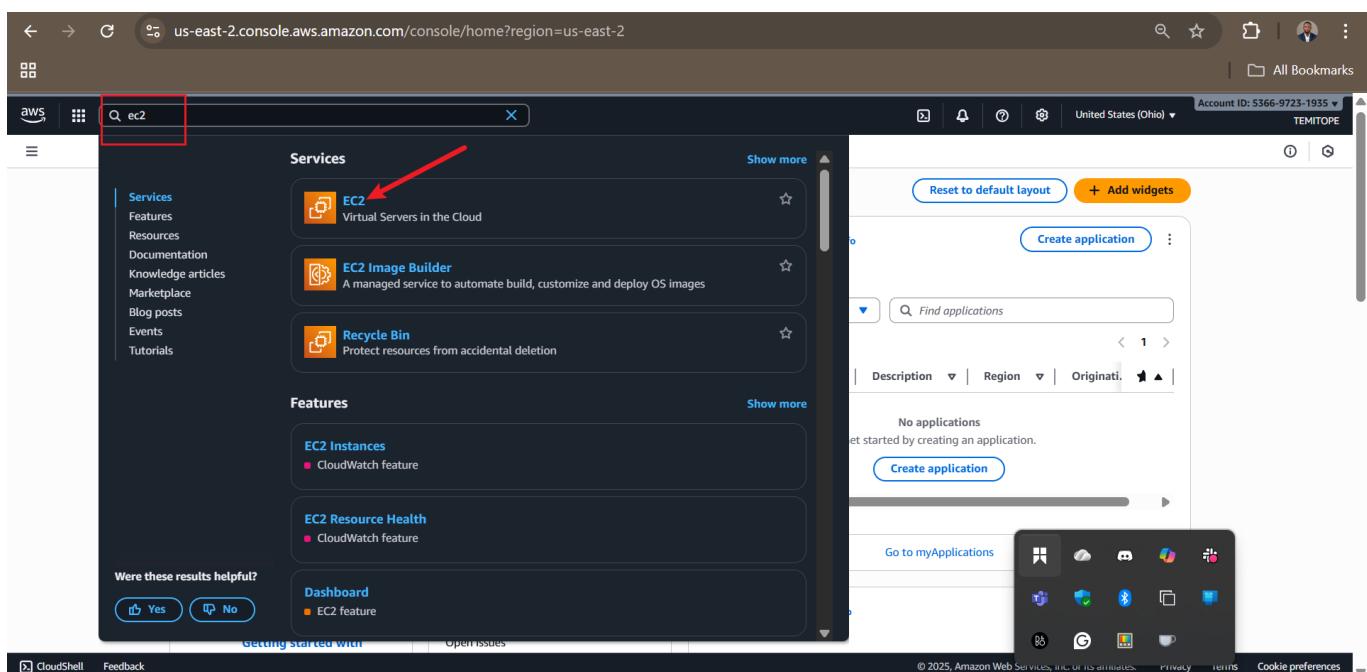
Purpose: This script will automate EC2 instance creation and S3 bucket management.

```
Activities Terminal Oct 23 15:39
seun@seun-virtual-machine:~$ touch aws-resources.sh
seun@seun-virtual-machine:~$
```

Step 49: Navigate to EC2 Service

- Use the search bar to search for "EC2" and click on it

Purpose: Access EC2 dashboard for key pair creation.



Step 50: Access Key Pairs Section

- Find "Key Pairs" under "Network and Security" on the left sidebar and click on it

Purpose: Manage key pairs for secure SSH access to EC2 instances.

The screenshot shows the AWS EC2 home page. On the left sidebar, under the 'Network & Security' section, the 'Key Pairs' option is highlighted with a red arrow. The main content area features the heading 'Amazon Elastic Compute Cloud (EC2)' and the sub-headline 'Create, manage, and monitor virtual servers in the cloud.' Below this, there's a section titled 'Benefits and features' with a sub-section 'EC2 offers ultimate scalability and control'. A call-to-action button 'Launch instance' is visible on the right.

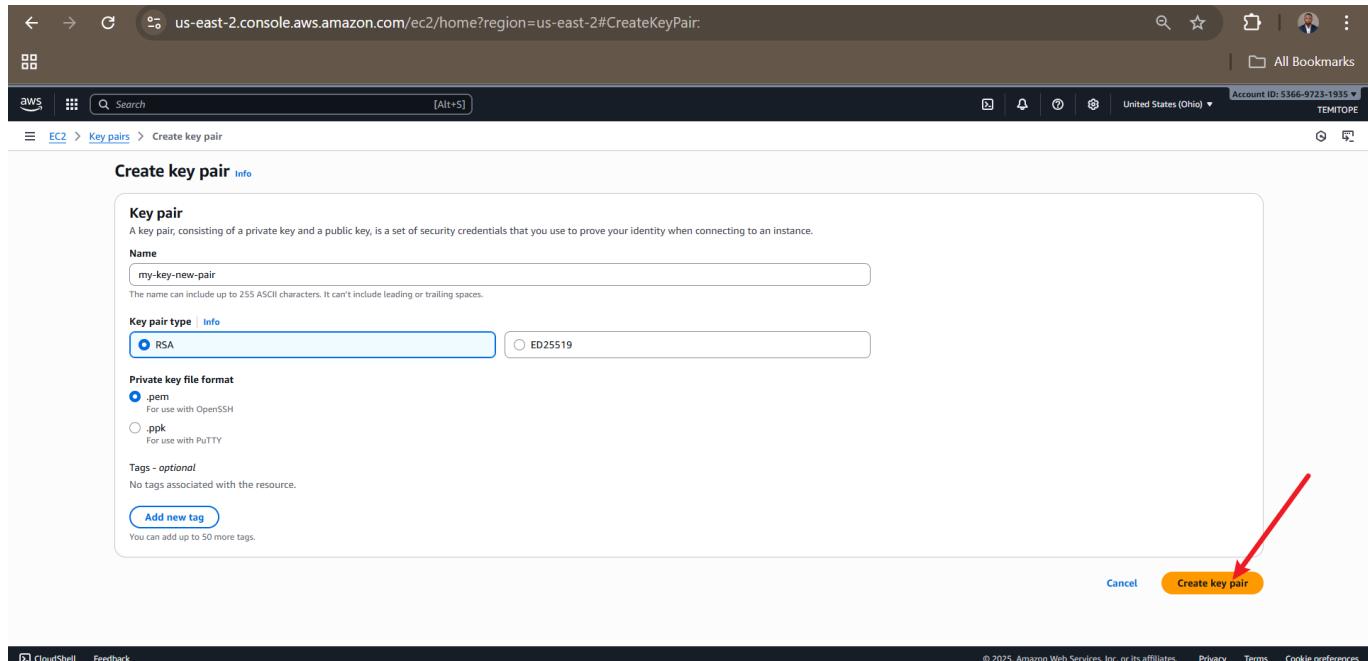
Step 51: Initiate Key Pair Creation

- On the Key Pairs page, click "Create key pair"

The screenshot shows the 'Key pairs' page in the AWS EC2 console. The left sidebar shows the 'Key Pairs' option under the 'Network & Security' section. In the main content area, there is a table titled 'Key pairs (1)'. At the top right of the table, there is a 'Create key pair' button, which is highlighted with a red arrow. The table lists one key pair named 'temskey1' with details: Type 'rsa', Created '2025/01/30 20:31 GMT+1', Fingerprint '81:e5:6b:d4:8e:a7:ab:20:3c:14:4c:36:cde...', and ID 'key-02d680261a5cf5da'.

Step 52: Configure and Create Key Pair

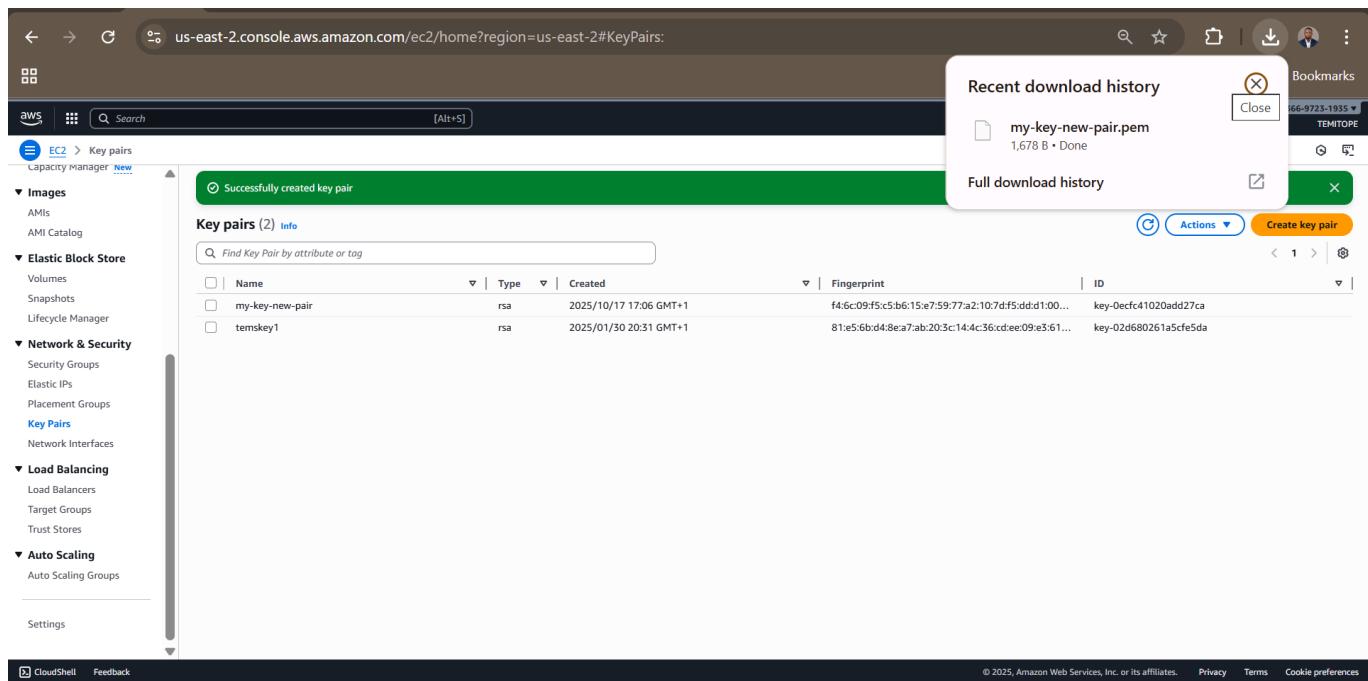
- Name the key pair (e.g., MyKeyPair)
- Select key type (e.g., RSA) and file format (e.g., .pem)
- Click "Create key pair"



Step 53: Verify Key Pair Creation

- Key pair created and .pem file downloaded

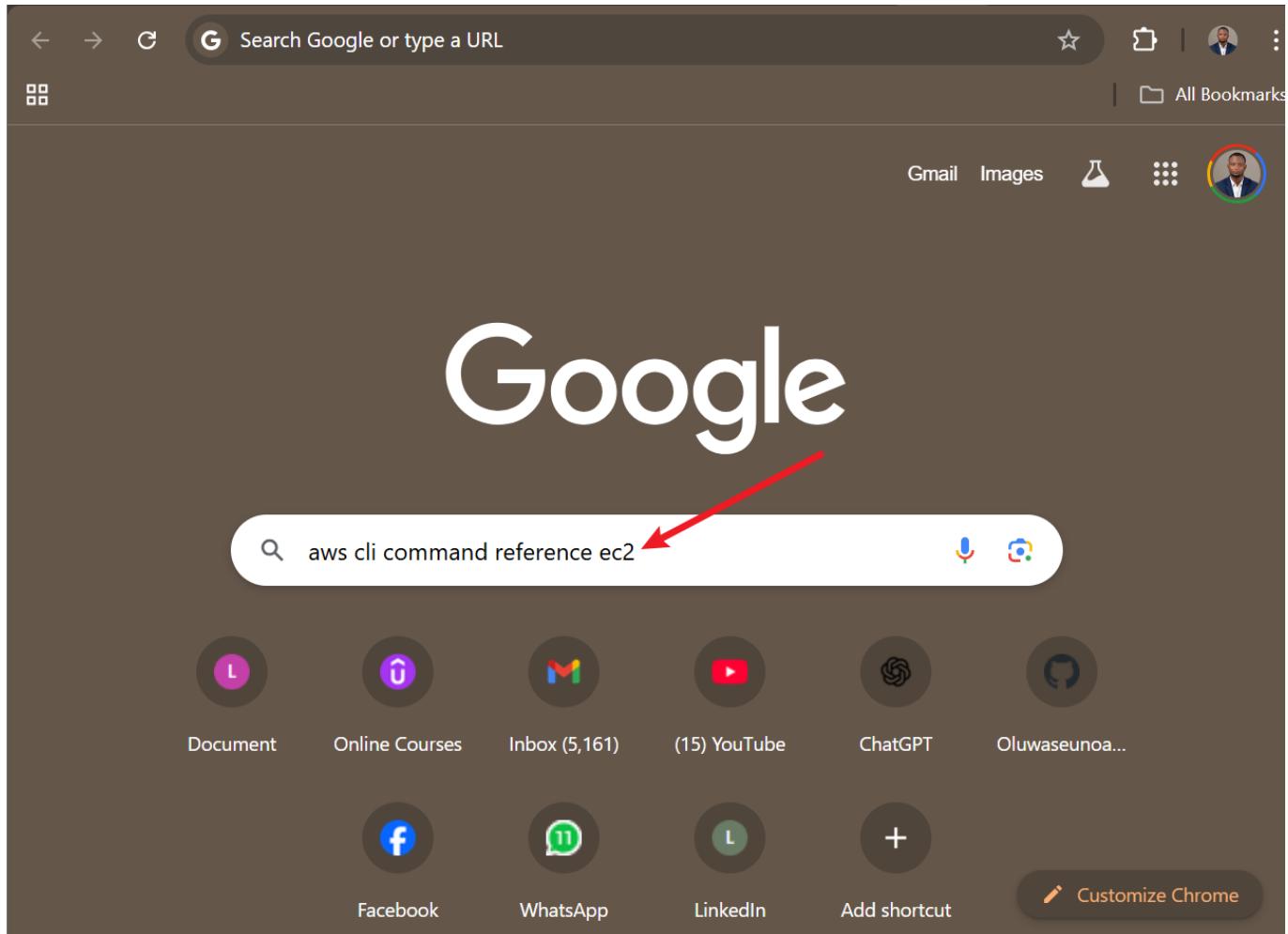
Purpose: Use this key pair in the automation script for EC2 launches.



Step 54: Search for AWS CLI EC2 Reference

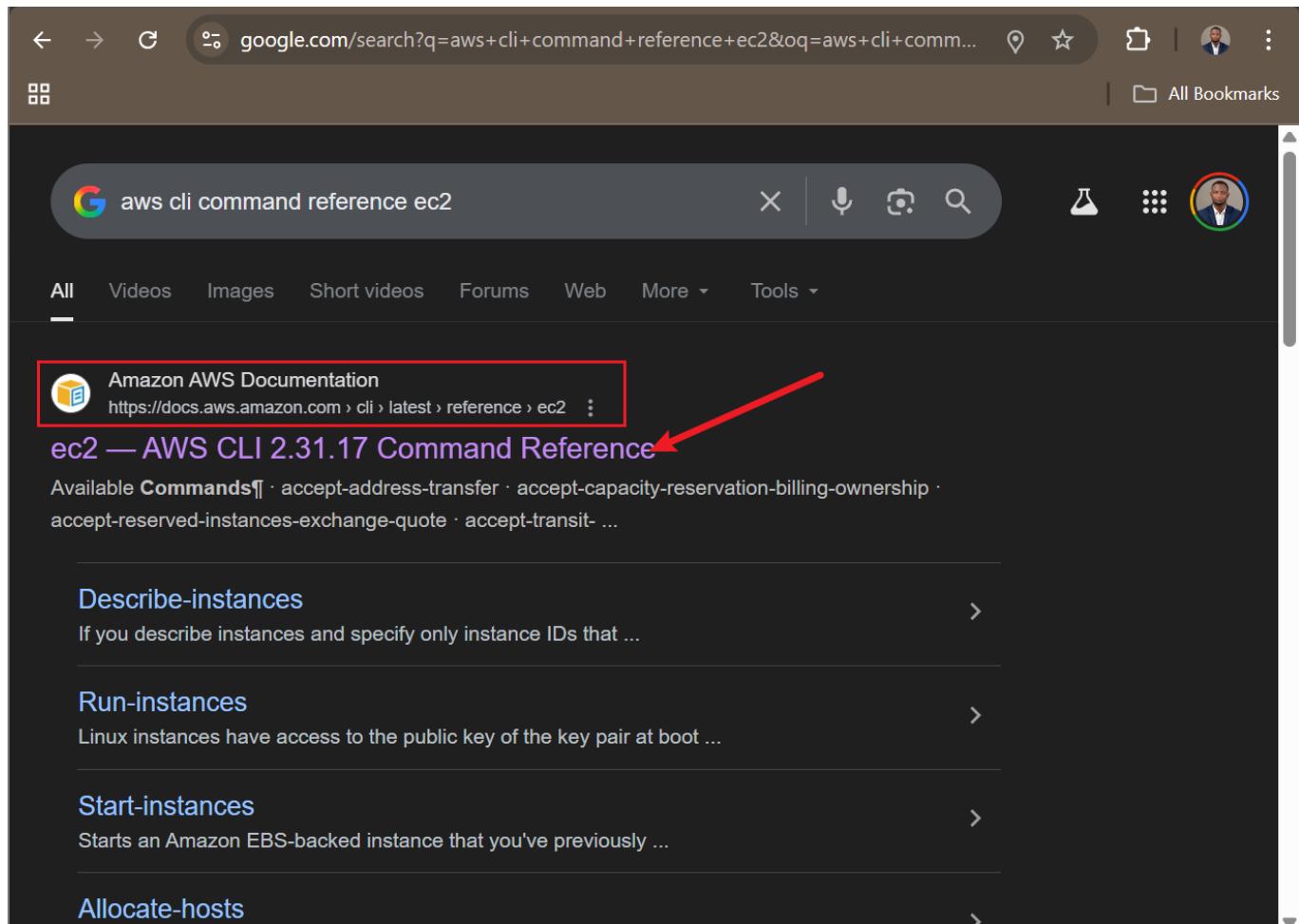
- In your preferred browser, search for "aws cli command reference ec2"

Purpose: Reference official documentation for scripting EC2 commands.



Step 55: Access CLI Reference

- Click on the first link for CLI latest reference (EC2 section)



A screenshot of a web browser showing search results for "aws cli command reference ec2". The results page has a dark background. A red arrow points to the first result, which is a link to the "ec2 — AWS CLI 2.31.17 Command Reference" documentation page. The link is highlighted with a red box.

Amazon AWS Documentation
https://docs.aws.amazon.com / cli / latest / reference / ec2 ...

ec2 — AWS CLI 2.31.17 Command Reference

Available Commands | accept-address-transfer · accept-capacity-reservation-billing-ownership · accept-reserved-instances-exchange-quote · accept-transit- ...

Describe-instances >
If you describe instances and specify only instance IDs that ...

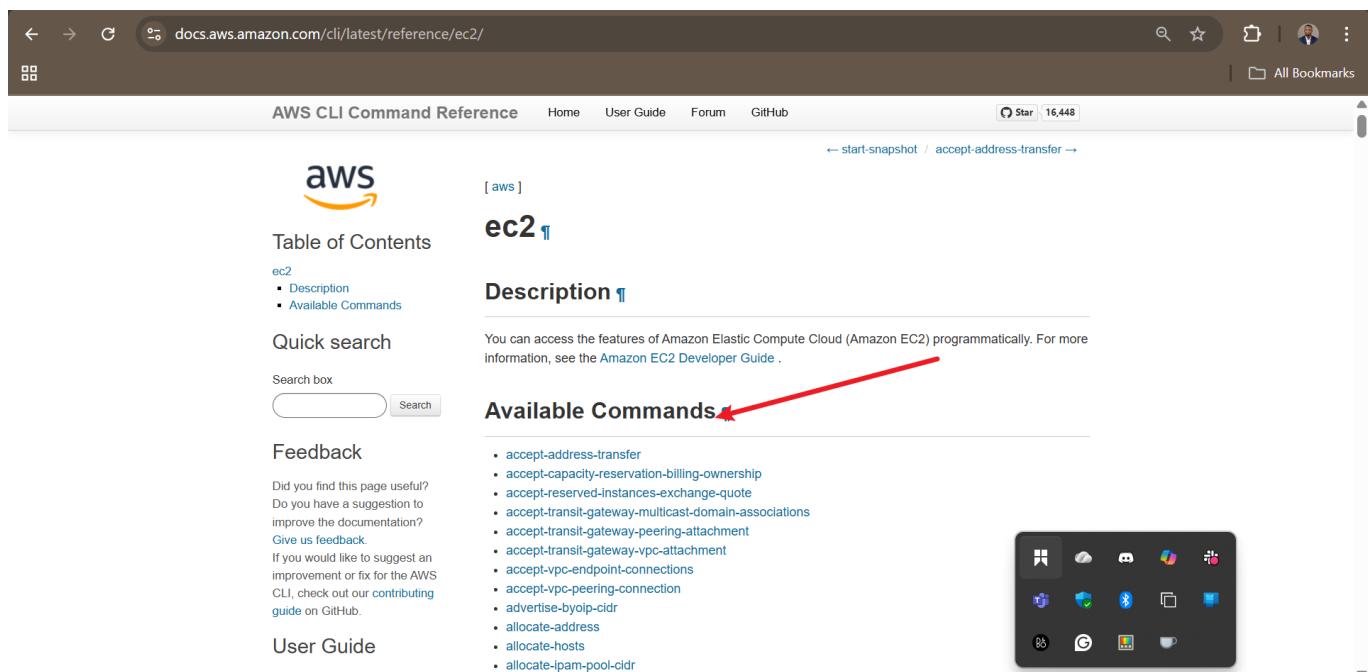
Run-instances >
Linux instances have access to the public key of the key pair at boot ...

Start-instances >
Starts an Amazon EBS-backed instance that you've previously ...

Allocate-hosts >

Step 56: Explore Documentation

- Now on the AWS CLI commands official documentation webpage



A screenshot of the AWS CLI Command Reference documentation for the ec2 service. The page has a white background. A red arrow points to the "Available Commands" section, which lists various AWS CLI commands for EC2. The "Available Commands" section is highlighted with a red box.

AWS CLI Command Reference

Table of Contents

ec2

- Description
- Available Commands

Quick search

Search box

Feedback

Did you find this page useful?
Do you have a suggestion to improve the documentation?
Give us feedback.
If you would like to suggest an improvement or fix for the AWS CLI, check out our contributing guide on GitHub.

User Guide

First time using the AWS CLI?

[aws]

ec2

Description

You can access the features of Amazon Elastic Compute Cloud (Amazon EC2) programmatically. For more information, see the [Amazon EC2 Developer Guide](#).

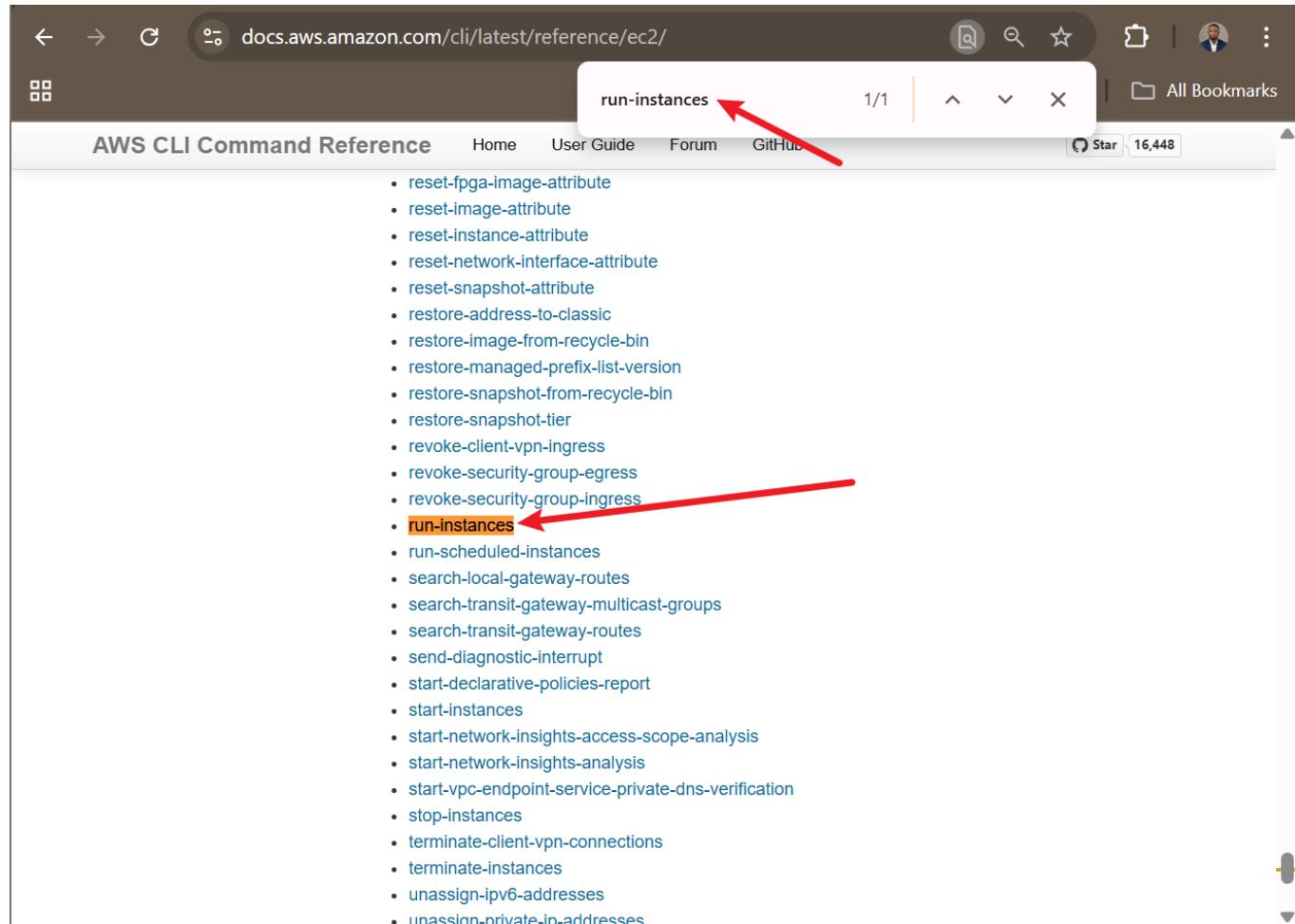
Available Commands

- accept-address-transfer
- accept-capacity-reservation-billing-ownership
- accept-reserved-instances-exchange-quote
- accept-transit-gateway-multicast-domain-associations
- accept-transit-gateway-peering-attachment
- accept-transit-gateway-vpc-attachment
- accept-vpc-endpoint-connections
- accept-vpc-peering-connection
- advertise-byoi-p-cidr
- allocate-address
- allocate-hosts
- allocate-lpm-pool-cidr

Step 57: Find run-instances Command

- Search for "run-instances" using Ctrl+F and click on it when found

Purpose: Use this command in the script for creating EC2 instances.



Phase 9: Develop the Automation Shell Script

Step 58: Prepare Script for Editing

- Create a new file called 'aws-resources.sh'
- Make aws-resources.sh executable
- Open in VS Code (or your code editor)
-

```
touch aws-resources.sh
chmod +x aws-resources.sh
code aws-resources.sh
```

Purpose: Begin building the automation script incrementally.

A screenshot of a terminal window titled "seun@seun-virtual-machine: ~". The terminal shows the following commands being run:

```
seun@seun-virtual-machine:~$ touch aws-resources.sh
seun@seun-virtual-machine:~$ chmod +x aws-resources.sh
seun@seun-virtual-machine:~$ code aws-resources.sh
```

Red arrows point from the right side of the image to the terminal output:

- An arrow points to the first command "touch aws-resources.sh" with the text "create aws-resources.sh".
- An arrow points to the second command "chmod +x aws-resources.sh" with the text "turn aws-resources.sh to an executable".
- An arrow points to the third command "code aws-resources.sh" with the text "open aws-resources.sh in vscode".

Step 59: Add Shebang and ENVIRONMENT Variable

Add the following to the script:

```
#!/bin/bash

# Environment variables
ENVIRONMENT=$1"
```

- Run `./aws-resources.sh` to test basic execution (no output expected yet)

A screenshot of the Visual Studio Code interface. On the left is the sidebar with various icons. In the center, there's a code editor with a file named "aws-resources.sh" open. The code contains the following:

```
1 #!/bin/bash
2
3 # Environment variables
4 ENVIRONMENT=$1"
```

The terminal tab at the bottom shows the script being run:

```
seun@seun-virtual-machine:~$ ./aws-resources.sh
```

Step 60: Add Argument Check Function

Add the function and call it:

```

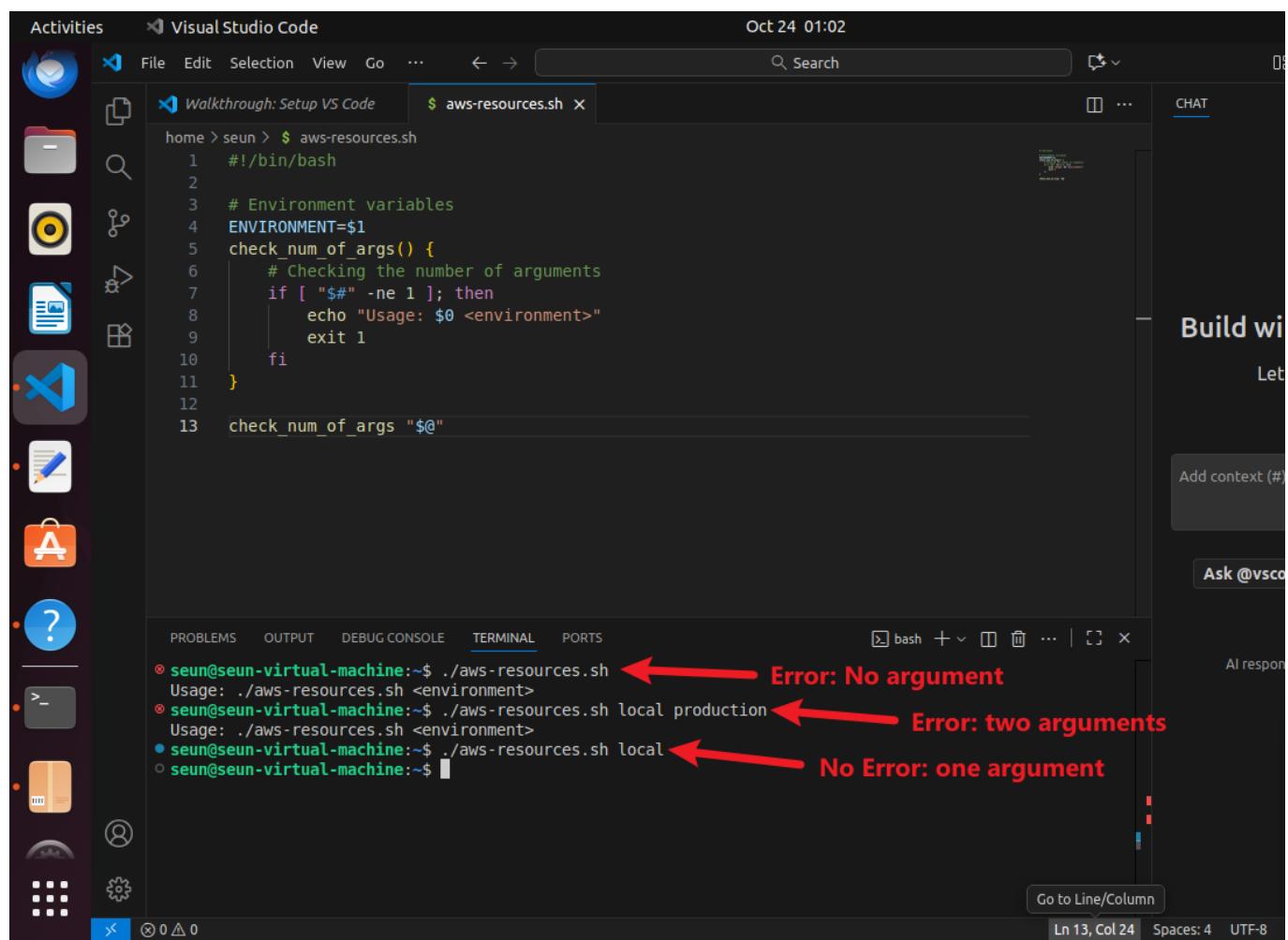
check_num_of_args() {
    if [ $# -ne 1 ]; then
        echo "Usage: $0 <environment>"
        exit 1
    fi
}

check_num_of_args "$@"

```

- Run without argument (error expected)
- Run with argument (e.g., `./aws-resources.sh local`) (success)

Purpose: Ensure script is called with environment parameter.



The screenshot shows a Visual Studio Code interface with a terminal window open. The terminal shows the following bash session:

```

home > seun > $ aws-resources.sh
1 #!/bin/bash
2
3 # Environment variables
4 ENVIRONMENT=$1
5 check_num_of_args()
6     # Checking the number of arguments
7     if [ $# -ne 1 ]; then
8         echo "Usage: $0 <environment>"
9         exit 1
10    fi
11 }
12
13 check_num_of_args "$@"

```

Red arrows point from the right side of the image to specific lines in the terminal output:

- An arrow points to the first line of output: **Error: No argument**
- An arrow points to the second line of output: **Error: two arguments**
- An arrow points to the third line of output: **No Error: one argument**

Step 61: Add Environment Activation Function

Add the function and call it:

```

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  
}  
activate_infra_environment
```

```
11 }  
12  
13 activate_infra_environment() {  
14     # Acting based on the argument value  
15     if [ "$ENVIRONMENT" == "local" ]; then  
16         echo "Running script for Local Environment..."  
17     elif [ "$ENVIRONMENT" == "testing" ]; then  
18         echo "Running script for Testing Environment..."  
19     elif [ "$ENVIRONMENT" == "production" ]; then  
20         echo "Running script for Production Environment..."  
21     else  
22         echo "Invalid environment specified. Please use 'local', 'testing', or  
23         'production'."  
24     exit 2  
25 fi  
26 }  
27  
28 check_num_of_args "$@"  
29 activate_infra_environment
```

Step 62: Test Environment Activation

- Run with invalid environment (error expected) `./aws-resources.sh invalid`

- Run with "local" environment (success) `./aws-resources.sh local`

The screenshot shows a Visual Studio Code interface. On the left is a sidebar with various icons. The main area has a terminal tab open with the command `$ aws-resources.sh`. The code editor shows a script named `aws-resources.sh` with several conditional statements for environment variables. The terminal output shows two runs of the script: one with an invalid argument ('invalid') which fails with an error message, and one with the correct argument ('local') which succeeds with a message. Red arrows point from the error message to the text 'Error: "invalid" environment was passed' and from the success message to the text 'No Error: "local" environment was passed'.

```
11 }
12
13     activate_infra_environment() {
14         # Acting based on the argument value
15         if [ "$ENVIRONMENT" == "local" ]; then
16             echo "Running script for Local Environment..."
17         elif [ "$ENVIRONMENT" == "testing" ]; then
18             echo "Running script for Testing Environment..."
19         elif [ "$ENVIRONMENT" == "production" ]; then
20             echo "Running script for Production Environment..."
21         else
22             echo "Invalid environment specified. Please use 'local', 'testing', or
23             'production'."
24         exit 2
25     fi
26 }
27
28 check_num_of_args "$@"
29 activate_infra_environment

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
① seun@seun-virtual-machine:~/Documents$ ./aws-resources.sh invalid
Invalid environment specified. Please use 'local', 'testing', or
'production'.
② seun@seun-virtual-machine:~/Documents$ ./aws-resources.sh local
Running script for Local Environment...
③ seun@seun-virtual-machine:~/Documents$ 
```

Step 63: Add AWS CLI Check Function

Add the function and call it:

```
# 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."
        return 1
    fi
}
check_aws_cli || exit 1
```

```

21 else
22 echo "Invalid environment specified. Please use 'local', 'testing', or
23 'production'."
24 exit 2
25 fi
26 }
27
28 # Function to check if AWS CLI is installed
29 check_aws_cli() {
30     if ! command -v aws >& /dev/null; then
31         echo "AWS CLI is not installed. Please install it before proceeding."
32         return 1
33     fi
34 }
35
36
37 check_num_of_args "$@"
38 activate_infra_environment
39 check_aws_cli || exit 1

```

Step 64: Run AWS CLI Check

- Execute the script; no error as CLI is installed `./aws-resources.sh local`

```

21 else
22 echo "Invalid environment specified. Please use 'local', 'testing', or
23 'production'."
24 exit 2
25 fi
26 }
27
28 # Function to check if AWS CLI is installed
29 check_aws_cli() {
30     if ! command -v aws >& /dev/null; then
31         echo "AWS CLI is not installed. Please install it before proceeding."
32         return 1
33     fi
34 }
35
36
37 check_num_of_args "$@"
38 activate_infra_environment
39 check_aws_cli || exit 1

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

● seun@seun-virtual-machine:~\$./aws-resources.sh local
○ Running script for Local Environment...
○ seun@seun-virtual-machine:~\$ █

Step 65: Add AWS Profile Check Function

Add the function and call it:

```

# Function to check if AWS profile is set
check_aws_profile() {
    if [ -z "$AWS_PROFILE" ]; then
        echo "AWS profile environment variable is not set."
        return 1
    fi
}

```

```
}
```

The screenshot shows a Visual Studio Code interface with a terminal window open. The terminal title is "aws-resources.sh" and the command run is "\$ aws-resources.sh". The terminal content displays a shell script for AWS CLI setup. A red box highlights the function to check if an AWS profile is set, and a red arrow points to the final command "check_aws_profile || exit 1" on line 47.

```
29  check_aws_cli() {
30      if ! command -v aws >& /dev/null; then
31          echo "AWS CLI is not installed. Please install it before proceeding."
32          return 1
33      fi
34  }
35
36 # Function to check if AWS profile is set
37 check_aws_profile() {
38     if [ -z "$AWS_PROFILE" ]; then
39         echo "AWS profile environment variable is not set."
40         return 1
41     fi
42 }
43
44 check_num_of_args "$@"
45 activate_infra_environment
46 check_aws_cli || exit 1
47 check_aws_profile || exit 1
48
```

Step 66: Test AWS Profile Check

- Run script: `./aws-resources.sh local` (error if not set)
 - Export `AWS_PROFILE=default`
 - Run again: `./aws-resources.sh local` (success)

The screenshot shows a Visual Studio Code interface with the following details:

- Activity Bar:** On the left, there are icons for File Explorer, Search, Problems, Taskbar, Terminal, and Help.
- Terminal:** The bottom panel displays a terminal session:
 - Seun's local environment: "seun@seun-virtual-machine:~\$./aws-resources.sh local".
 - Output: "Running script for Local Environment..."
 - Warning: "AWS profile environment variable is not set."
 - Seun exports the default profile: "seun@seun-virtual-machine:~\$ export AWS_PROFILE=default".
 - Seun runs the script again: "seun@seun-virtual-machine:~\$./aws-resources.sh local".
 - Output: "Running script for Local Environment..."
 - Success: "No error: Default profile now found".
- Code Editor:** The main panel shows a file named "aws-resources.sh" with the following content:

```
home > seun > $ aws-resources.sh
37 check_aws_profile() {
38     if [ -z "$AWS_PROFILE" ]; then
39         return 1
40     fi
41 }
42 }
43
44 check_num_of_args "$@"
45 activate_infra_environment
46 check_aws_cli || exit 1
47 check_aws_profile || exit 1
48
```
- Right Panel:** A "Build with agent mode" section with a "Let's get started" button and a "Add context (#), extensions (@), commands" input field.
- Bottom Right:** A system tray with icons for Spotify, Xbox, Microsoft Edge, and other system status indicators.

Step 67: Add EC2 Instance Creation Function

Add the function and call it:

```
# Function to create EC2 Instances
create_ec2_instances() {

    # Specify the parameters for the EC2 instances
    instance_type="t2.micro"
    ami_id="resolve:ssm:/aws/service/ami-amazon-linux-latest/al2023-ami-kernel-
default-x86_64"
    count=2
    region="us-east-2"
    subnet_id="subnet-<pseudo-id:Enter yours>"          # Replace with your public
subnet ID
    security_group_id="sg-<pseudo-id:Enter yours>"      # Replace with your SG ID
    key_name="my-key-new-pair"

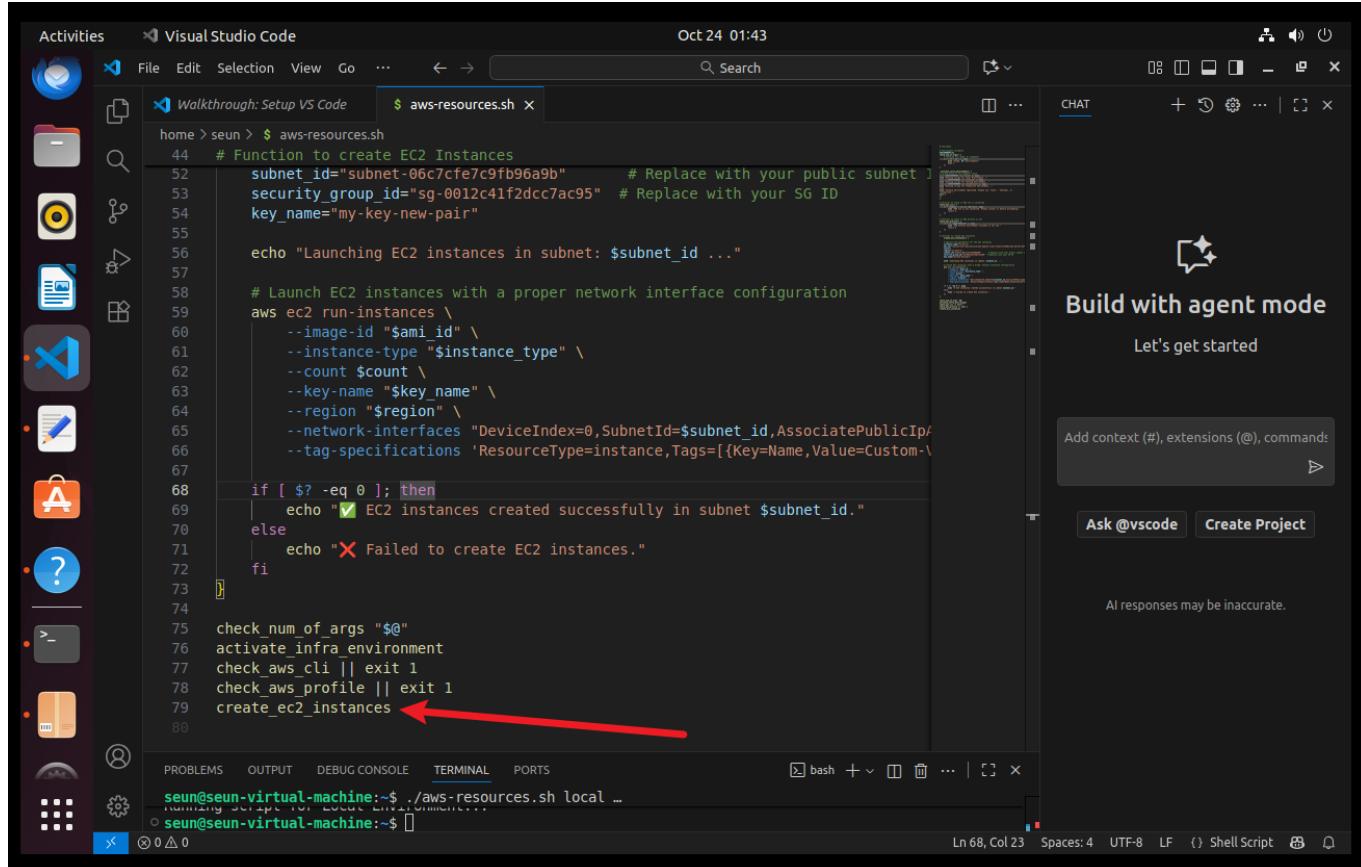
    echo "Launching EC2 instances in subnet: $subnet_id ..."

    # Launch EC2 instances with a proper network interface configuration
    aws ec2 run-instances \
        --image-id "$ami_id" \
        --instance-type "$instance_type" \
        --count $count \
        --key-name "$key_name" \
        --region "$region" \
        --network-interfaces
    "DeviceIndex=0,SubnetId=$subnet_id,AssociatePublicIpAddress=true,Groups=$security_
group_id" \
        --tag-specifications 'ResourceType=instance,Tags=[{Key=Name,Value=Custom-
VPC-Instance}]'

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

create_ec2_instances
```

Note: Adjust parameters like AMI ID, key name, and instance profile as per your setup. This example uses SSM to fetch a dynamic AMI ID.



```

Activities Visual Studio Code Oct 24 01:43
File Edit Selection View Go ... Search CHAT
Walkthrough: Setup VS Code $ aws-resources.sh
home > seu > $ aws-resources.sh
44 # Function to create EC2 Instances
52     subnet_id="subnet-06c7cfec9fb96a9b"      # Replace with your public subnet ID
53     security_group_id="sg-0012c41f2dcc7ac95" # Replace with your SG ID
54     key_name="my-key-new-pair"
55
56     echo "Launching EC2 instances in subnet: $subnet_id ..."
57
58     # Launch EC2 instances with a proper network interface configuration
59     aws ec2 run-instances \
60         --image-id "$ami_id" \
61         --instance-type "$instance_type" \
62         --count $count \
63         --key-name "$key_name" \
64         --region "$region" \
65         --network-interfaces "DeviceIndex=0,SubnetId=$subnet_id,AssociatePublicIp=$associate_public_ip" \
66         --tag-specifications 'ResourceType=instance,Tags=[{Key=Name,Value=Custom-EC2}]'
67
68     if [ $? -eq 0 ]; then
69         echo "✓ EC2 instances created successfully in subnet $subnet_id."
70     else
71         echo "✗ Failed to create EC2 instances."
72     fi
73 }
74
75 check_num_of_args "$@"
76 activate_infra_environment
77 check_aws_cli || exit 1
78 check_aws_profile || exit 1
79 create_ec2_instances
80

```

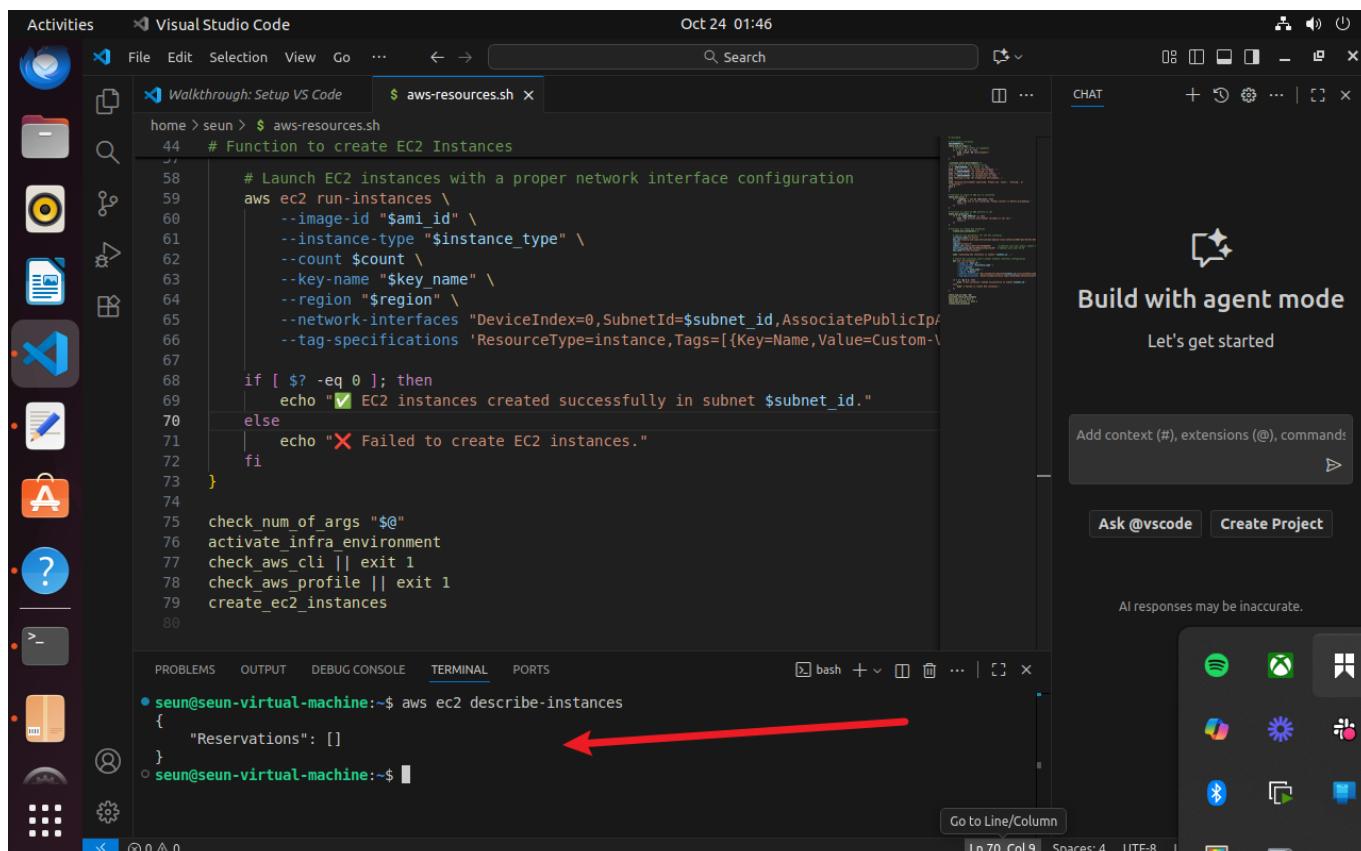
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS bash + ... | x

seun@seun-virtual-machine:~/aws-resources.sh\$./aws-resources.sh local ...
Running script for Local Environment...
seun@seun-virtual-machine:~\$

Ln 68, Col 23 Spaces: 4 UTF-8 LF { } Shell Script

Step 68: Verify No Existing EC2 Instances

- Before execution, note there are currently no EC2 instances (check via `aws ec2 describe-instances` or console)



```

Activities Visual Studio Code Oct 24 01:46
File Edit Selection View Go ... Search CHAT
Walkthrough: Setup VS Code $ aws-resources.sh
home > seu > $ aws-resources.sh
44 # Function to create EC2 Instances
58     # Launch EC2 instances with a proper network interface configuration
59     aws ec2 run-instances \
60         --image-id "$ami_id" \
61         --instance-type "$instance_type" \
62         --count $count \
63         --key-name "$key_name" \
64         --region "$region" \
65         --network-interfaces "DeviceIndex=0,SubnetId=$subnet_id,AssociatePublicIp=$associate_public_ip" \
66         --tag-specifications 'ResourceType=instance,Tags=[{Key=Name,Value=Custom-EC2}]'
67
68     if [ $? -eq 0 ]; then
69         echo "✓ EC2 instances created successfully in subnet $subnet_id."
70     else
71         echo "✗ Failed to create EC2 instances."
72     fi
73 }
74
75 check_num_of_args "$@"
76 activate_infra_environment
77 check_aws_cli || exit 1
78 check_aws_profile || exit 1
79 create_ec2_instances
80

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS bash + ... | x

● seun@seun-virtual-machine:~\$ aws ec2 describe-instances
{ "Reservations": [] }
seun@seun-virtual-machine:~\$

Go to Line/Column Ln 70, Col 9 Spaces: 4 UTF-8 LF { } Shell Script

Step 69: Add S3 Bucket Creation Function

Add the function and call it:

```
create_s3_buckets() {
    company="datawise364ng"
    departments=("marketing" "sales" "hr" "operations" "media")
    region="us-west-2"

    for department in "${departments[@]}"; do
        bucket_name="${company}-${department}-data-bucket"

        aws s3api create-bucket \
            --bucket "$bucket_name" \
            --region "$region" \
            --create-bucket-configuration LocationConstraint="$region"

        if [ $? -eq 0 ]; then
            echo "S3 bucket '$bucket_name' created successfully."
        else
            echo "Failed to create S3 bucket '$bucket_name'.""
        fi
    done
}

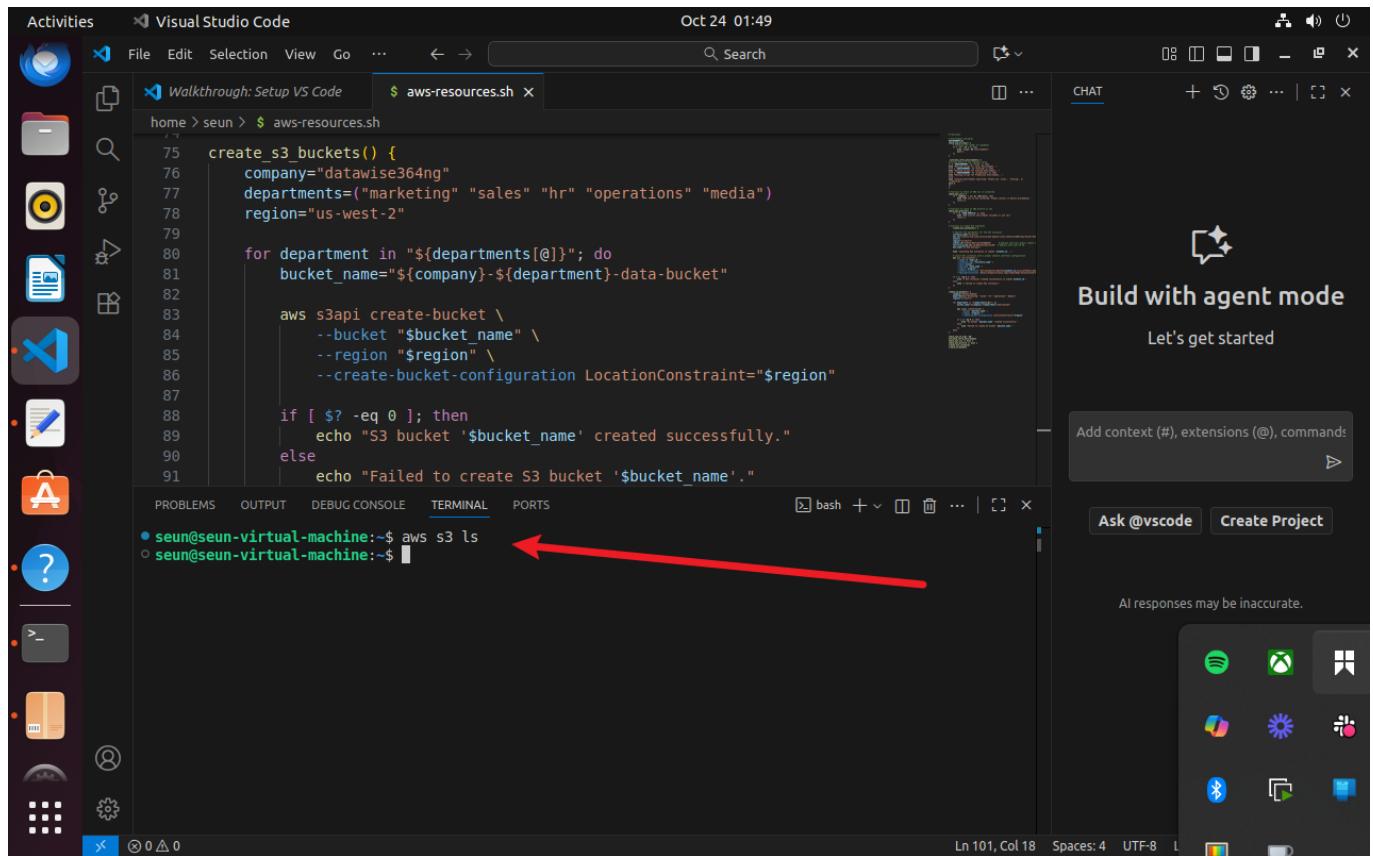
create_s3_buckets
```

Note: Creates multiple unique buckets to demonstrate automation.

The screenshot shows the Visual Studio Code interface. On the left is the file explorer with a single file named 'aws-resources.sh'. The main editor area contains the provided shell script. A red box highlights the 'create_s3_buckets()' function, and a red arrow points from the end of the function definition to the line 'create_s3_buckets()' at the bottom of the script. The terminal tab at the bottom shows the command 'aws ec2 describe-instances' being run, with the output indicating no reservations found. The status bar at the bottom right shows the line count as 'Ln 101, Col 18'.

Step 70: Verify No Existing S3 Buckets

- Before execution, note there are currently no S3 buckets (check via `aws s3 ls` or console)



A screenshot of the Visual Studio Code interface. On the left is the Activity Bar with various icons. The main area shows a terminal window titled 'aws-resources.sh' with the following content:

```

75  create_s3_buckets() {
76      company="datawise364ng"
77      departments=( "marketing" "sales" "hr" "operations" "media" )
78      region="us-west-2"
79
80      for department in "${departments[@]}"; do
81          bucket_name="${company}-${department}-data-bucket"
82
83          aws s3api create-bucket \
84              --bucket "$bucket_name" \
85              --region "$region" \
86              --create-bucket-configuration LocationConstraint="$region"
87
88          if [ $? -eq 0 ]; then
89              echo "S3 bucket '$bucket_name' created successfully."
90          else
91              echo "Failed to create S3 bucket '$bucket_name'." 

```

The terminal tab is selected at the bottom. A red arrow points from the text above to the terminal input field, which contains the command `aws s3 ls`. The status bar at the bottom right shows 'Ln 101, Col 18'.

Phase 10: Test Script and Troubleshoot Permissions

Step 71: Run Script to Create Resources

- Execute `./aws-resources.sh local`



A screenshot of a terminal window with the following text:

```

HP@DESKTOP-19M74R1 MINGW64 ~/Documents/Workspace/DevOps-Projects/Linux Shell Scripting - Advanced Application/Creating AWS Resources with Functions & Arrays using Shell Scripting (main)
$ ./aws-resources.sh local

```

Step 72: Initial Execution Results

- S3 buckets created successfully
- EC2 creation denied due to SSM permissions error (e.g., access denied for ssm:GetParameter)

Purpose: Identifies missing permissions in the policy.

The screenshot shows a Visual Studio Code interface with a terminal window open. The terminal output is as follows:

```
seun@seun-virtual-machine:~$ ./aws-resources.sh local
Running script for Local Environment...
Launching EC2 instances in subnet: subnet-06c7cfe7c9fb96a9b ...
An error occurred (SsmAccessDenied) when calling the RunInstances operation: User: arn:aws:iam::536697231935:user/automation_user is not authorized to perform: ssm:GetParameters on resource: arn:aws:ssm:us-east-2::parameter/aws/service/ami-amazon-linux-latest/al2023-ami-kernel-default-x86_64 because no identity-based policy allows the ssm:GetParameters action
✖ Failed to create EC2 instances.
{
    "Location": "http://datawise364ng-marketing-data-bucket.s3.amazonaws.com/"
}
S3 bucket 'datawise364ng-marketing-data-bucket' created successfully.
{
    "Location": "http://datawise364ng-sales-data-bucket.s3.amazonaws.com/"
}
S3 bucket 'datawise364ng-sales-data-bucket' created successfully.
{
    "Location": "http://datawise364ng-hr-data-bucket.s3.amazonaws.com/"
}
S3 bucket 'datawise364ng-hr-data-bucket' created successfully.
{
    "Location": "http://datawise364ng-operations-data-bucket.s3.amazonaws.com/"
}
S3 bucket 'datawise364ng-operations-data-bucket' created successfully.
{
    "Location": "http://datawise364ng-media-data-bucket.s3.amazonaws.com/"
}
S3 bucket 'datawise364ng-media-data-bucket' created successfully.
o seun@seun-virtual-machine:~$
```

The right side of the interface features a sidebar with various icons and sections. A large red arrow points from the word "Failed" in the sidebar to the error message in the terminal. Another red arrow points from the word "Successful" in the sidebar to the "S3 bucket ... created successfully." entries in the terminal output.

Step 73: Verify Created S3 Buckets

- Check the created buckets on the S3 console dashboard

General purpose buckets (5) [Info](#)

Buckets are containers for data stored in S3.

Find buckets by name

Name	AWS Region	Creation date
datawise364ng-hr-data-bucket	US West (Oregon) us-west-2	October 24, 2025, 02:51:26 (UTC+01:00)
datawise364ng-marketing-data-bucket	US West (Oregon) us-west-2	October 24, 2025, 02:51:16 (UTC+01:00)
datawise364ng-media-data-bucket	US West (Oregon) us-west-2	October 24, 2025, 02:51:38 (UTC+01:00)
datawise364ng-operations-data-bucket	US West (Oregon) us-west-2	October 24, 2025, 02:51:33 (UTC+01:00)
datawise364ng-sales-data-bucket	US West (Oregon) us-west-2	October 24, 2025, 02:51:20 (UTC+01:00)

[Create bucket](#)

Account snapshot [Info](#)
Updated daily

Storage Lens provides visibility into storage usage and activity trends.

External access summary - new [Info](#)
Updated daily

External access findings help you identify bucket permissions that allow public access or access from other AWS accounts.

Step 74: Navigate Back to IAM Policies

- Back to IAM dashboard, click on Policies

The screenshot shows the AWS IAM Dashboard. On the left sidebar, under 'Access management', the 'Policies' option is selected and highlighted with a red arrow. The main content area displays 'Security recommendations' with two items: 'Root user has MFA' and 'Root user has no active access keys'. Below this is a summary of 'IAM resources' with counts: User groups (0), Users (2), Roles (7), Policies (13), and Identity providers (0). A 'What's new' section lists recent updates. On the right side, there are boxes for 'AWS Account' (Account ID: 536697231935, Account Alias: Create), 'Quick Links' (My security credentials), and 'Tools' (Policy simulator). At the bottom, there are links for CloudShell, Feedback, and various AWS services.

Step 75: Open Existing Policy

- Search for EC2S3FullAccessPolicy created initially and click to open it

The screenshot shows the 'Policies' list page. The left sidebar shows 'Policies' selected. The main table lists one policy: 'EC2S3FullAccessPolicy'. A red arrow points to the policy name 'EC2S3FullAccessPolicy' in the table. The table includes columns for 'Policy name' (EC2S3FullAccessPolicy), 'Type' (Customer managed), 'Used as' (Permissions policy (2)), and 'Description' (Policy granting full access to EC2 and S3). The top right of the table has buttons for 'Actions' and 'Delete', and a 'Create policy' button. The bottom right corner shows a small preview of the Windows taskbar with several icons.

Step 76: Edit Policy

- On EC2S3FullAccessPolicy dashboard, in permissions section, click Edit

EC2S3FullAccessPolicy Info

Policy granting full access to EC2 and S3

Policy details

Type Customer managed	Creation time October 23, 2025, 06:27 (UTC+01:00)	Edited time October 23, 2025, 06:27 (UTC+01:00)	ARN arn:aws:iam::536697231935:policy/EC2S3FullAccessPolicy
--------------------------	--	--	---

Permissions Entities attached Tags Policy versions (1) Last Accessed

Permissions defined in this policy Info

Permissions defined in this policy document specify which actions are allowed or denied. To define permissions for an IAM identity (user, user group, or role), attach a policy to it.

Allow (of 450 services)

Service	Access level	Resource	Request condition
EC2	Full access	All resources	None
S3	Full access	All resources	None

Show remaining 448 services

Edit **Summary** **JSON**

Step 77: Access JSON Editor

- Ensure JSON is selected and policy editor JSON shows

Modify permissions in EC2S3FullAccessPolicy Info

Add permissions by selecting services, actions, resources, and conditions. Build permission statements using the JSON editor.

Policy editor

```

1 Version: "2012-10-17",
2 Statement: [
3   {
4     Effect: "Allow",
5     Action: [
6       "ec2:*",
7       "s3:*"
8     ],
9     Resource: "*"
10    }
11  ]
12 ]
13

```

Actions

Edit statement

Select a statement

Select an existing statement in the policy or add a new statement.

+ Add new statement

Show hidden icons

Step 78: Update JSON for SSM Permissions

Update the JSON by adding a new statement for SSM:

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

```

    "s3:*",
    "ssm:GetParameter",
    "ssm:GetParameters",
    "ssm:DescribeParameters"
],
"Resource": "*"
}
]
}

```

Purpose: Add permissions for SSM Parameter Store access used in the script.

Step 1
Modify permissions in EC2S3FullAccessPolicy

Step 2
Review and save

Modify permissions in EC2S3FullAccessPolicy Info

Add permissions by selecting services, actions, resources, and conditions. Build permission statements using the JSON editor.

Policy editor

```

1 ▼ {
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Effect": "Allow",
6       "Action": [
7         "ec2:*",
8         "s3:*",
9         "ssm:GetParameter",
10        "ssm:GetParameters",
11        "ssm:DescribeParameters"
12      ],
13      "Resource": "*"
14    }
15  ]
16 }
17

```

Edit statement

Select a statement

Select an existing statement in the policy or add a new statement.

+ Add new statement

Show hidden icons

© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Step 79: Proceed with Edit

- Scroll down and click Next

Modify permissions in EC2S3FullAccessPolicy Info

Add permissions by selecting services, actions, resources, and conditions. Build permission statements using the JSON editor.

Policy editor

```

10    "ssm:GetParameters",
11    "ssm:DescribeParameters"
12  ],
13  "Resource": "*"
14 }
15 ]
16 }
17

```

+ Add new statement

JSON Ln 17, Col 0 5982 of 6144 characters remaining

Security: 0 Errors: 0 Warnings: 0 Suggestions: 0

Check for new access

Cancel **Next**

© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Step 80: Save Policy Changes

- Click to set new version as default and click Save changes

Step 1
Modify permissions in EC2S3FullAccessPolicy
Step 2
Review and save

Review and save Info

Review the permissions, specify details, and tags.

Permissions defined in this policy Info

Permissions defined in this policy document specify which actions are allowed or denied. To define permissions for an IAM identity (user, user group, or role), attach a policy to it.

Service	Access level	Resource	Request condition
EC2	Full access	All resources	None
S3	Full access	All resources	None
Systems Manager	Limited: List, Read	All resources	None

Set this new version as the default.
Permissions defined in this version will be applied to all the entities this policy is attached to.

Save changes

Step 81: Verify Policy Edit

- Policy edited successfully

Identity and Access Management (IAM)

EC2S3FullAccessPolicy Info

Policy granting full access to EC2 and S3

Policy details

Type Customer managed	Creation time October 23, 2025, 06:27 (UTC+01:00)	Edited time October 24, 2025, 03:50 (UTC+01:00)	ARN arn:aws:iam::536697231935:policy/EC2S3FullAccessPolicy
--------------------------	--	--	---

Permissions **Entities attached** **Tags** **Policy versions (2)** **Last Accessed**

Permissions defined in this policy Info

Permissions defined in this policy document specify which actions are allowed or denied. To define permissions for an IAM identity (user, user group, or role), attach a policy to it.

Service	Access level	Resource	Request condition
EC2	Full access	All resources	None
S3	Full access	All resources	None

Save changes

Step 82: Delete Existing S3 Buckets

- Run commands to delete all buckets starting with "datawise" created in first execution (e.g., `aws s3 rb s3://datawise-bucket --force` in a loop)

```
for bucket in $(aws s3api list-buckets --query "Buckets[].Name" --output text | tr '\t' '\n' | grep '^datawise'); do
```

```
echo "⚡ Deleting $bucket ..."  
aws s3 rb "s3://$bucket" --force  
done
```

Purpose: Clean up before re-testing to avoid duplicates.



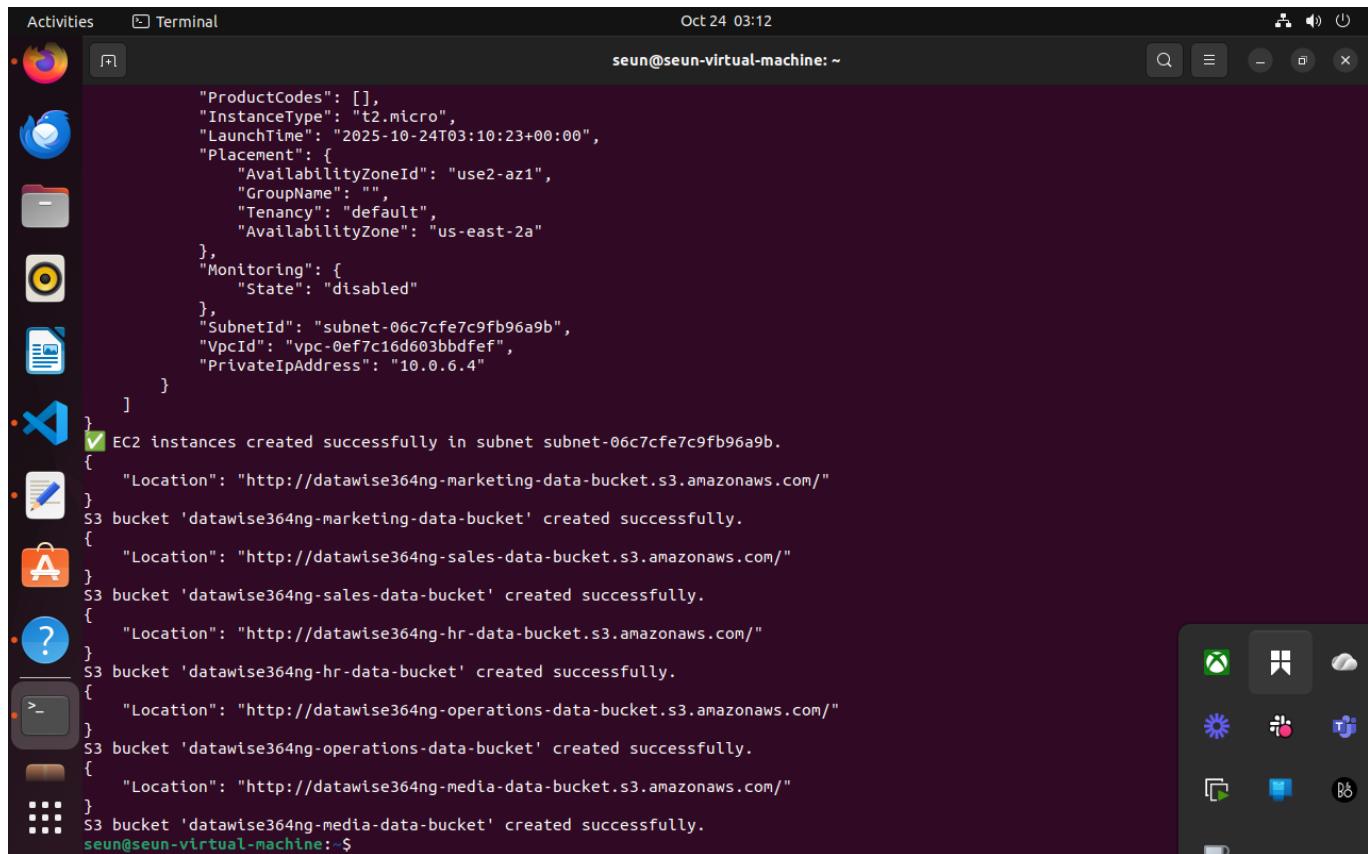
Step 83: Re-run Script for EC2

- Execute the script again; EC2 block now launches successfully with no error, EC2 creation initiated

```
Activities Terminal Oct 24 03:10  
seun@seun-virtual-machine: ~  
seun@seun-virtual-machine:~$ ./aws-resources.sh local  
Running script for Local Environment...  
Launching EC2 instances in subnet: subnet-06c7cfec7c9fb96a9b ...  
{  
    "ReservationId": "r-0315e38956a766203",  
    "OwnerId": "536697231935",  
    "Groups": [],  
    "Instances": [  
        {  
            "Architecture": "x86_64",  
            "BlockDeviceMappings": [],  
            "ClientToken": "7bbab70d-0c6c-449b-85a9-5eab640d5441",  
            "EbsOptimized": false,  
            "EnaSupport": true,  
            "Hypervisor": "xen",  
            "NetworkInterfaces": [  
                {  
                    "Attachment": {  
                        "AttachTime": "2025-10-24T03:10:23+00:00",  
                        "AttachmentId": "eni-attach-081fbf46a0f7852cd",  
                        "DeleteOnTermination": true,  
                        "DeviceIndex": 0,  
                        "Status": "attaching",  
                        "NetworkCardIndex": 0  
                    },  
                    "Description": "",  
                    "Groups": [  
                        {  
                            "GroupId": "sg-0012c41f2dcc7ac95",  
                            "GroupName": "my-first-security-group"  
                        }  
                    ],  
                    "Ipv6Addresses": [],  
                    "MacAddress": "02:1a:cc:29:b5:83",  
                    "NetworkInterfaceId": "eni-01059b68eae46262e",  
                    "OwnerId": "536697231935",  
                    "PrivateIpAddress": "10.0.6.178",  
                }  
            ]  
        }  
    ]  
}
```

Step 84: Successful Resource Creation

- EC2 and S3 now created successfully with no error



```

{
    "ProductCodes": [],
    "InstanceType": "t2.micro",
    "LaunchTime": "2025-10-24T03:10:23+00:00",
    "Placement": {
        "AvailabilityZoneId": "use2-az1",
        "GroupName": "",
        "Tenancy": "default",
        "AvailabilityZone": "us-east-2a"
    },
    "Monitoring": {
        "State": "disabled"
    },
    "SubnetId": "subnet-06c7cf7c9fb96a9b",
    "VpcId": "vpc-0ef7c16d603bbdfef",
    "PrivateIpAddress": "10.0.6.4"
}
]

}

EC2 instances created successfully in subnet subnet-06c7cf7c9fb96a9b.

{
    "Location": "http://datawise364ng-marketing-data-bucket.s3.amazonaws.com/"
}
S3 bucket 'datawise364ng-marketing-data-bucket' created successfully.

{
    "Location": "http://datawise364ng-sales-data-bucket.s3.amazonaws.com/"
}
S3 bucket 'datawise364ng-sales-data-bucket' created successfully.

{
    "Location": "http://datawise364ng-hr-data-bucket.s3.amazonaws.com/"
}
S3 bucket 'datawise364ng-hr-data-bucket' created successfully.

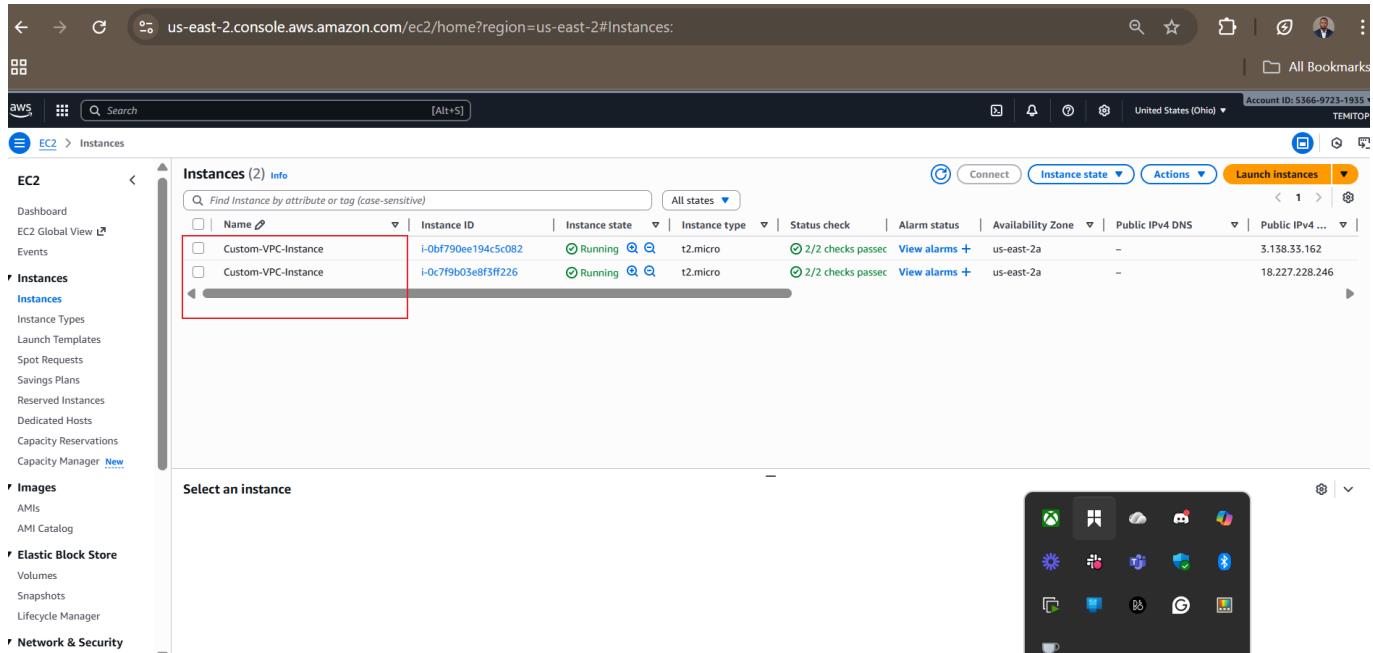
{
    "Location": "http://datawise364ng-operations-data-bucket.s3.amazonaws.com/"
}
S3 bucket 'datawise364ng-operations-data-bucket' created successfully.

{
    "Location": "http://datawise364ng-media-data-bucket.s3.amazonaws.com/"
}
S3 bucket 'datawise364ng-media-data-bucket' created successfully.

seun@seun-virtual-machine:~$
```

Step 85: Verify Created EC2 Instances

- Check created instances on EC2 instance dashboard



Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...
Custom-VPC-Instance	i-0bf790ee194c5c082	Running	t2.micro	2/2 checks passed	View alarms	us-east-2a	-	3.138.33.162
Custom-VPC-Instance	i-0c79b03e8f3ff226	Running	t2.micro	2/2 checks passed	View alarms	us-east-2a	-	18.227.228.246

Phase 11: Clean Up Resources

Step 86: Terminate EC2 Instances

- Select all the instances, click Instance state, and select Terminate instances

The screenshot shows the AWS EC2 Instances page. Two instances, both named "Custom-VPC-Instance", are selected. In the top right corner of the instance list, there is a context menu with several options: "Stop instance", "Start instance", "Reboot instance", "Hibernate instance", and "Terminate (delete) instance". A red arrow points to the "Terminate (delete) instance" button.

Step 87: Confirm Termination

- Click button to terminate and delete

The screenshot shows the AWS EC2 Instances page with the same two instances selected. A modal dialog box titled "Terminate (delete) instances" is open. It contains a warning message about EBS-backed instances and a confirmation question: "Are you sure you want to terminate these instances?". Below this, there are two sections: "Instance ID" and "Termination protection", each listing the two selected instances with "Disabled" checked. At the bottom of the dialog, there is a note about skipping OS shutdown and a checkbox for "Skip OS shutdown". A large red arrow points to the "Terminate (delete)" button at the bottom right of the dialog.

Step 88: Verify Termination

- Instances now terminated

The screenshot shows the AWS EC2 Instances page. On the left, a sidebar lists various EC2-related services like Dashboard, Global View, Instances, Images, Elastic Block Store, Network & Security, and CloudShell. The main area displays a table of instances. Two instances are selected, both named 'Custom-VPC-Instance'. Both instances are in the 'Terminated' state, with instance IDs i-0bf790ee194c5c082 and i-0c7f9b03e8f3ff226 respectively. The status check is '2/2 checks passed' for both. The availability zone is us-east-2a and the public IPv4 DNS and public IPv4 are listed as '-'.

Step 89: Delete S3 Buckets

- Run command to delete all buckets starting with "datawise" to avoid costs as well (e.g., `aws s3 rb s3://datawise-bucket --force` in a loop)

```
Activities Terminal Oct 24 03:06
seun@seun-virtual-machine: ~/aws
seun@seun-virtual-machine:~/.aws$ for bucket in $(aws s3api list-buckets --query "Buckets[].Name" --output text | tr '\t' '\n' | grep '^datawise'); do
    echo "Deleting $bucket ..."
    aws s3 rb "s3://$bucket" --force
done
Deleting datawise364ng-hr-data-bucket ...
remove_bucket: datawise364ng-hr-data-bucket
Deleting datawise364ng-marketing-data-bucket ...
remove_bucket: datawise364ng-marketing-data-bucket
Deleting datawise364ng-media-data-bucket ...
remove_bucket: datawise364ng-media-data-bucket
Deleting datawise364ng-operations-data-bucket ...
remove_bucket: datawise364ng-operations-data-bucket
Deleting datawise364ng-sales-data-bucket ...
remove_bucket: datawise364ng-sales-data-bucket
seun@seun-virtual-machine:~/.aws$
```

Conclusion

This guide has provided a thorough, step-by-step process for setting up secure AWS authentication and automation, from IAM configuration to scripting resource management. By following these instructions, you've built a foundation that not only automates EC2 and S3 tasks but also prioritizes security and efficiency. Remember, maintaining secure practices like regular key rotation and monitoring is key to long-term success in cloud environments.

Learning Outcomes

By completing this tutorial, you will:

- Understand IAM components and how to apply the least privilege principle.
- Be able to create custom policies, roles, and users for specific AWS services.
- Know how to install and configure the AWS CLI on Ubuntu.
- Gain skills in writing and debugging shell scripts for AWS automation.
- Learn to troubleshoot permissions and clean up resources to manage costs effectively.

Recommendations

- Regularly review and audit IAM policies to ensure they remain aligned with your needs.
- Explore advanced tools like AWS SDKs or Terraform for more complex infrastructure as code.
- Enable AWS CloudTrail for logging API activities and enhance monitoring.
- Consider using temporary credentials via STS for even greater security in production.
- For further learning, refer to the official AWS documentation on IAM best practices and CLI commands.

Next Steps

1. Enhance the script with additional features like error handling or multi-region support.
2. Integrate this setup into a CI/CD pipeline using tools like Jenkins or GitHub Actions.
3. Experiment with other AWS services, such as Lambda or RDS, using similar authentication methods.
4. Join AWS communities or forums for troubleshooting and advanced tips.

Total Documentation

89 Screenshots - Complete visual step-by-step guide

Author

Oluwaseun Osunsola

Date: October 24, 2025

License

This project is licensed under the MIT License - see the LICENSE file for details.