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.

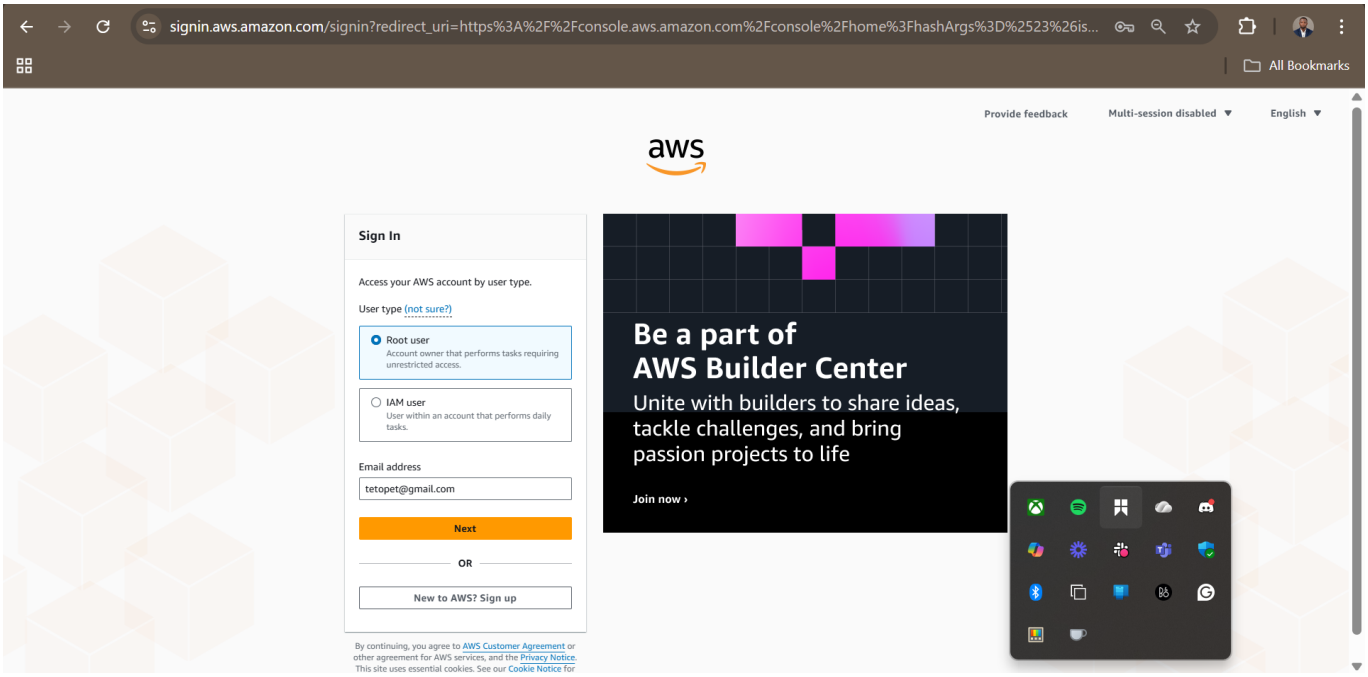
🚀 Step-by-Step Implementation

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

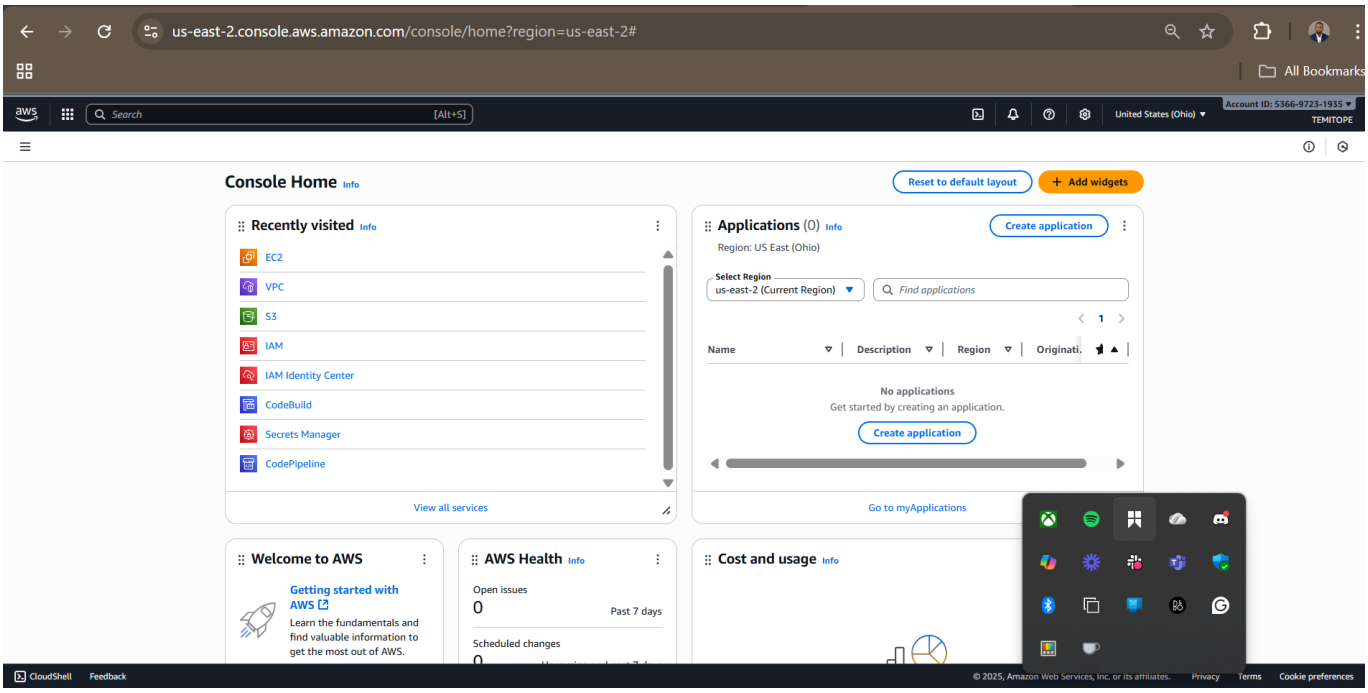
Purpose: Secure console access with 2FA



Step 2: Successful Console Login

- AWS Management Console dashboard loaded

Status: Ready for IAM configuration

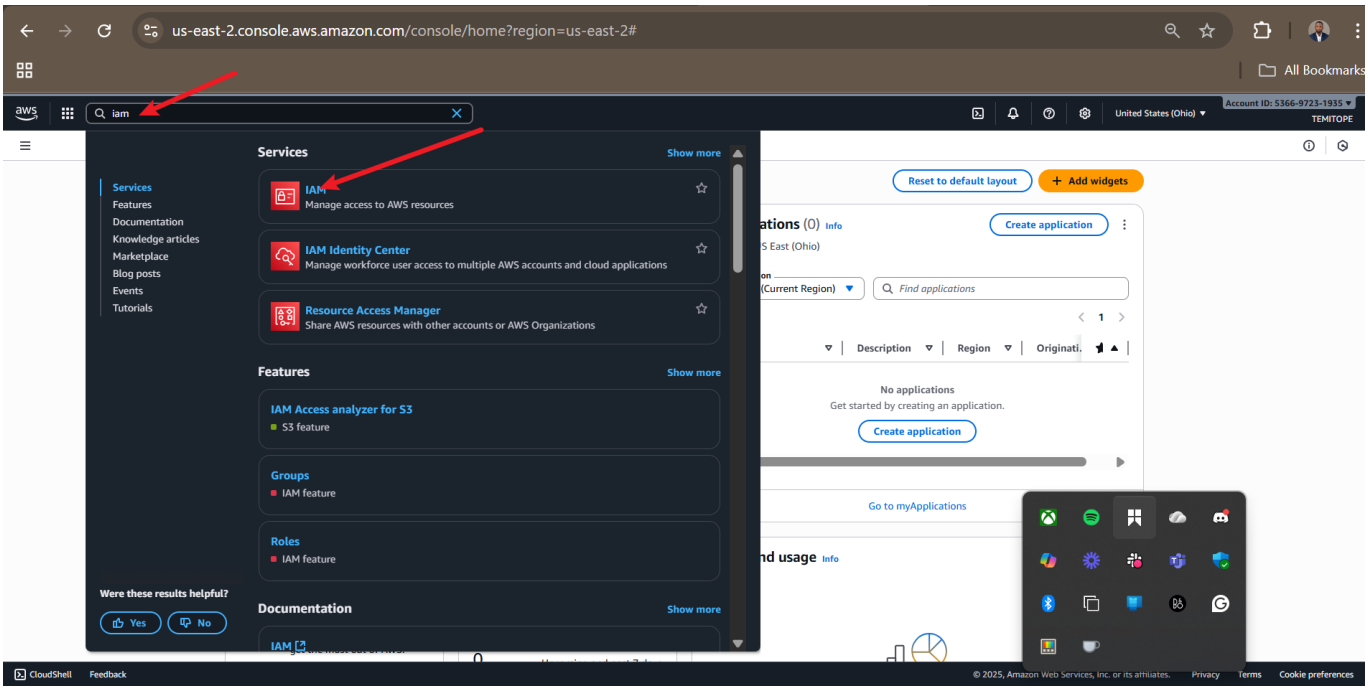


Phase 2: IAM Policy Creation

Step 3: Navigate to IAM Service

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

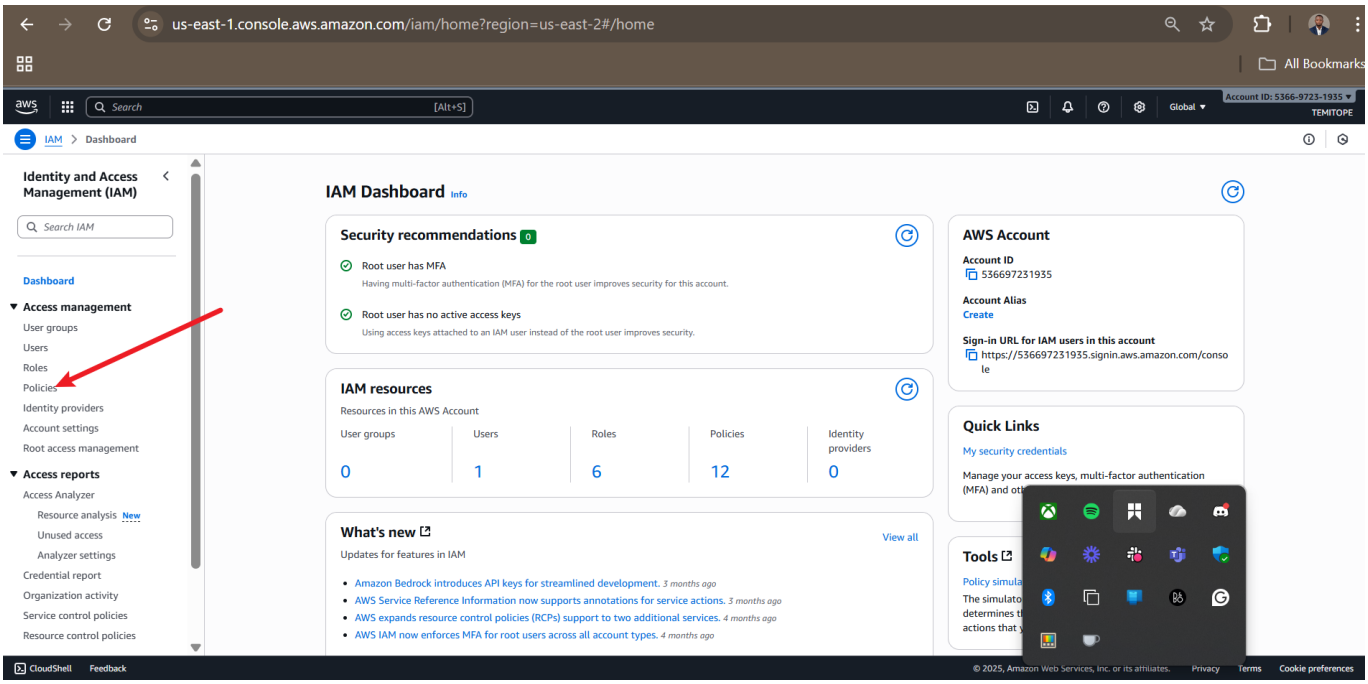
Purpose: Access Identity and Access Management console



Step 4: Access Policies Section

- Click Policies in left sidebar

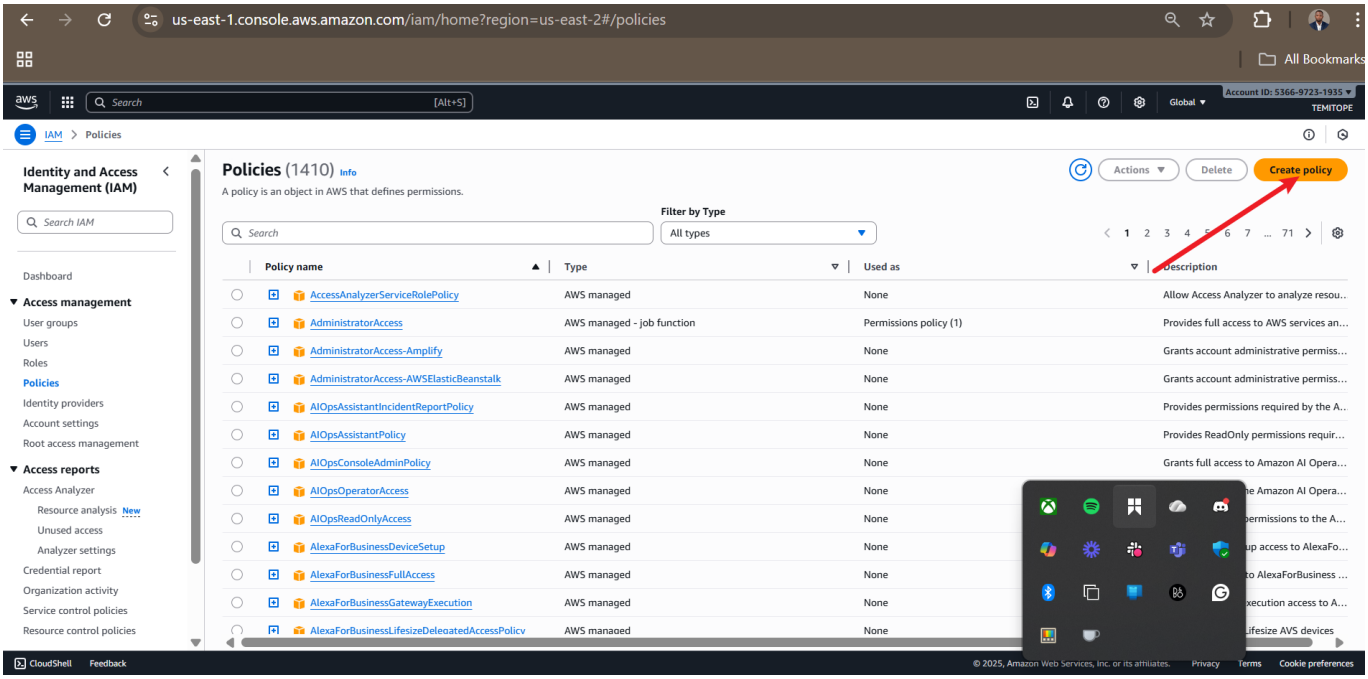
Purpose: Manage custom permission policies



Step 5: Initiate Policy Creation

- Click Create policy button

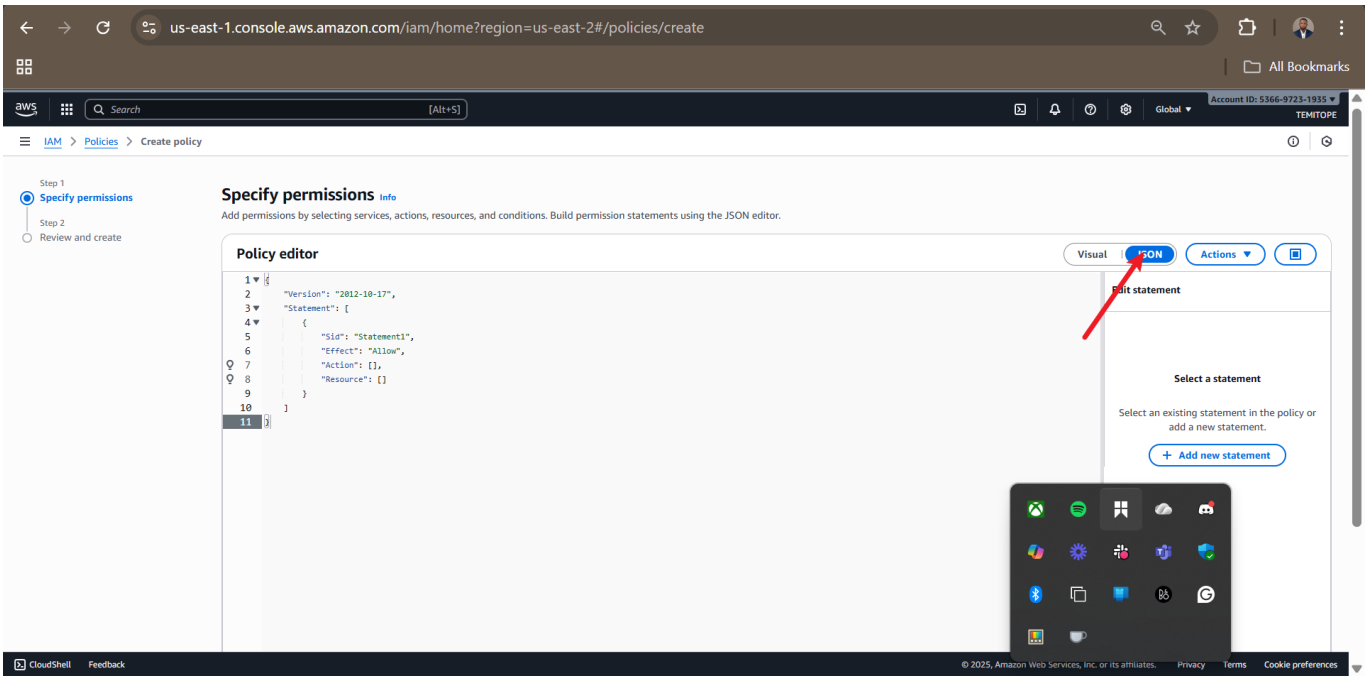
Purpose: Define custom EC2+S3 permissions



Step 6: JSON Policy Editor

- Select JSON tab

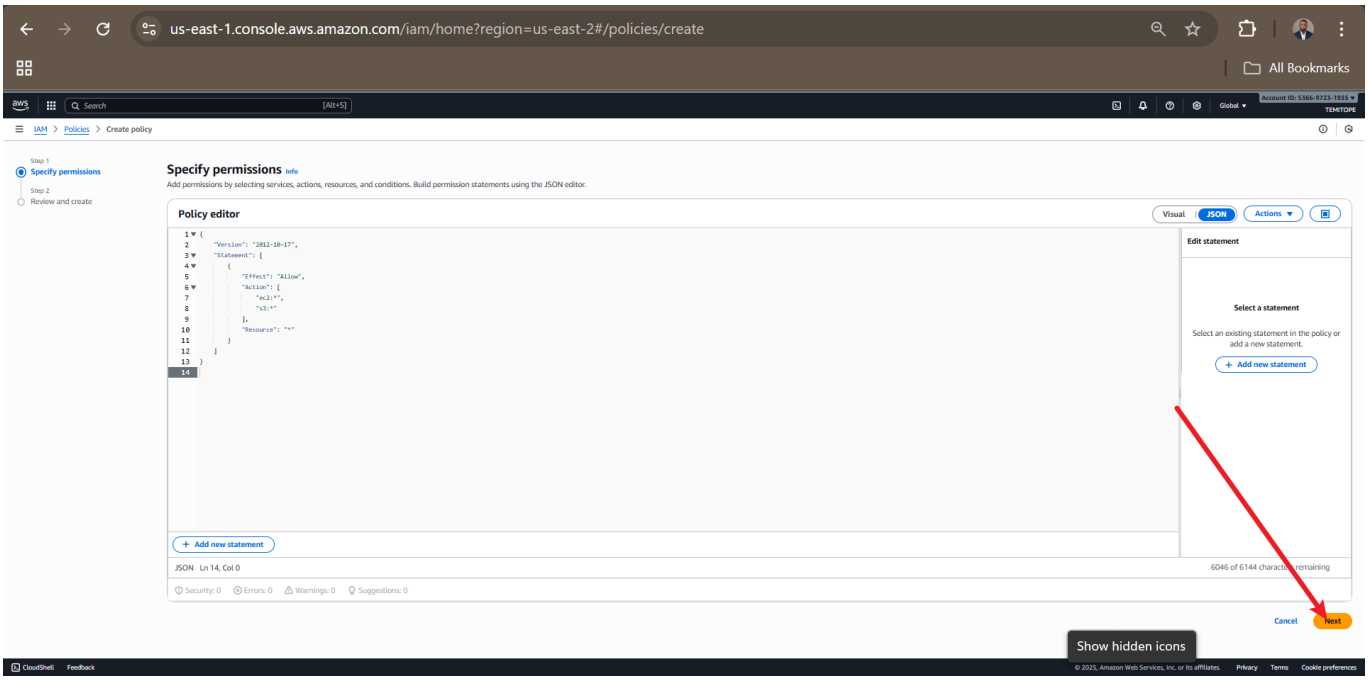
Purpose: Manual policy definition for precision



Step 7: Define EC2+S3 Full Access Policy

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

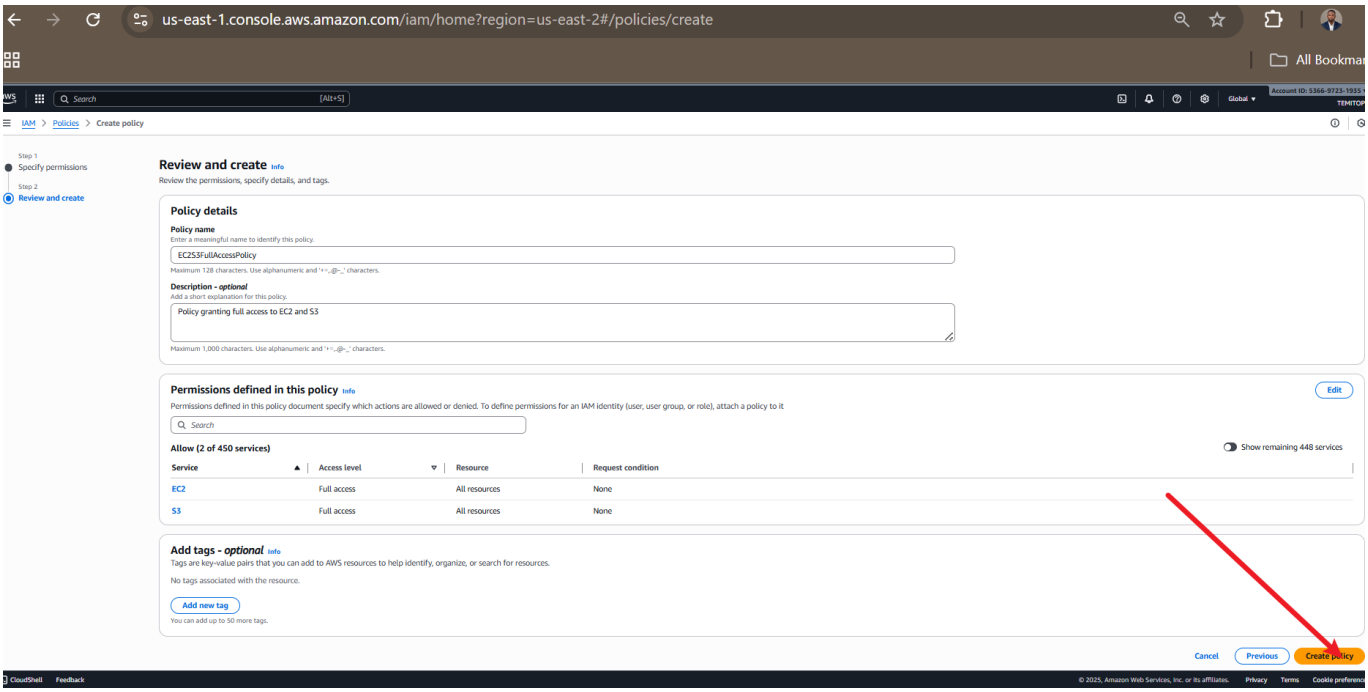
- Replace default JSON with EC2+S3 permissions
- Click Next



Step 8: Name and Create Policy

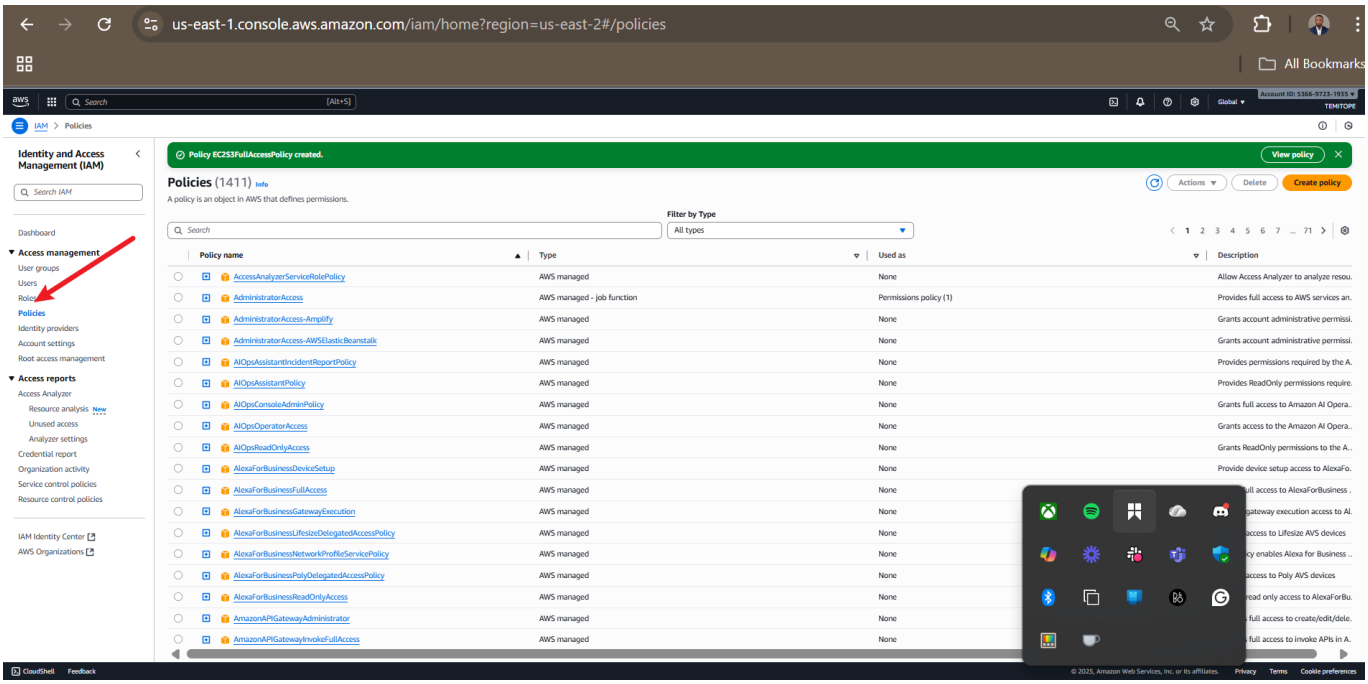
- Name: EC2S3FullAccessPolicy
- Click Create policy

Result: Custom policy available for attachment



Step 9: Verify Policy Creation

- Policy appears in Policies list
- Navigate to Roles section

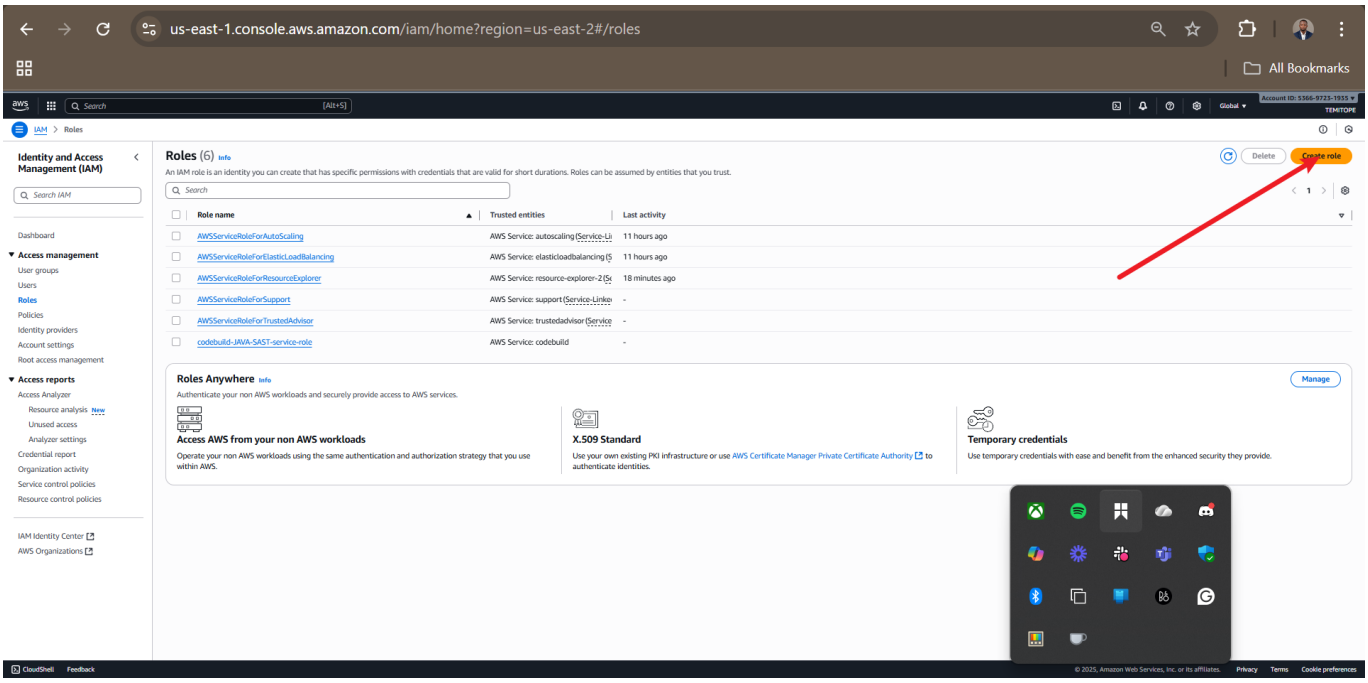


Phase 3: IAM Role Creation

Step 10: Start Role Creation

- Click Create role

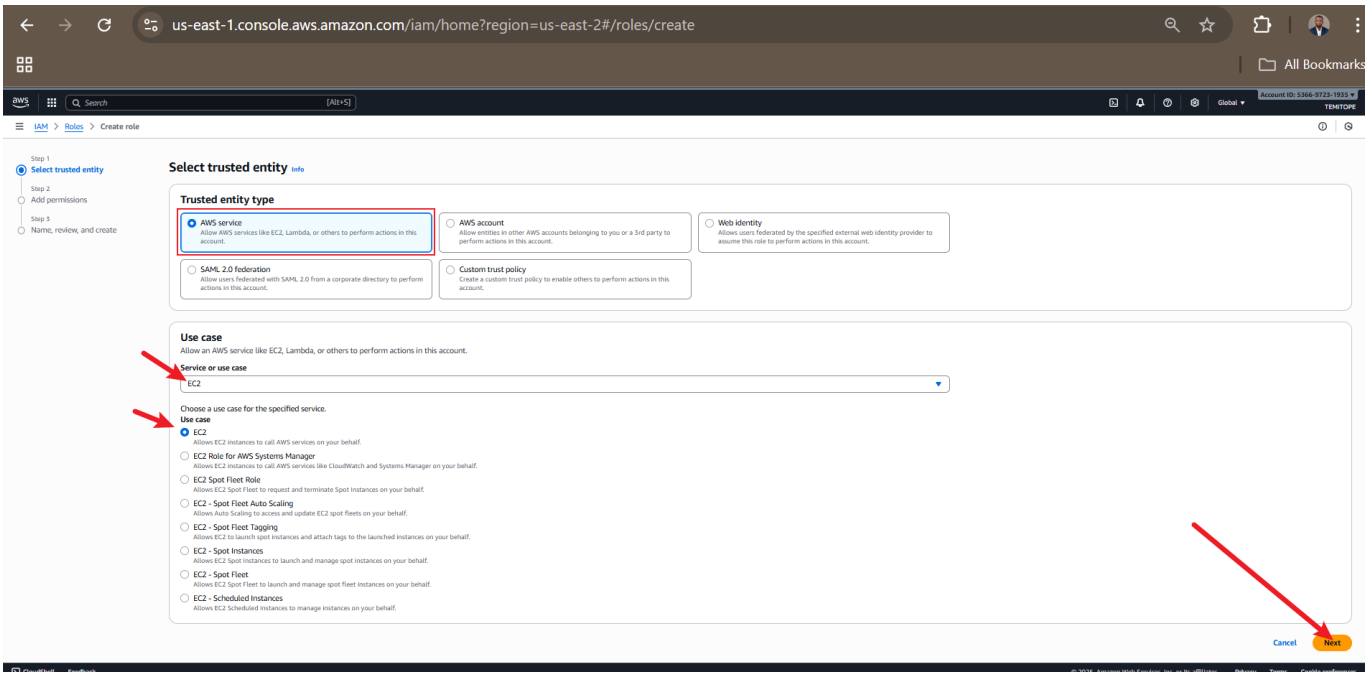
Purpose: Create service-trusted role for EC2



Step 11: Configure Trusted Entity

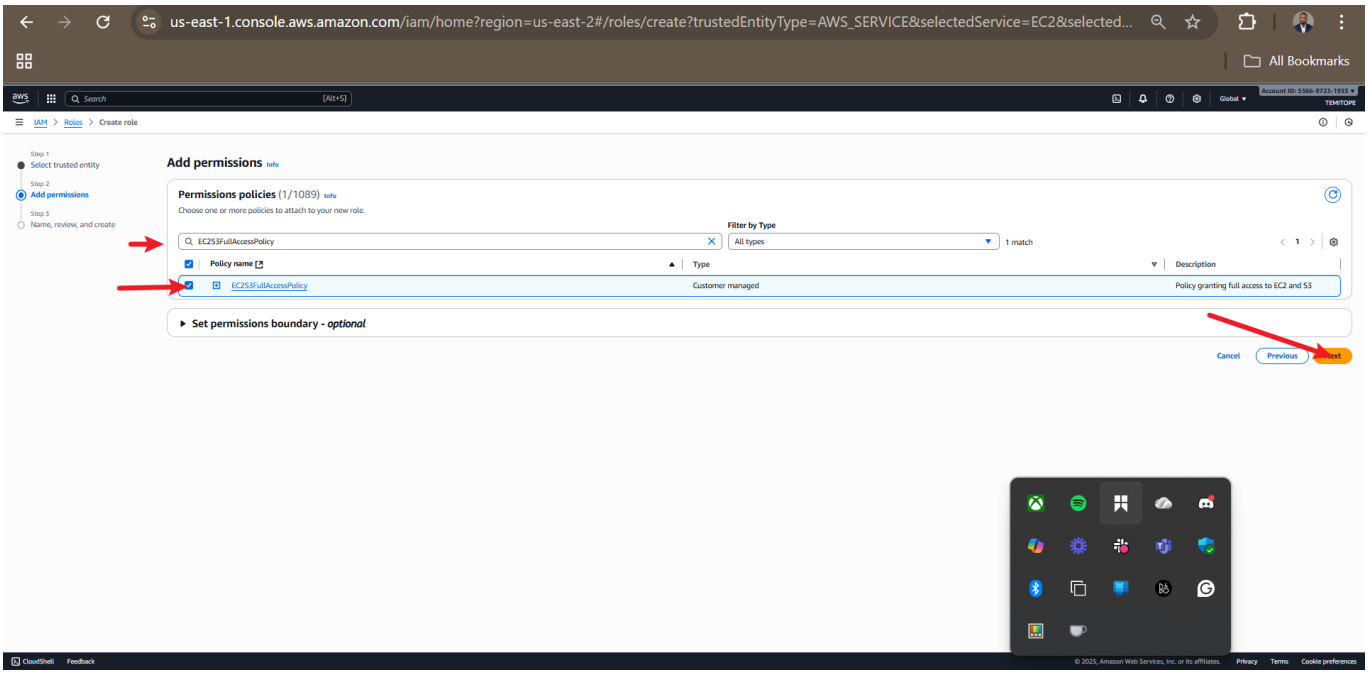
- Trusted entity: AWS service
- Use case: Search and select EC2
- Click Next

Purpose: Allow EC2 instances to assume this role



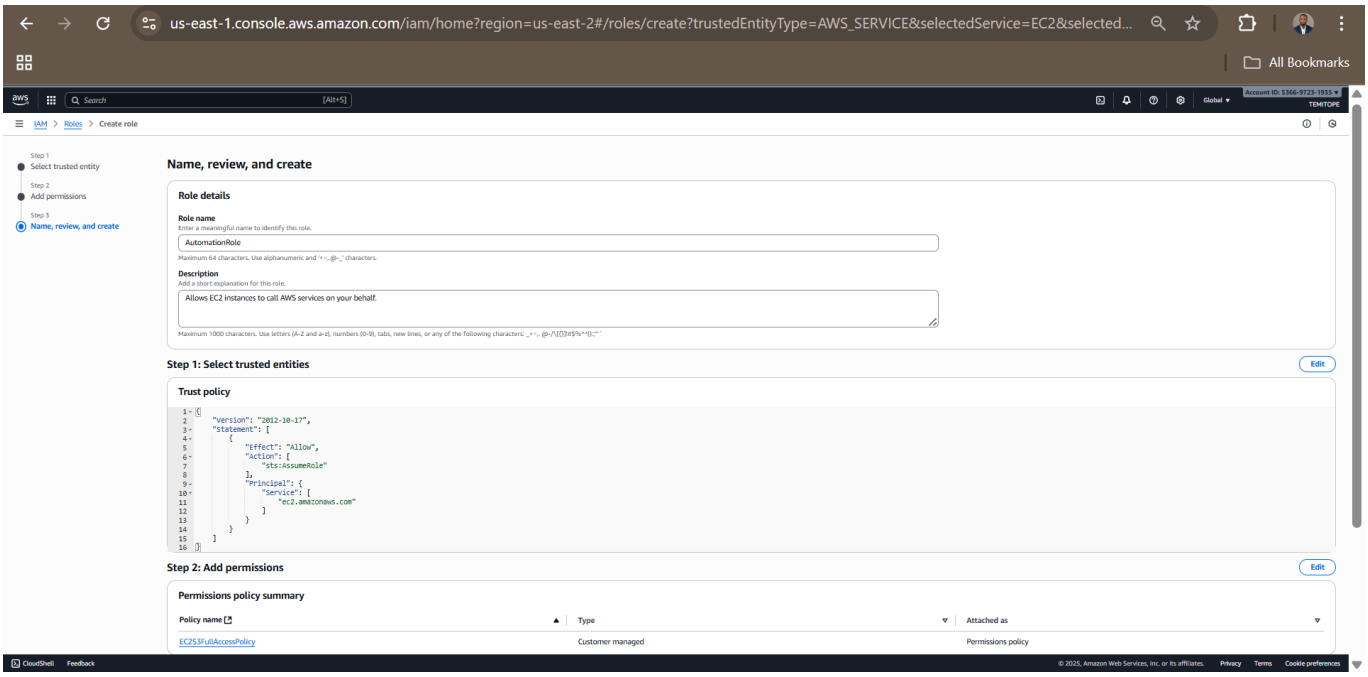
Step 12: Attach Policy to Role

- Search: EC2S3FullAccessPolicy
- Select policy checkbox
- Click Next



Step 13: Name the Role

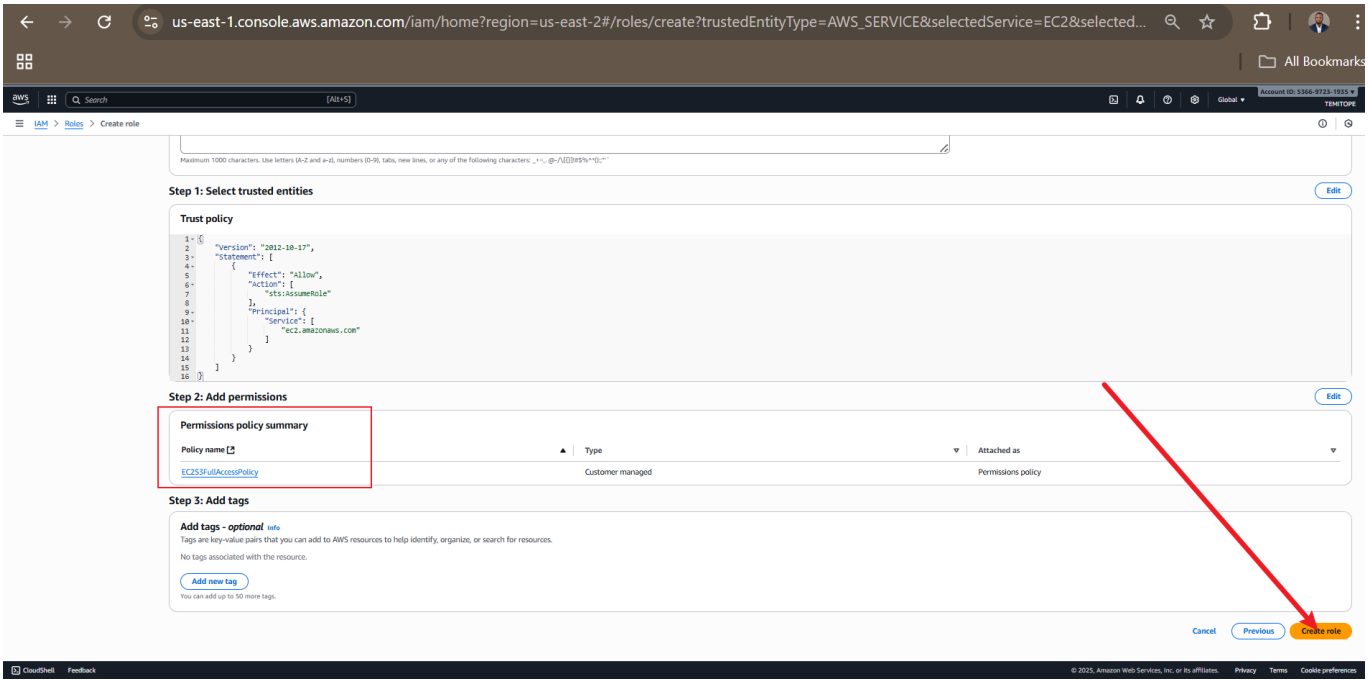
- Role name: AutomationRole
- Scroll to review settings



Step 14: Create Role

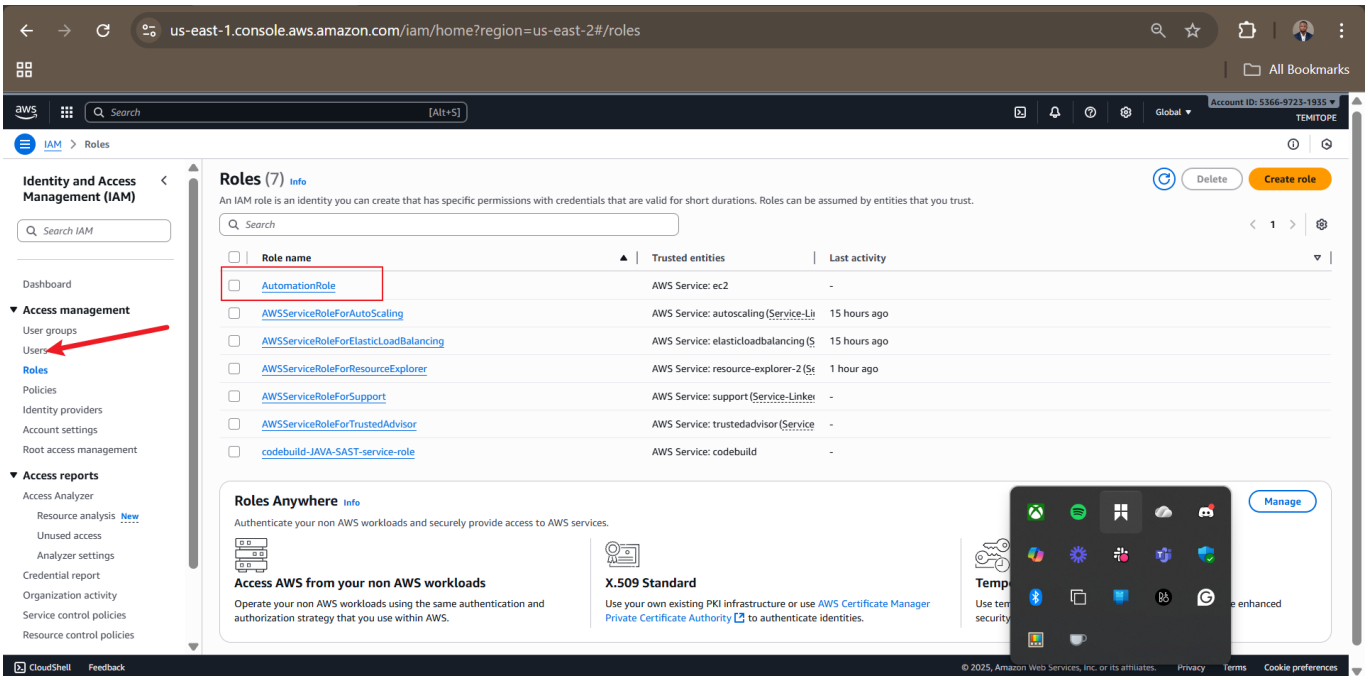
- Review summary
- Click Create role

Result: AutomationRole ready



Step 15: Role Dashboard Verification

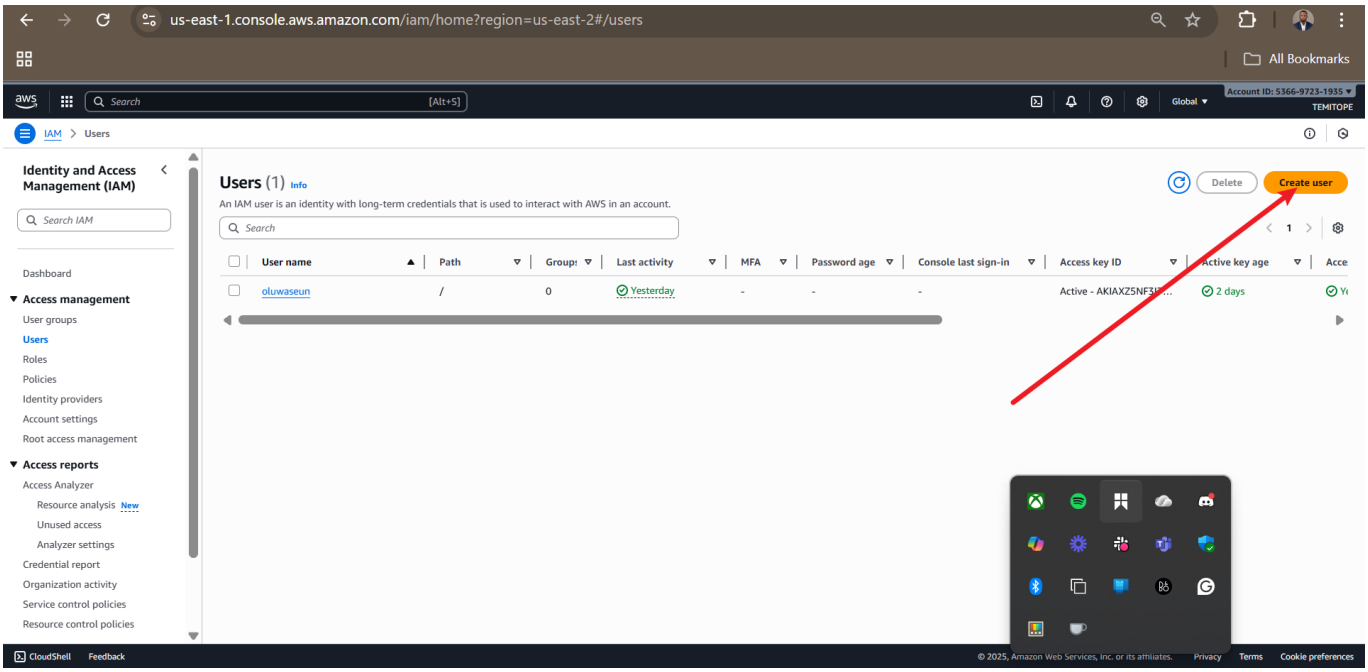
- AutomationRole visible in roles list
- Navigate to Users



Phase 4: IAM User Creation

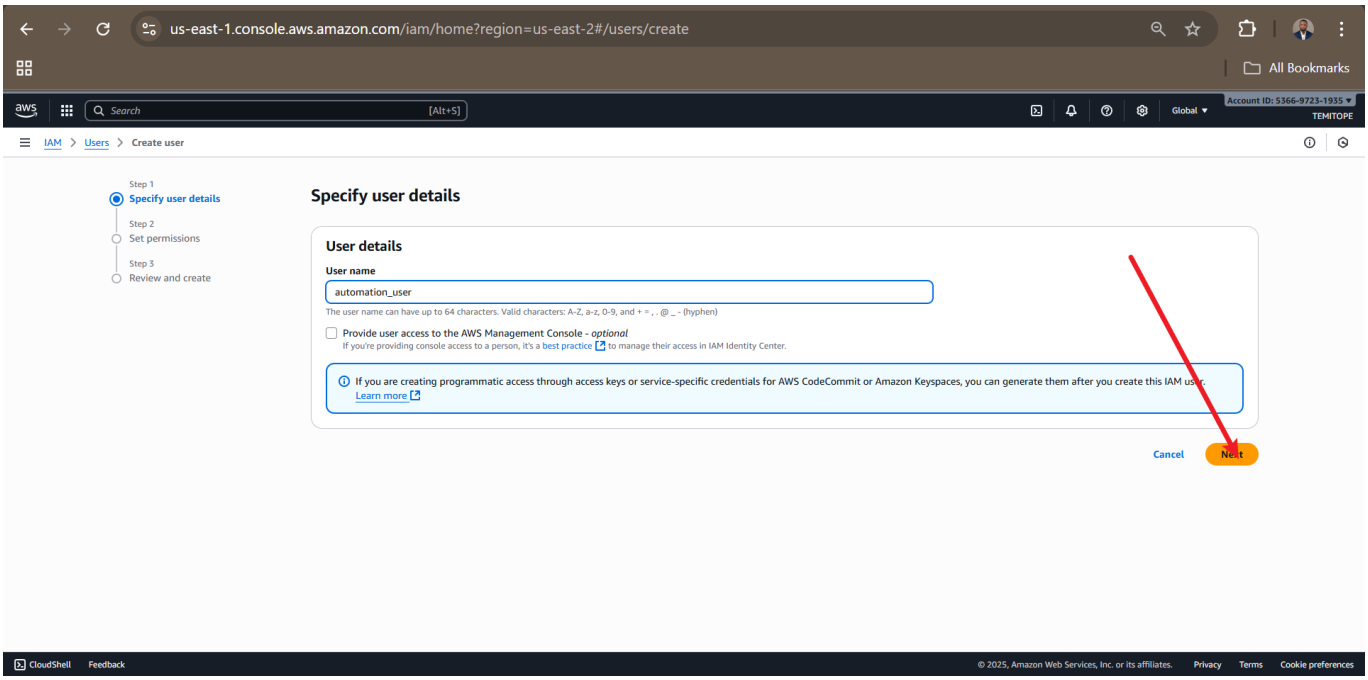
Step 16: Create Automation User

- Click Create user



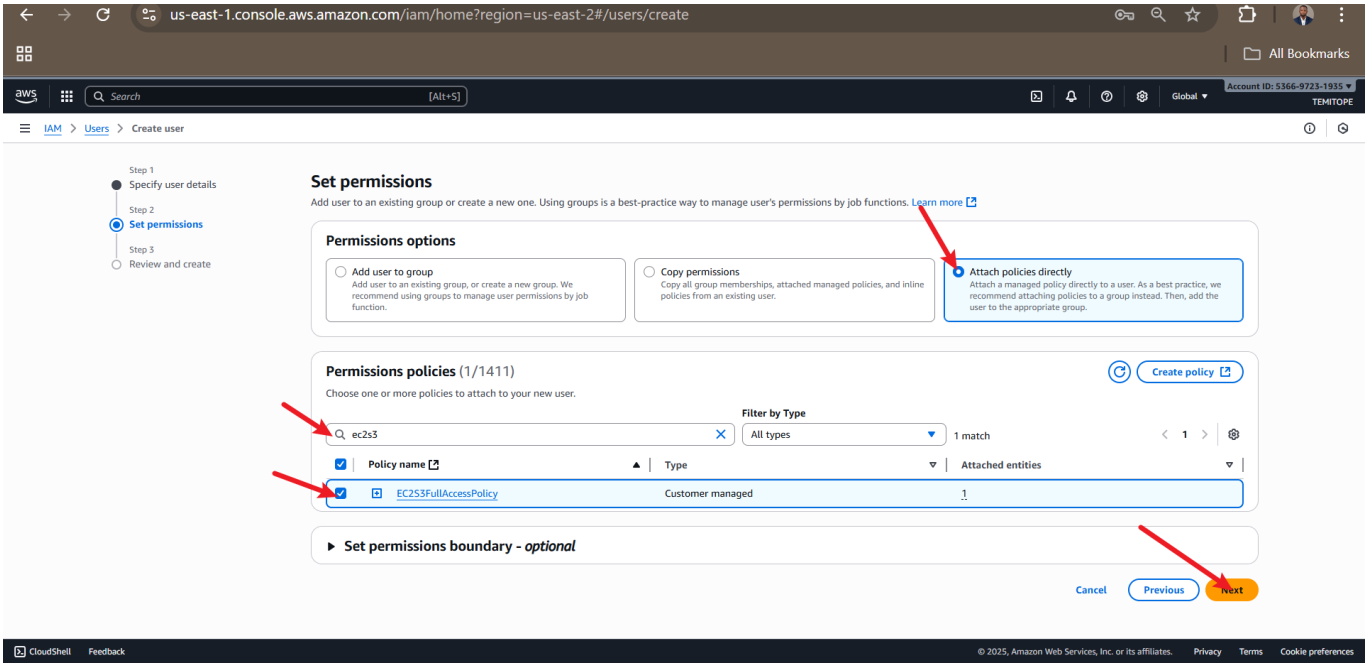
Step 17: Name the User

- User name: automation_user
- Click Next



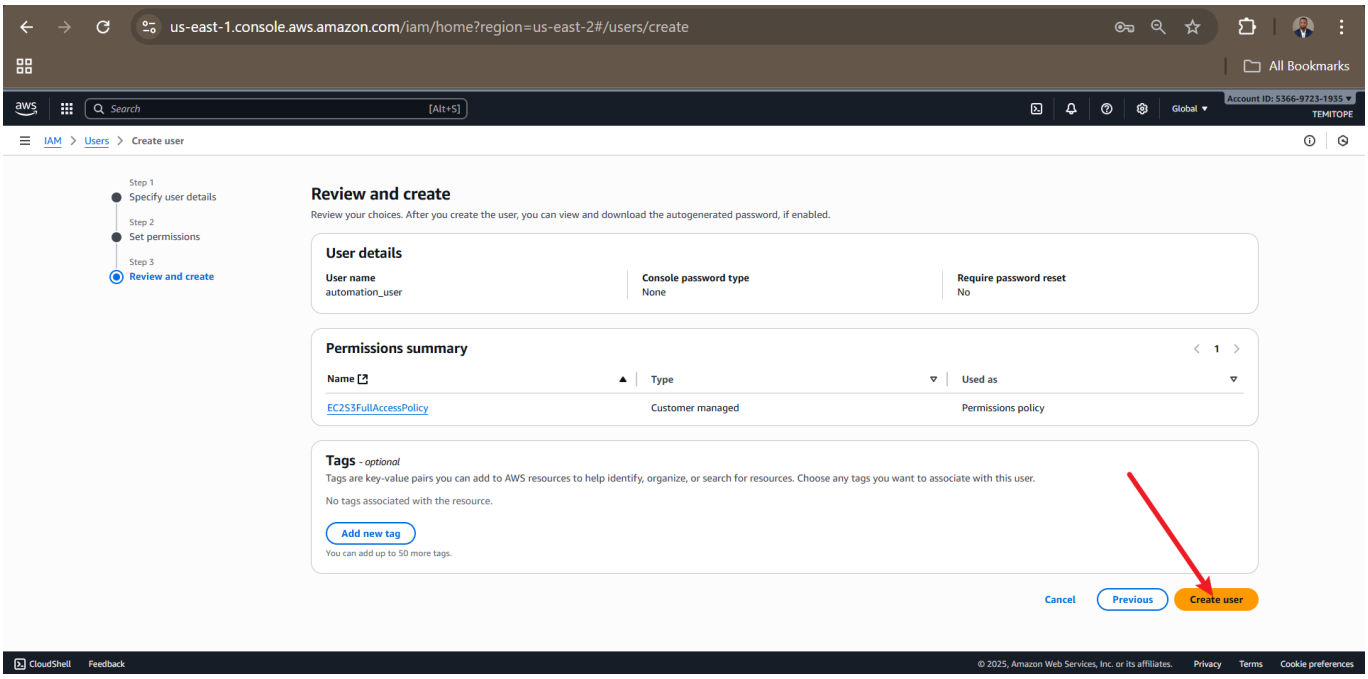
Step 18: Attach Policy to User

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



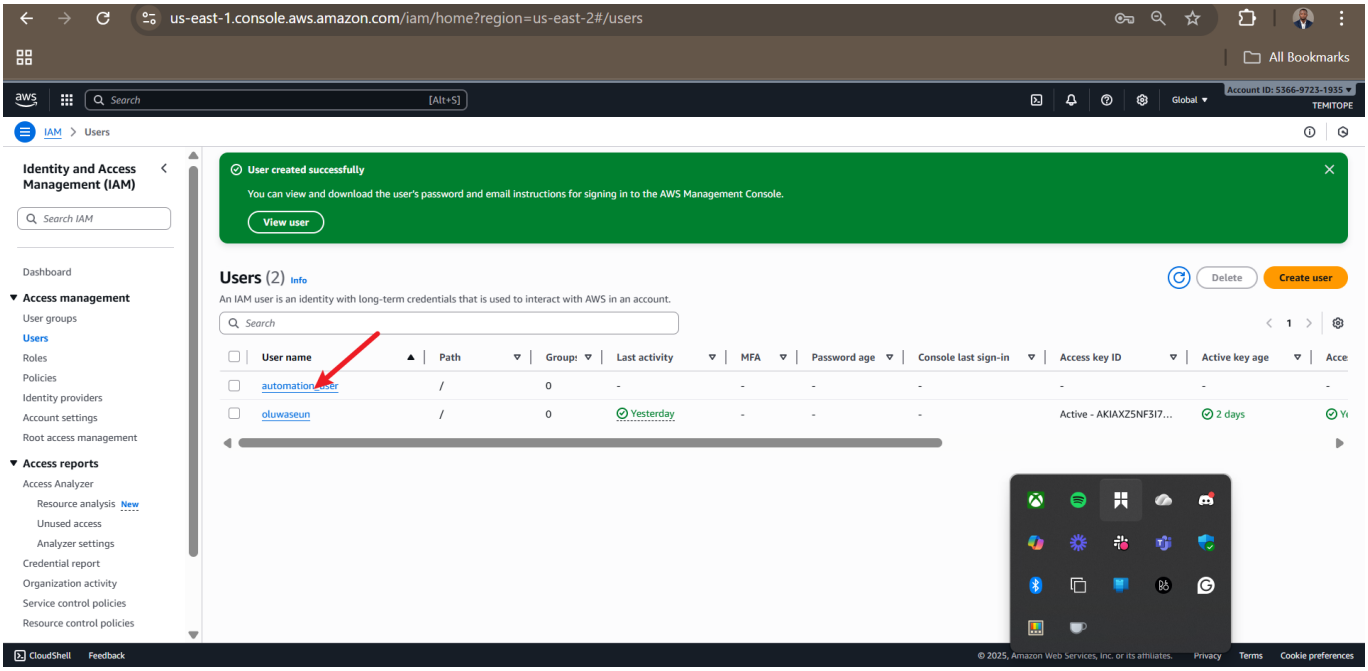
Step 19: Review and Create User

- Review user configuration
- Click Create user



Step 20: User Creation Success

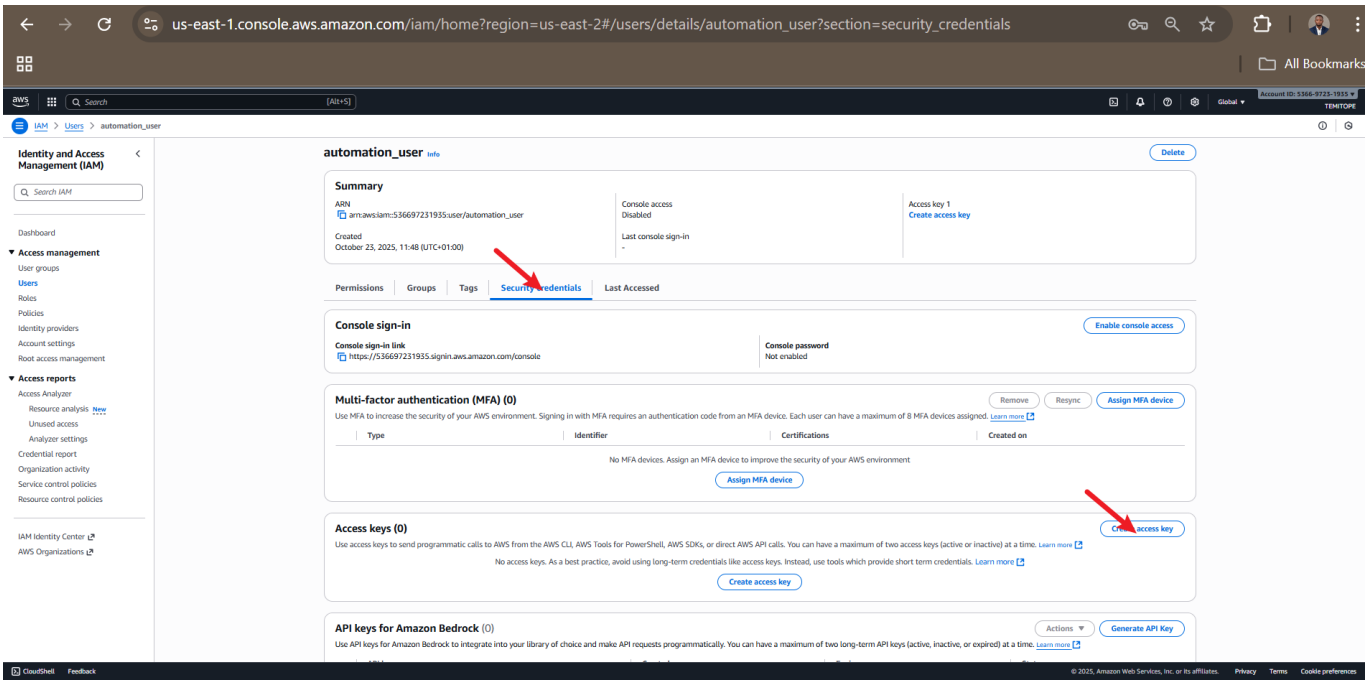
- automation_user appears in Users list
- Click user for credential setup



Phase 5: Generate Access Keys

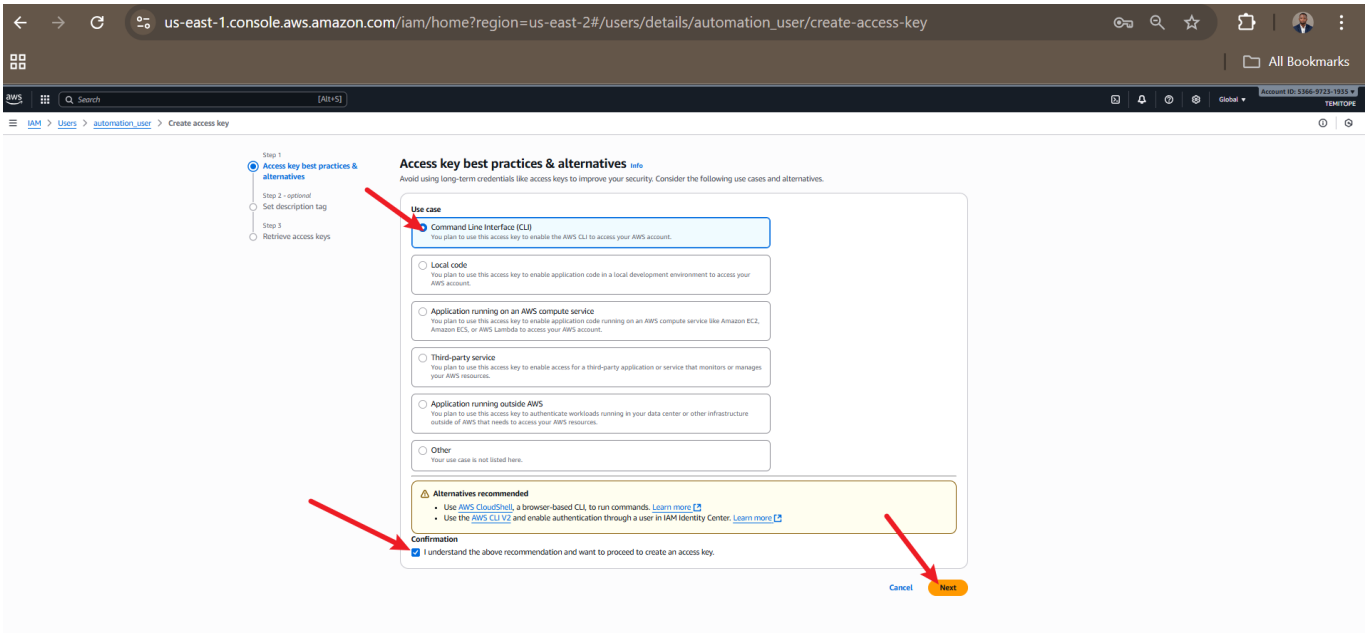
Step 21: Access Security Credentials

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



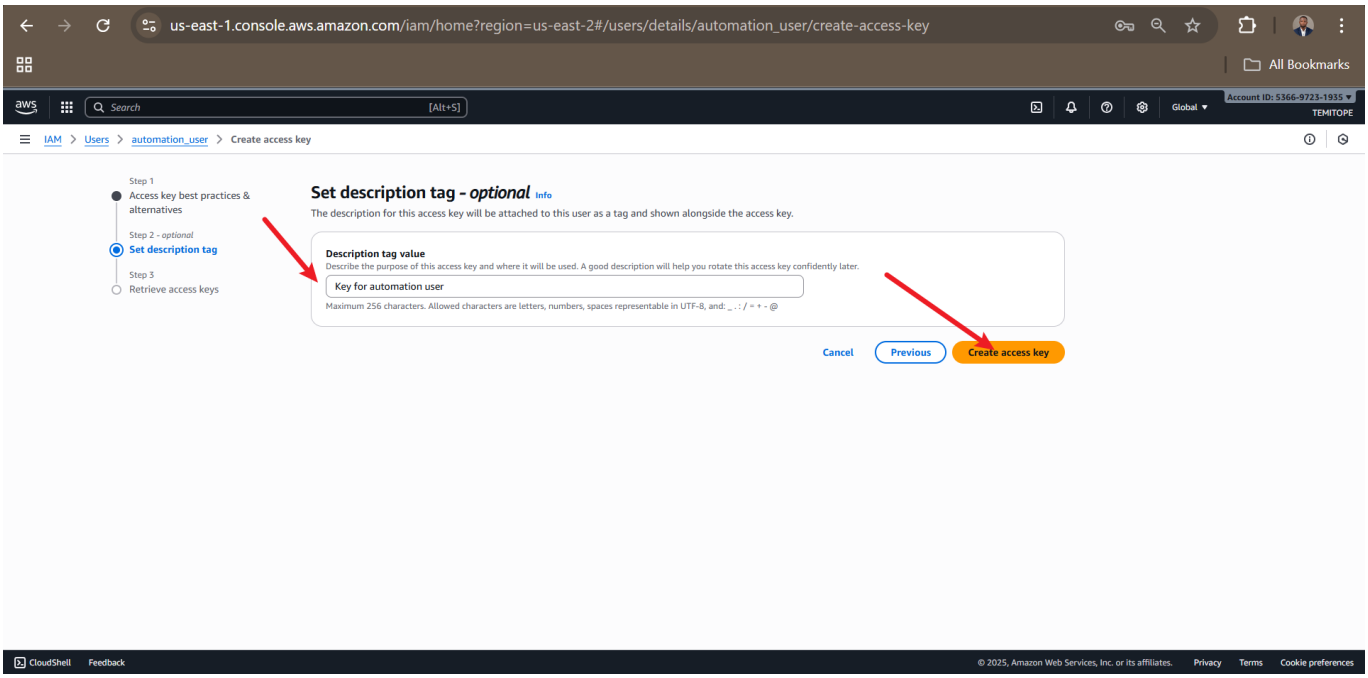
Step 22: Select CLI Use Case

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



Step 23: Add Description (Optional)

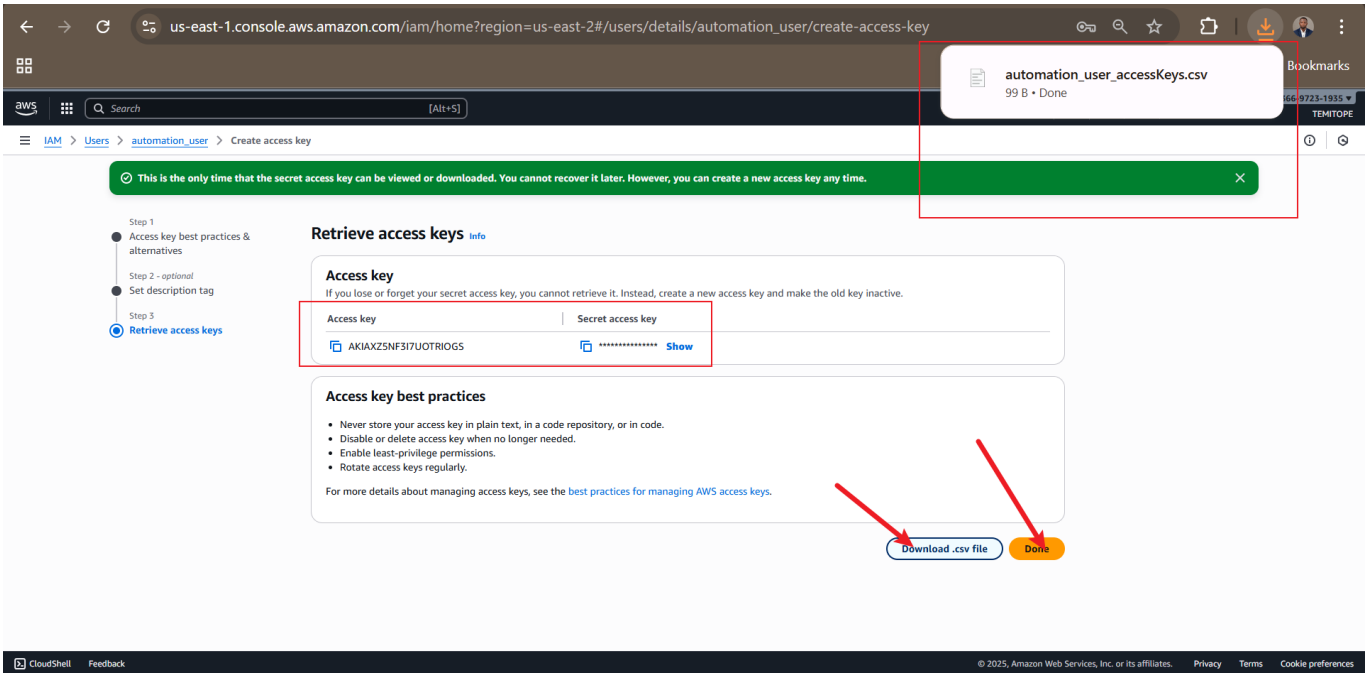
- Add optional description tag
- Click Create access key



Step 24: Download Credentials

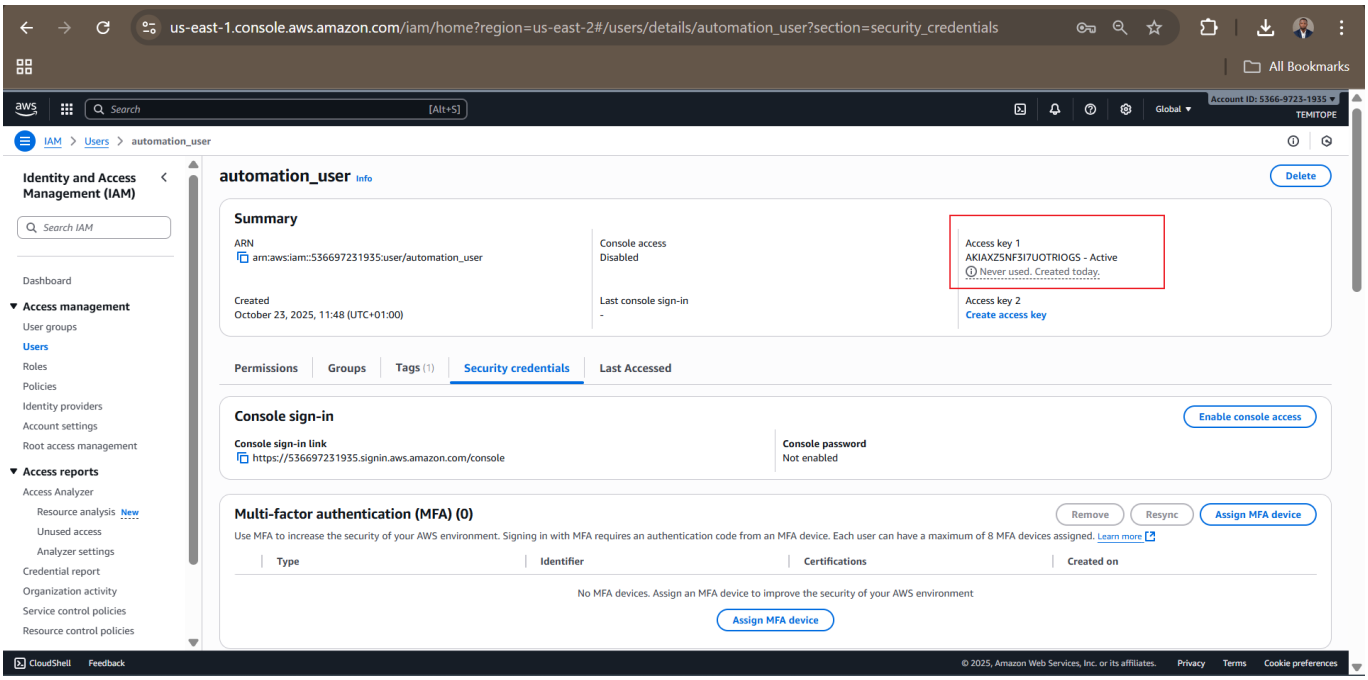
Critical: Download .csv file immediately

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



Step 25: Verify Key Status

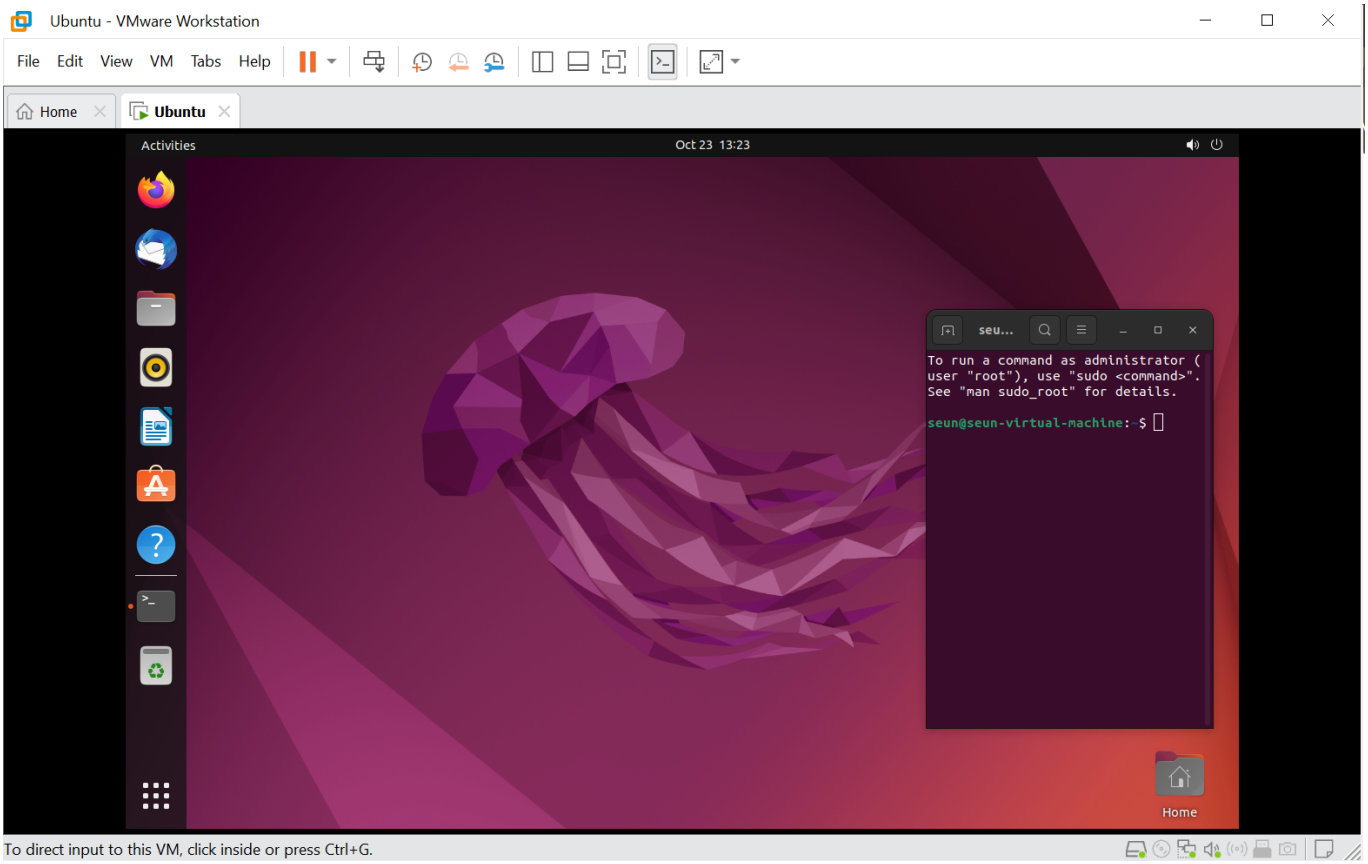
- Access key status: Active
- Usage: Unused (expected)



Phase 6: AWS CLI Installation (Ubuntu)

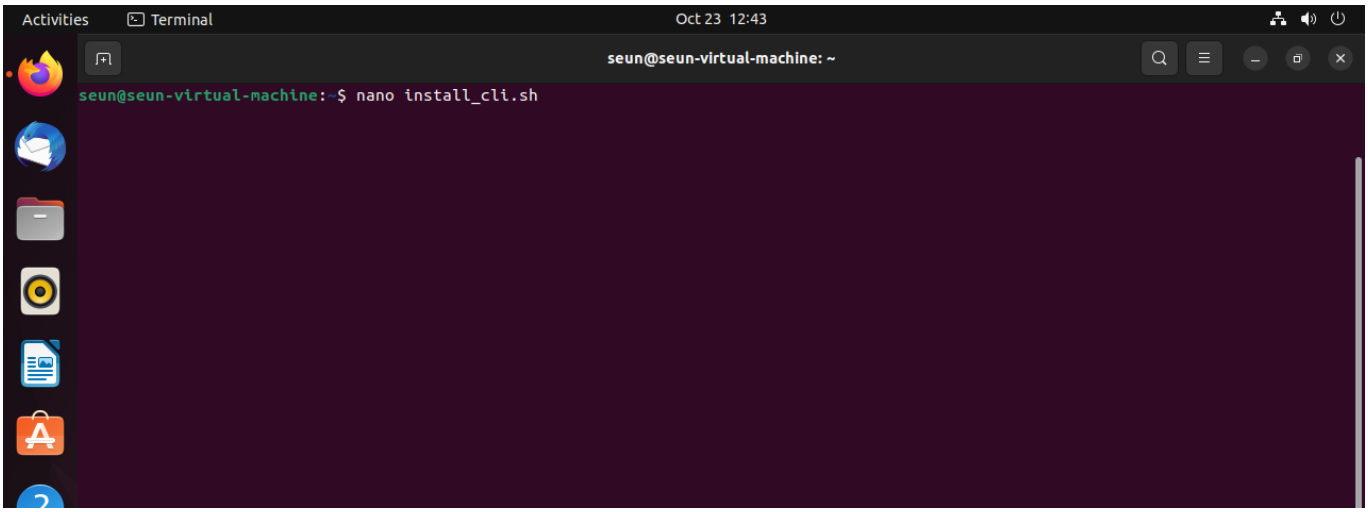
Step 26: Ubuntu Environment Setup

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



Step 27: Create Installation Script

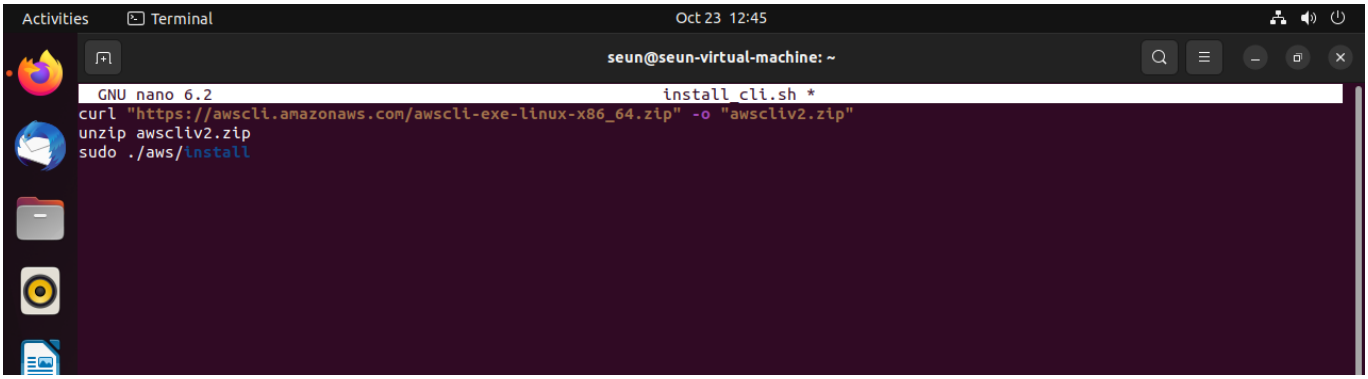
- `nano install_cli.sh`



Step 28: Write Installation Commands

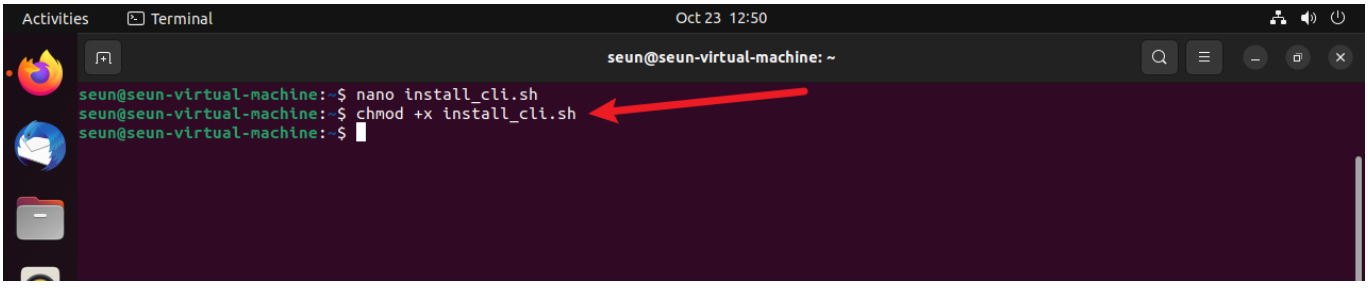
```
#!/bin/bash
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
```

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



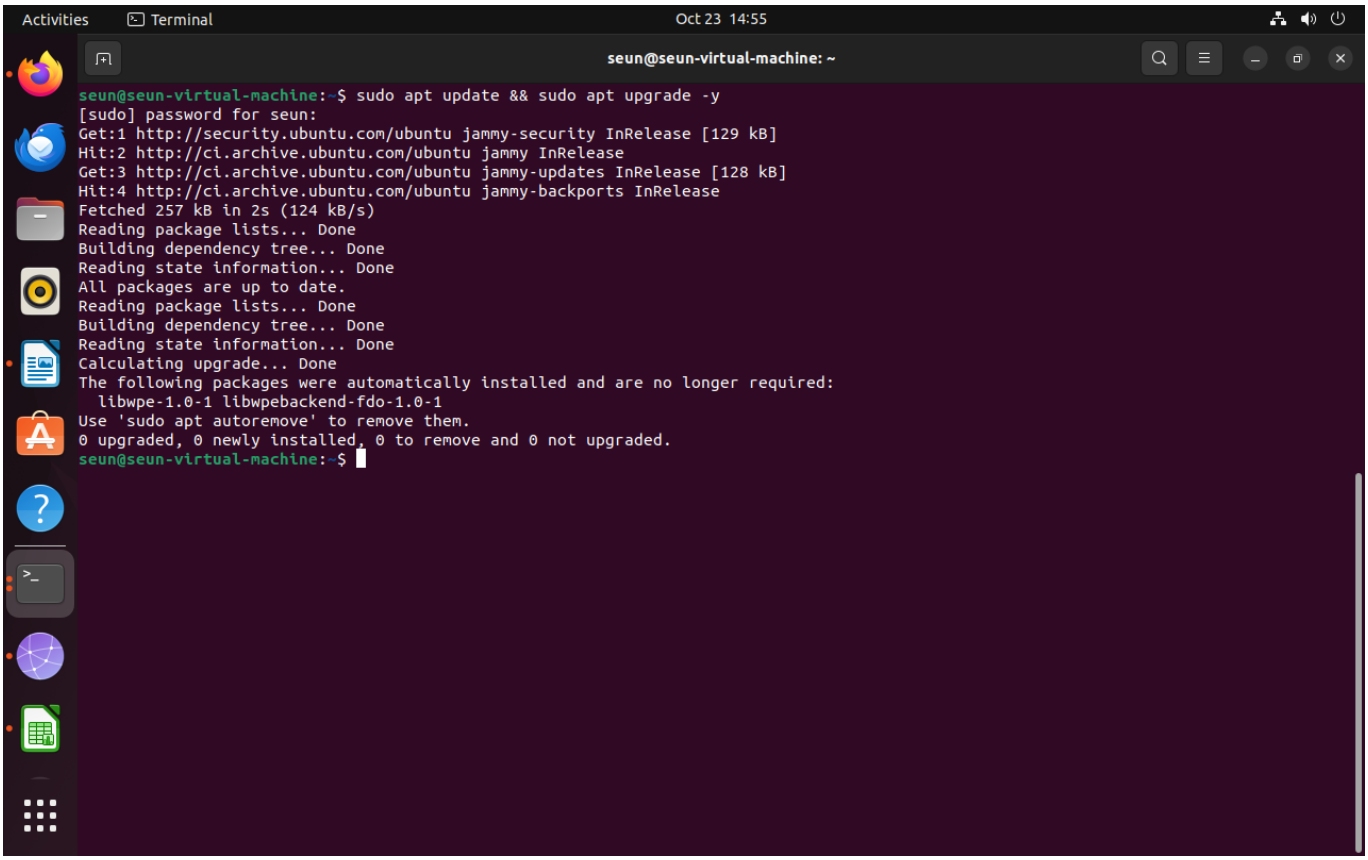
Step 29: Make Script Executable

- `chmod +x install_cli.sh`



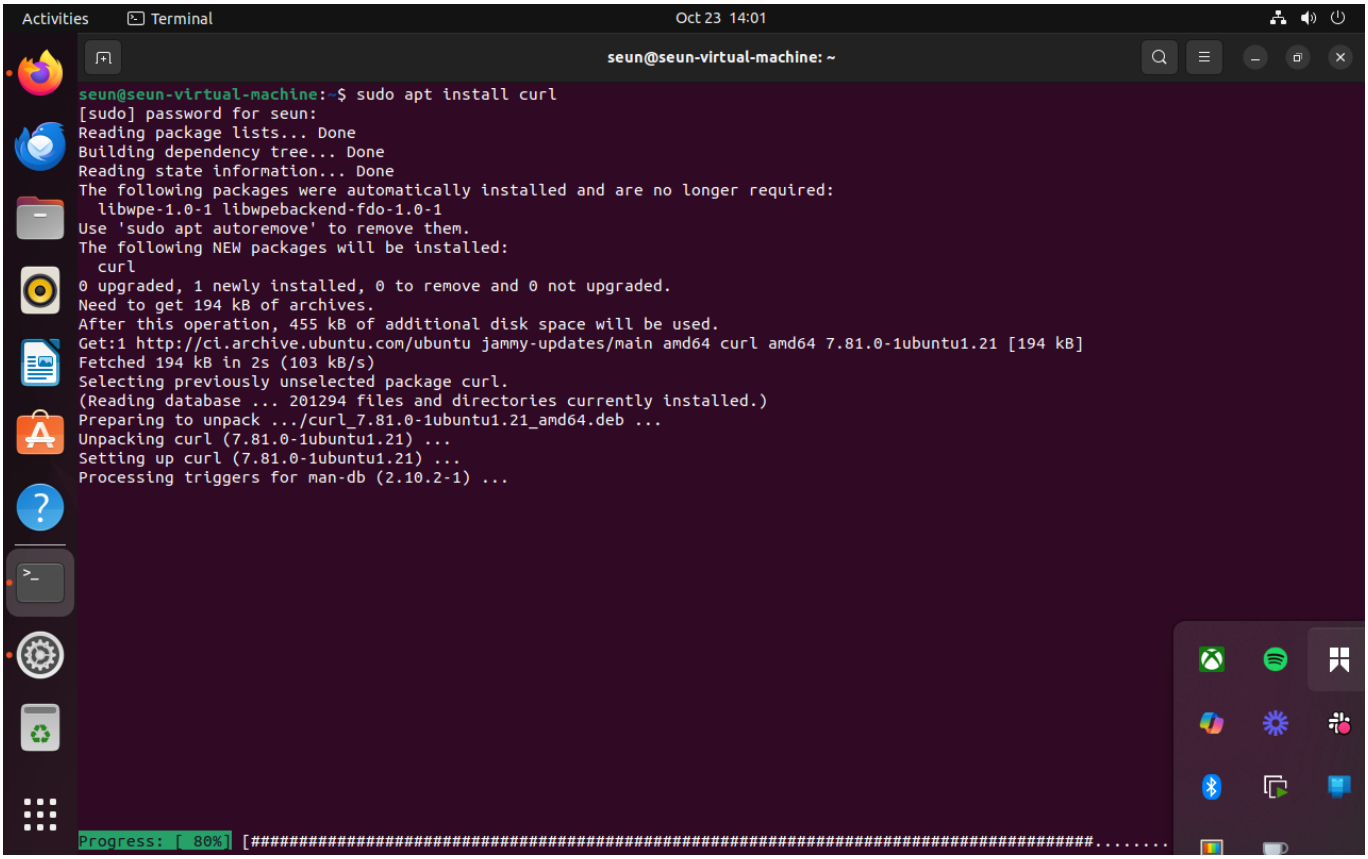
Step 30: System Update

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



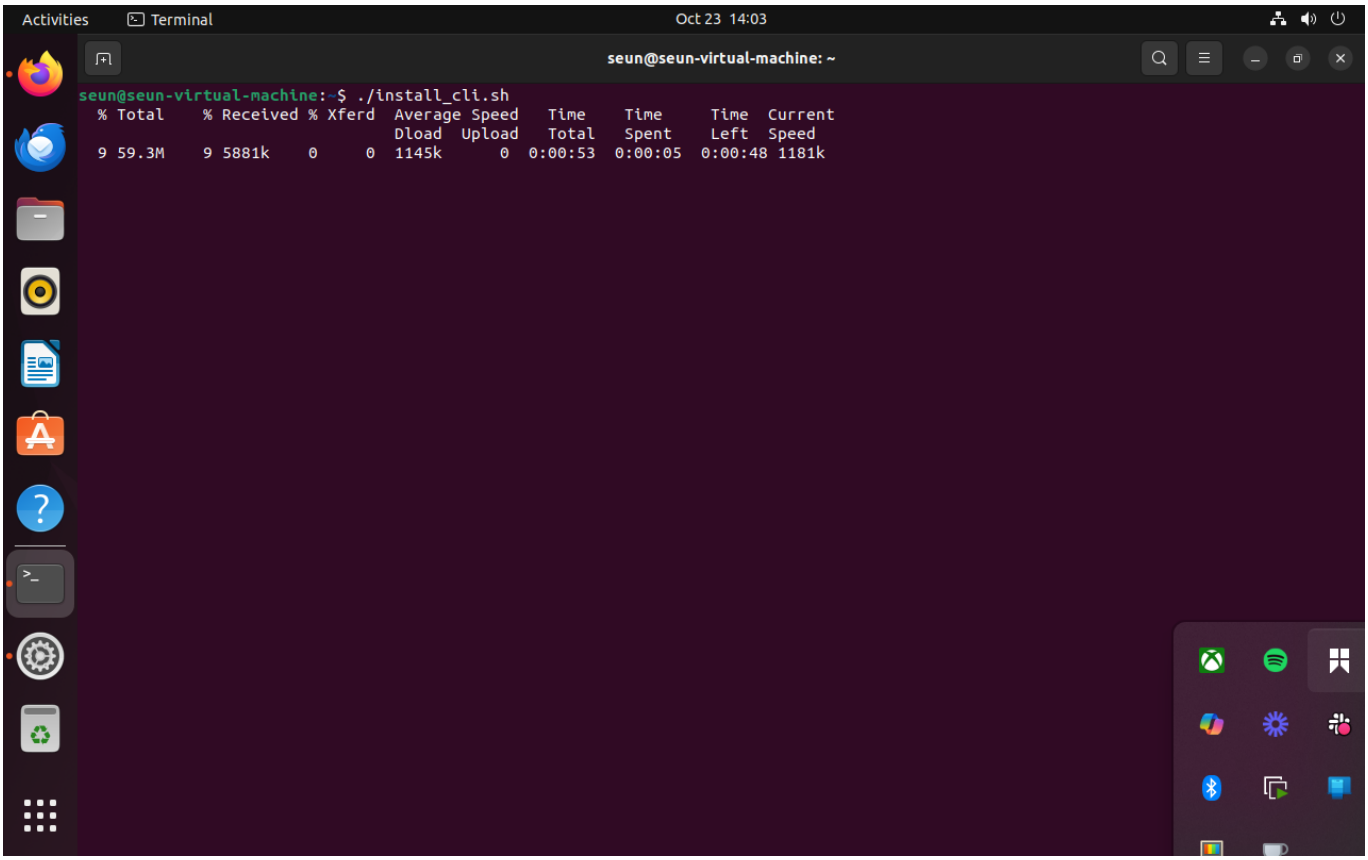
Step 31: Install Curl

- `sudo apt install curl`



Step 32: Execute Installation Script

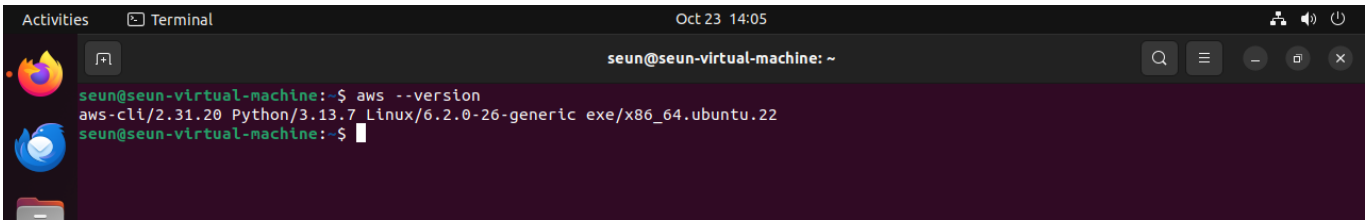
- `./install_cli.sh`



Step 33: Verify AWS CLI Installation

- `aws --version`

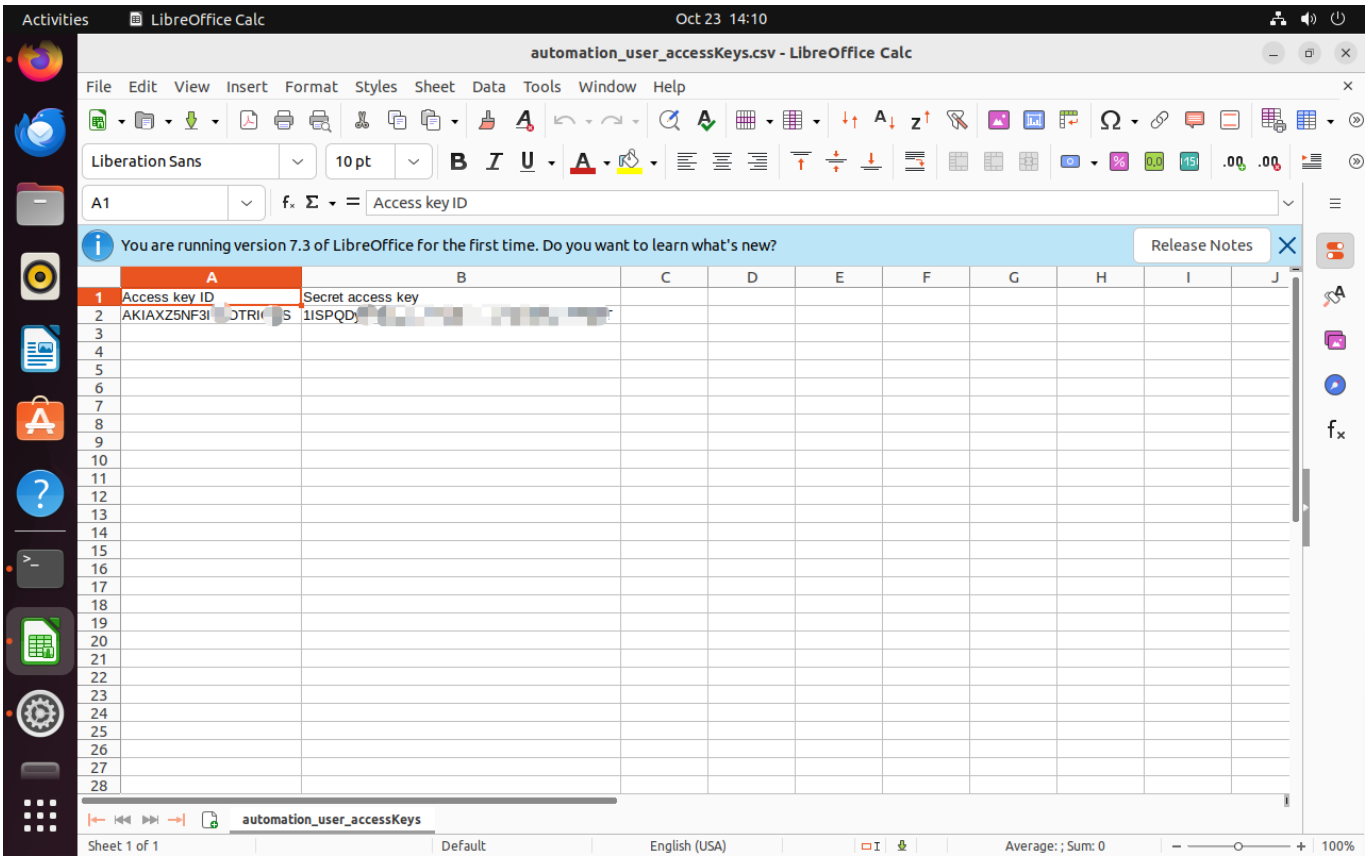
Expected Output: aws-cli/2.15.30 Python/3.11.6 Linux/...



Phase 7: AWS CLI Configuration & Testing

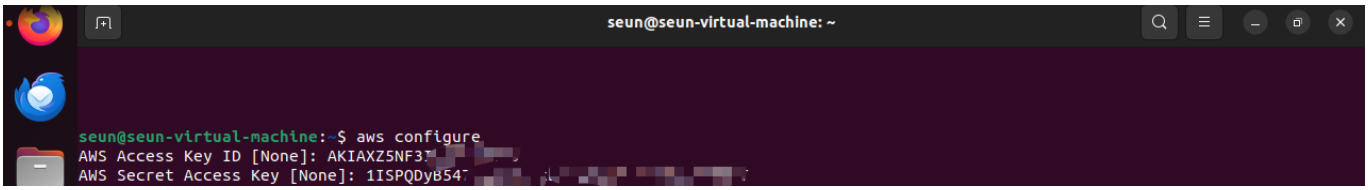
Step 34: Extract Credentials from CSV

- 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]: ...



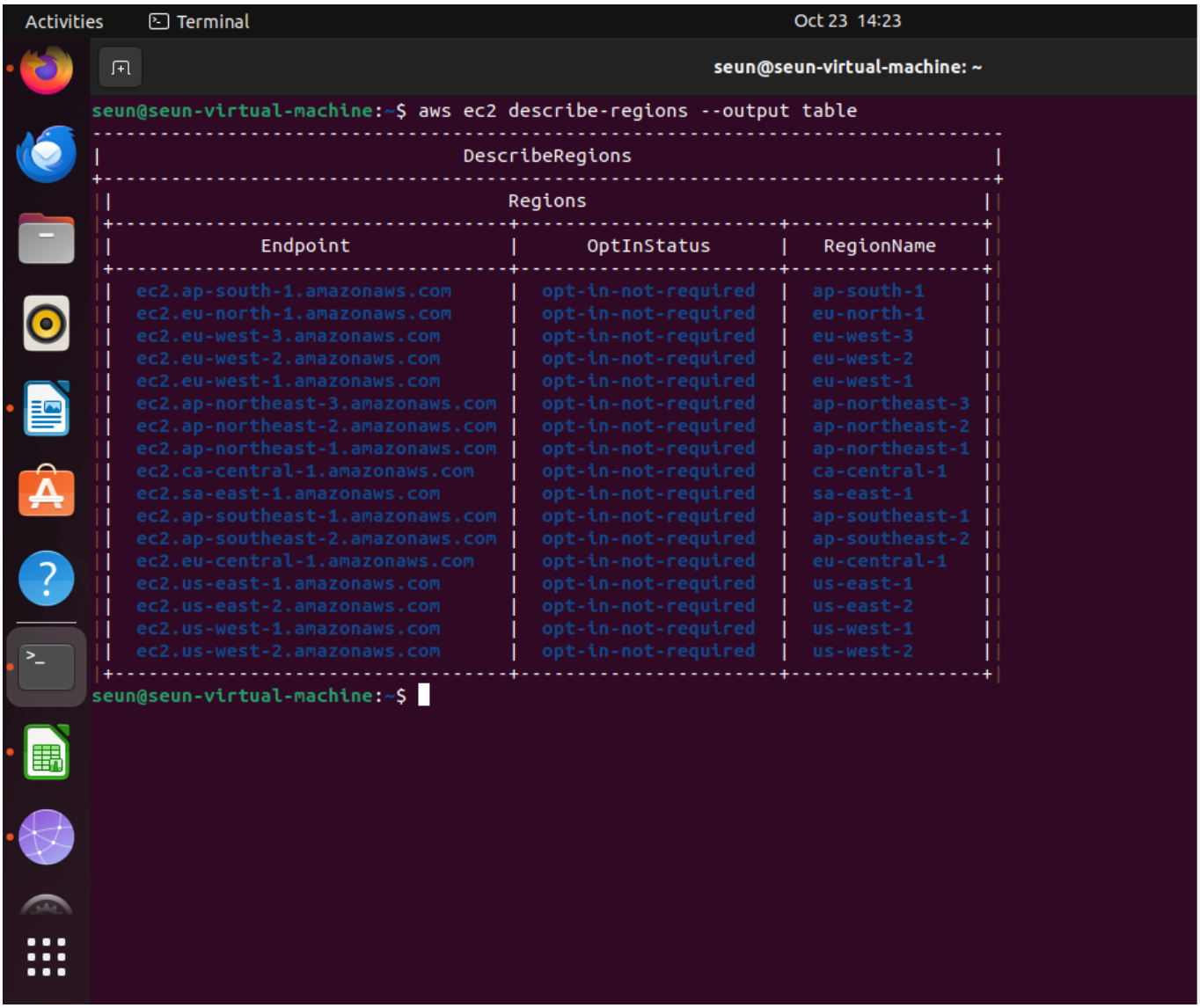
Step 36: Configure Region & Output

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

Step 37: Test API Connectivity

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

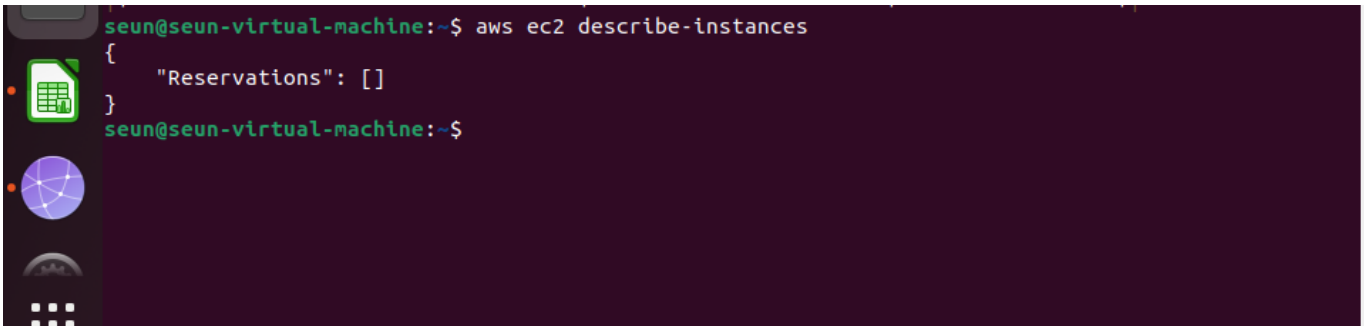
☑ Success: Complete regions table displayed



Step 38: Test EC2 Instance Listing

- `aws ec2 describe-instances`

☑ Success: Empty response (no instances = correct)

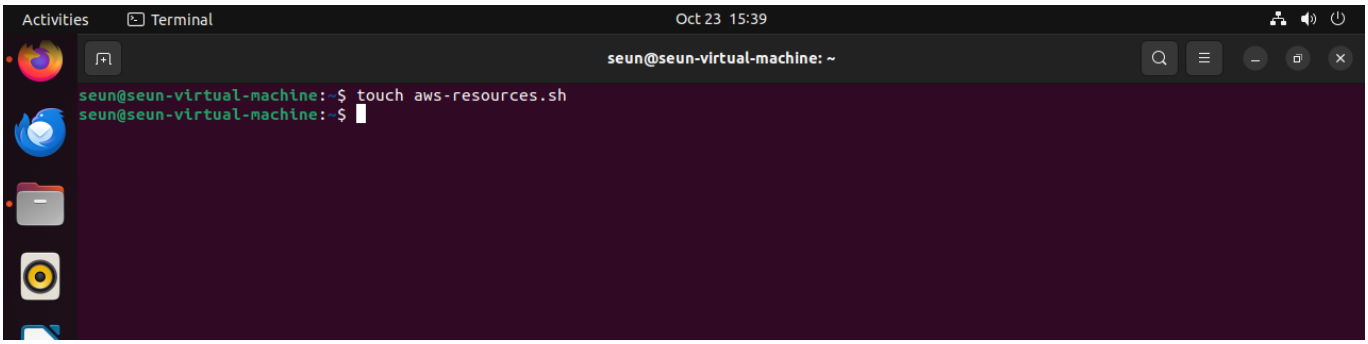


Phase 8: Prepare Automation Script and Resources

Step 39: Create Automation Script File

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

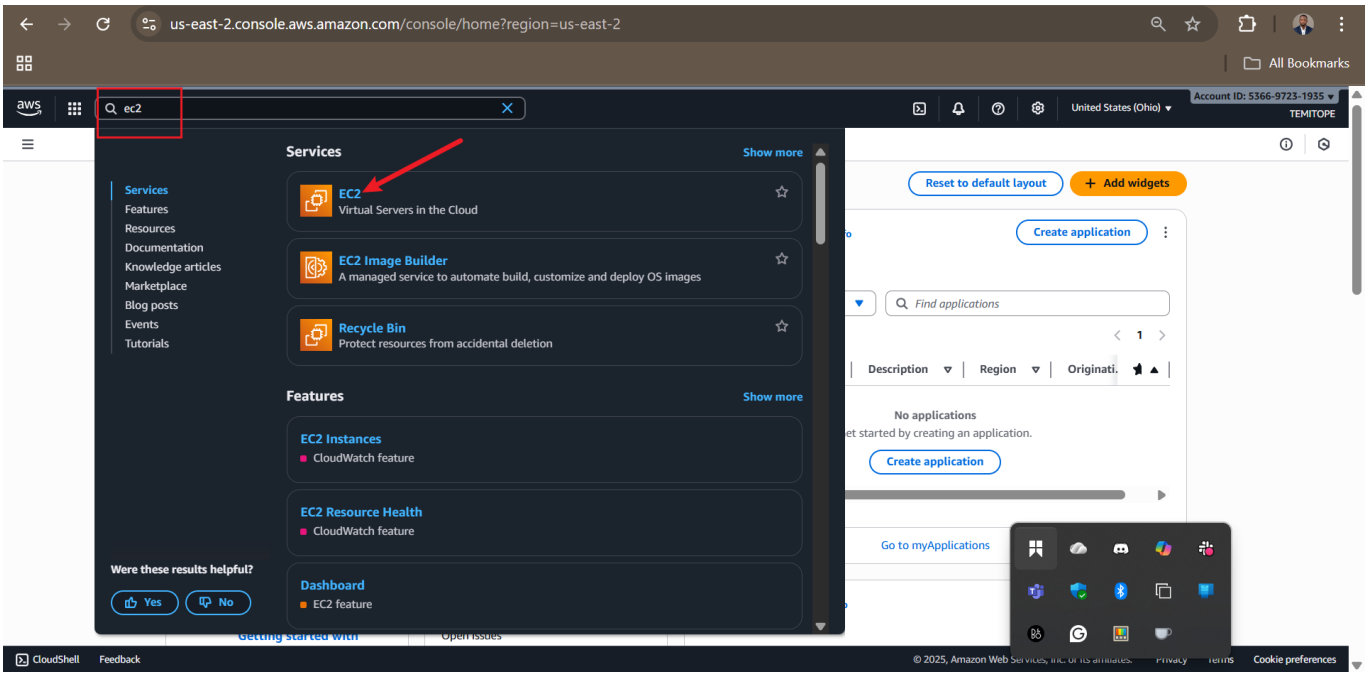
Purpose: This script will automate EC2 instance creation and S3 bucket management.



Step 49: Navigate to EC2 Service

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

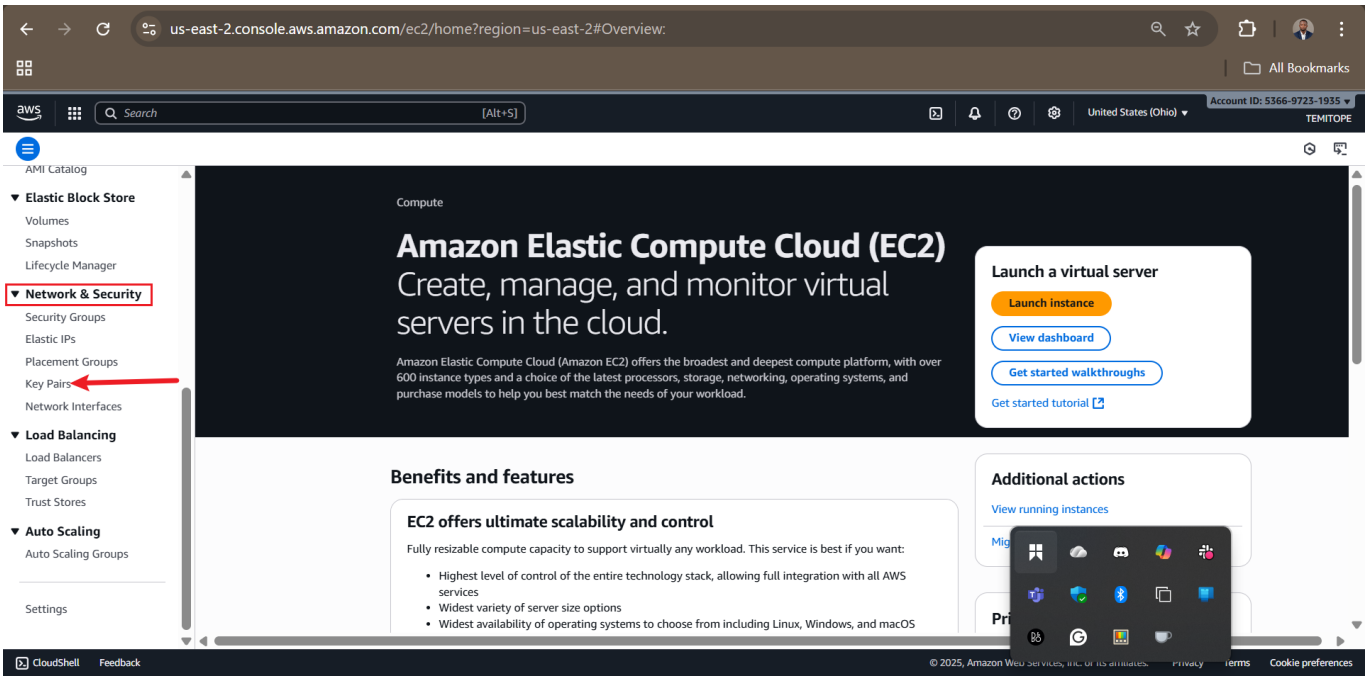
Purpose: Access EC2 dashboard for key pair creation.



Step 50: Access Key Pairs Section

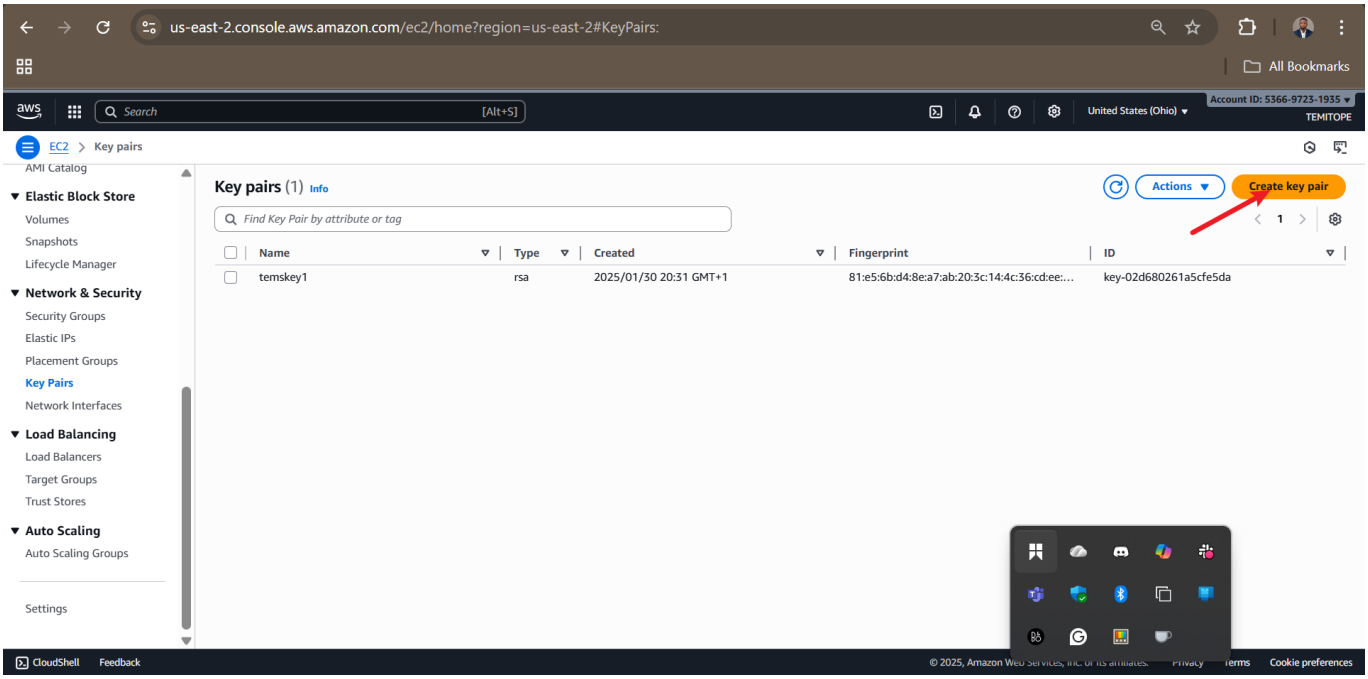
- Find "Key Pairs" under "Network and Security" on the left sidebar and click on it

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



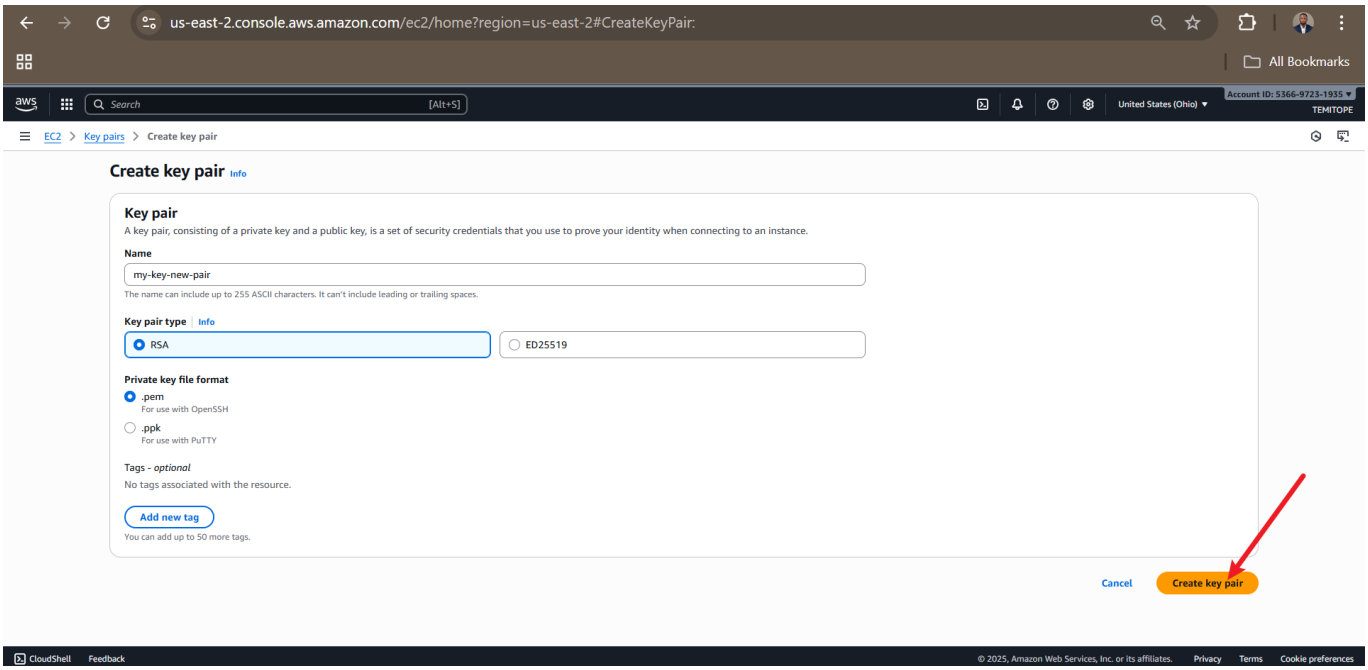
Step 51: Initiate Key Pair Creation

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



Step 52: Configure and Create Key Pair

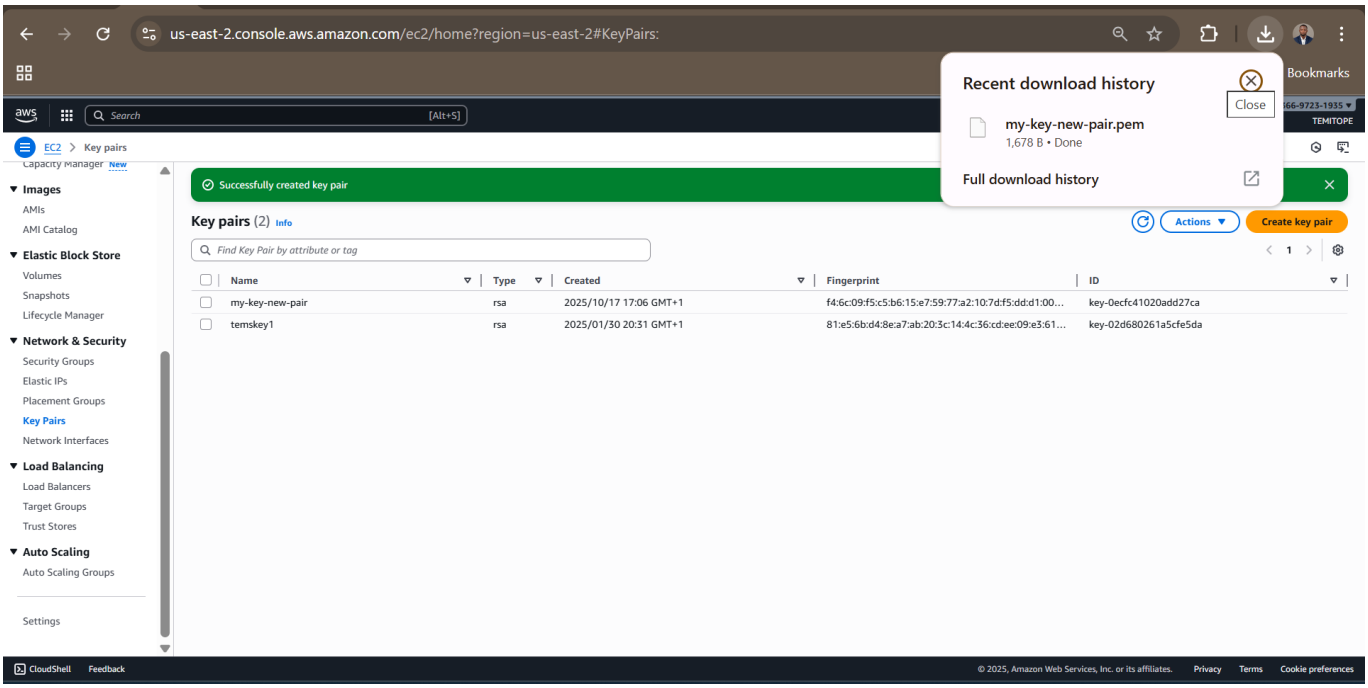
- Name the key pair (e.g., MyKeyPair)
- Select key type (e.g., RSA) and file format (e.g., .pem)
- Click "Create key pair"



Step 53: Verify Key Pair Creation

- Key pair created and .pem file downloaded

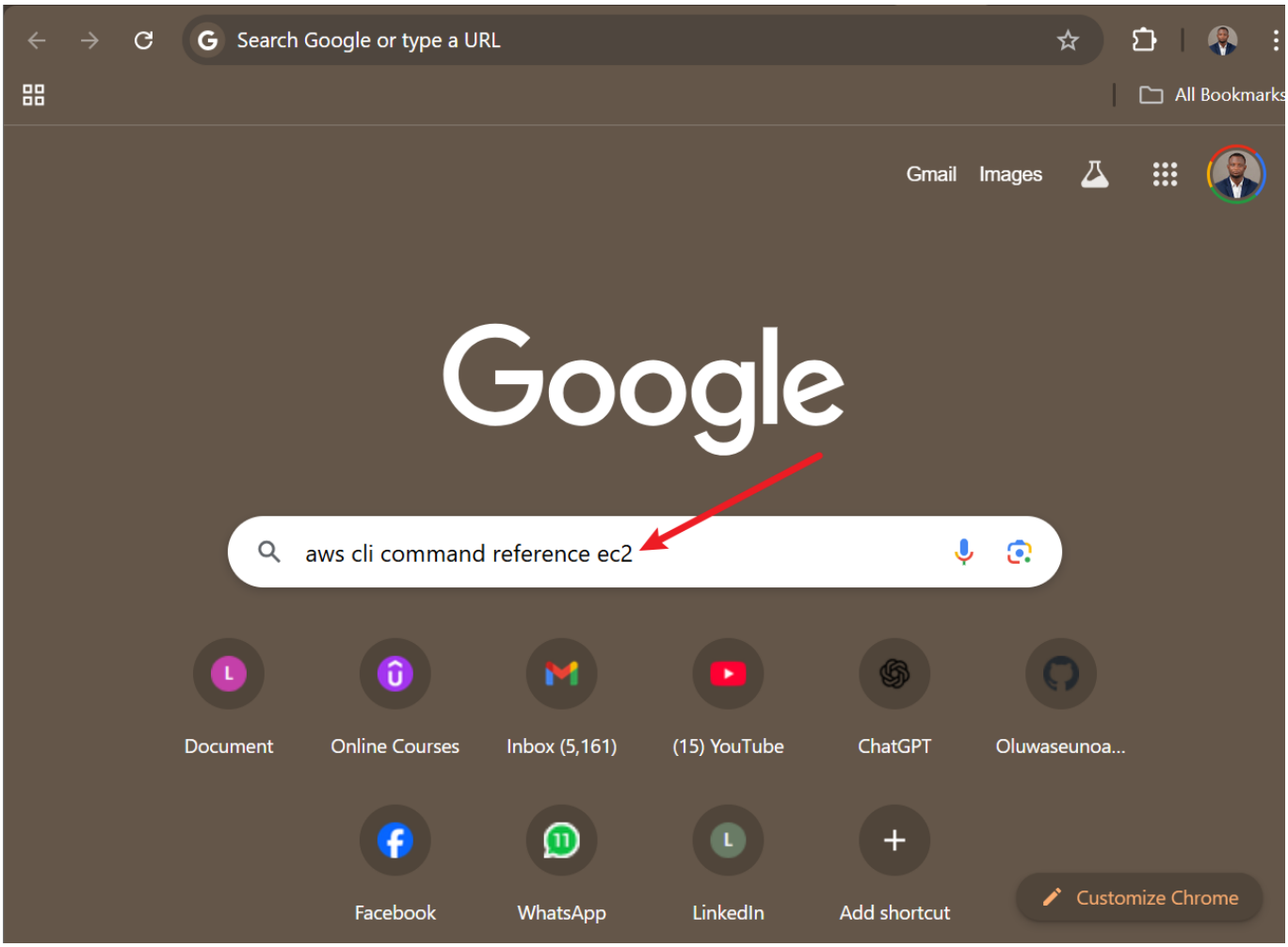
Purpose: Use this key pair in the automation script for EC2 launches.



Step 54: Search for AWS CLI EC2 Reference

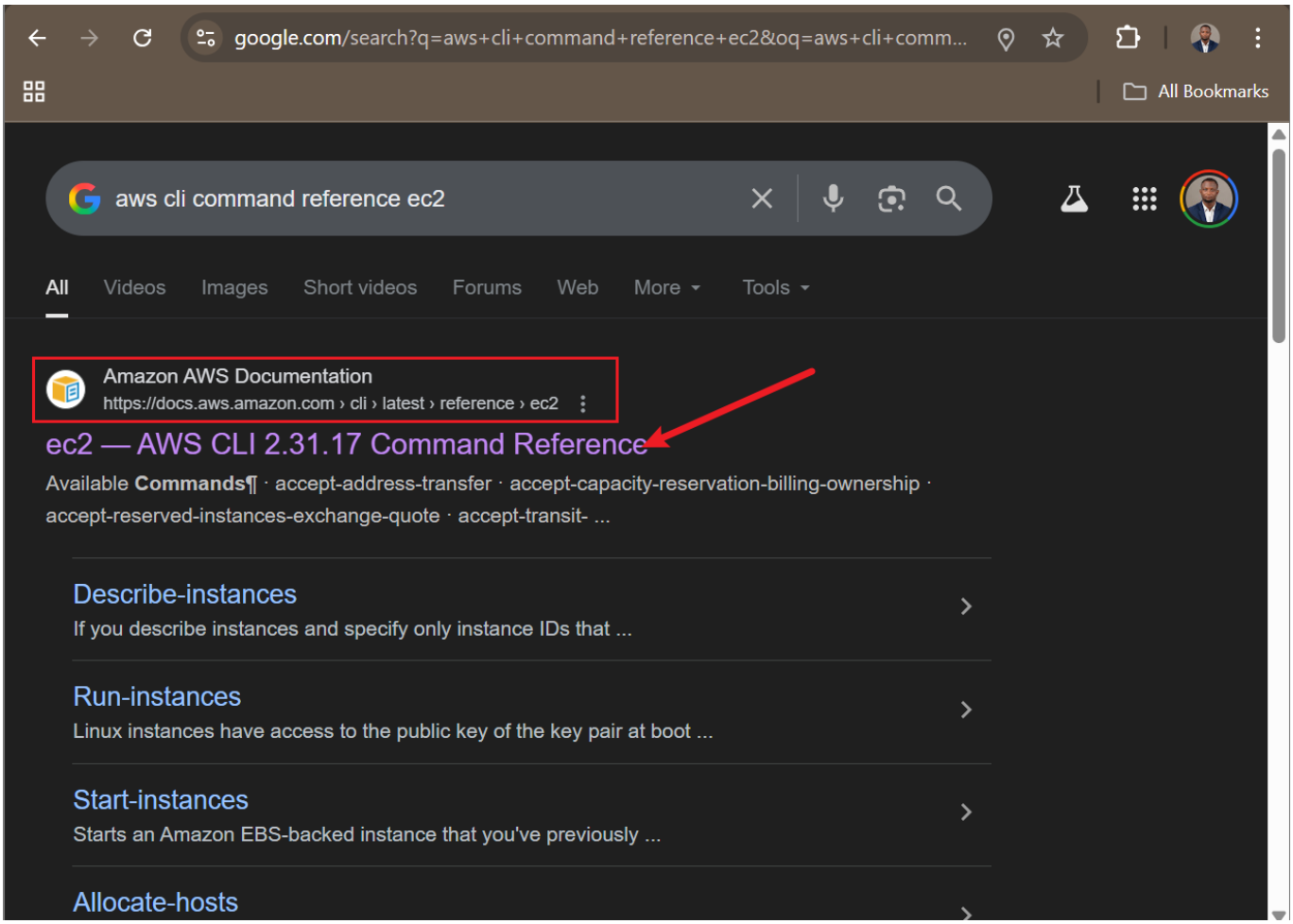
- In your preferred browser, search for "aws cli command reference ec2"

Purpose: Reference official documentation for scripting EC2 commands.



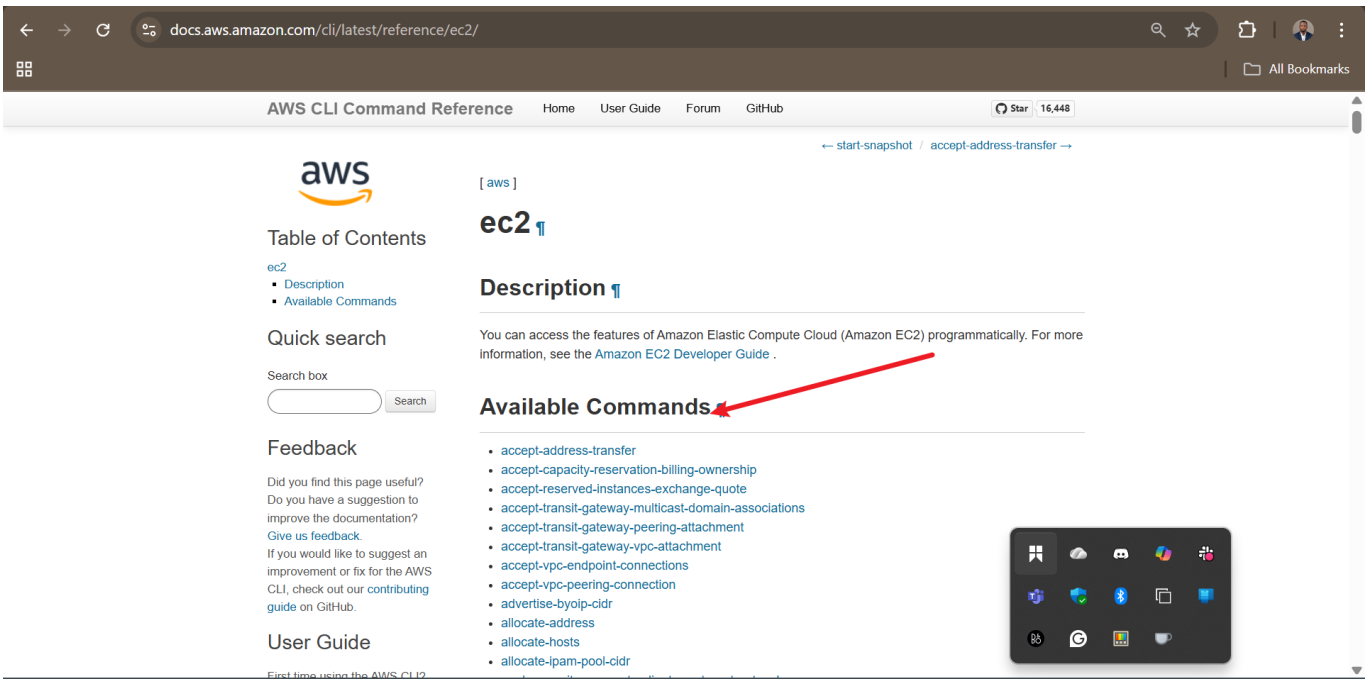
Step 55: Access CLI Reference

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



Step 56: Explore Documentation

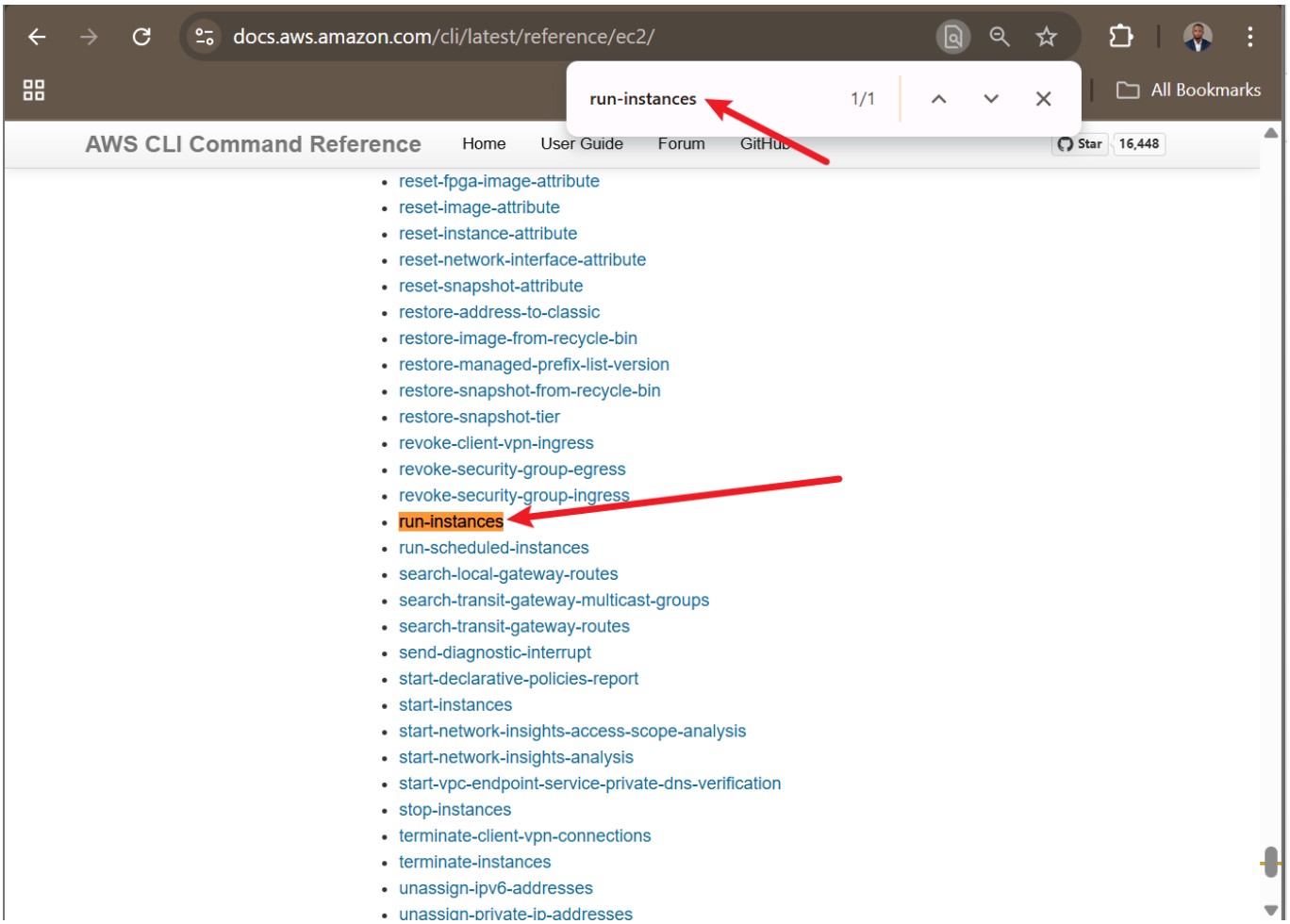
- Now on the AWS CLI commands official documentation webpage



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.



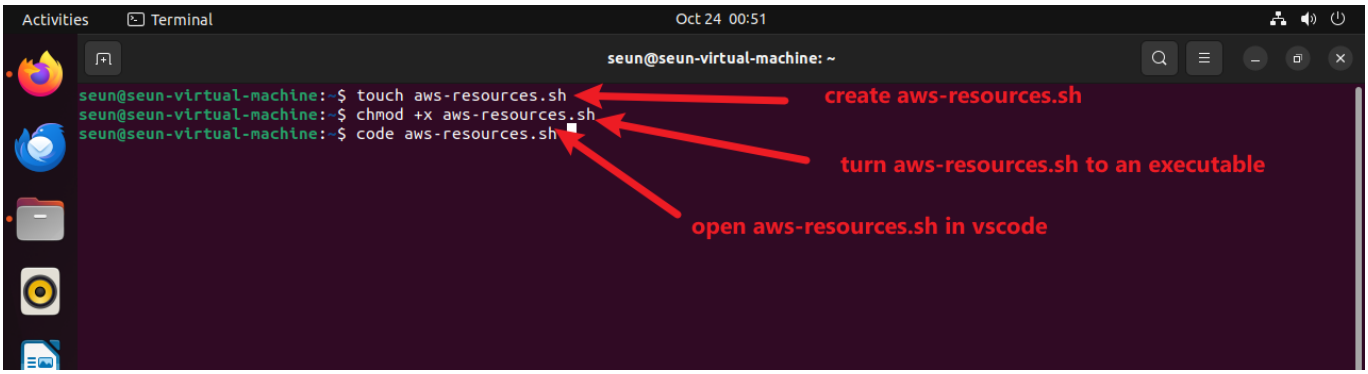
Phase 9: Develop the Automation Shell Script

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.

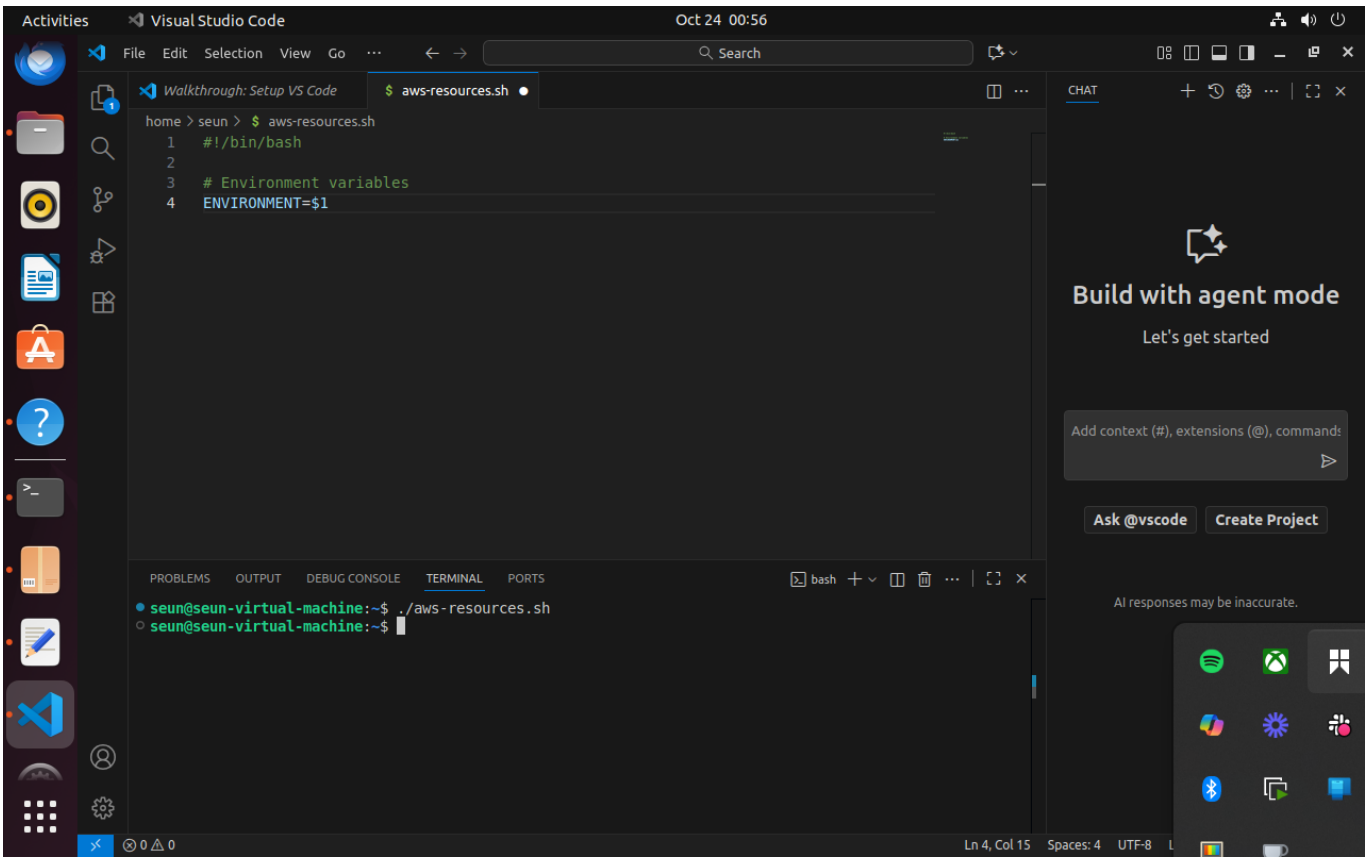


Step 59: Add Shebang and ENVIRONMENT Variable

Add the following to the script:

```
#!/bin/bash

# Environment variables
ENVIRONMENT=$1"
```

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



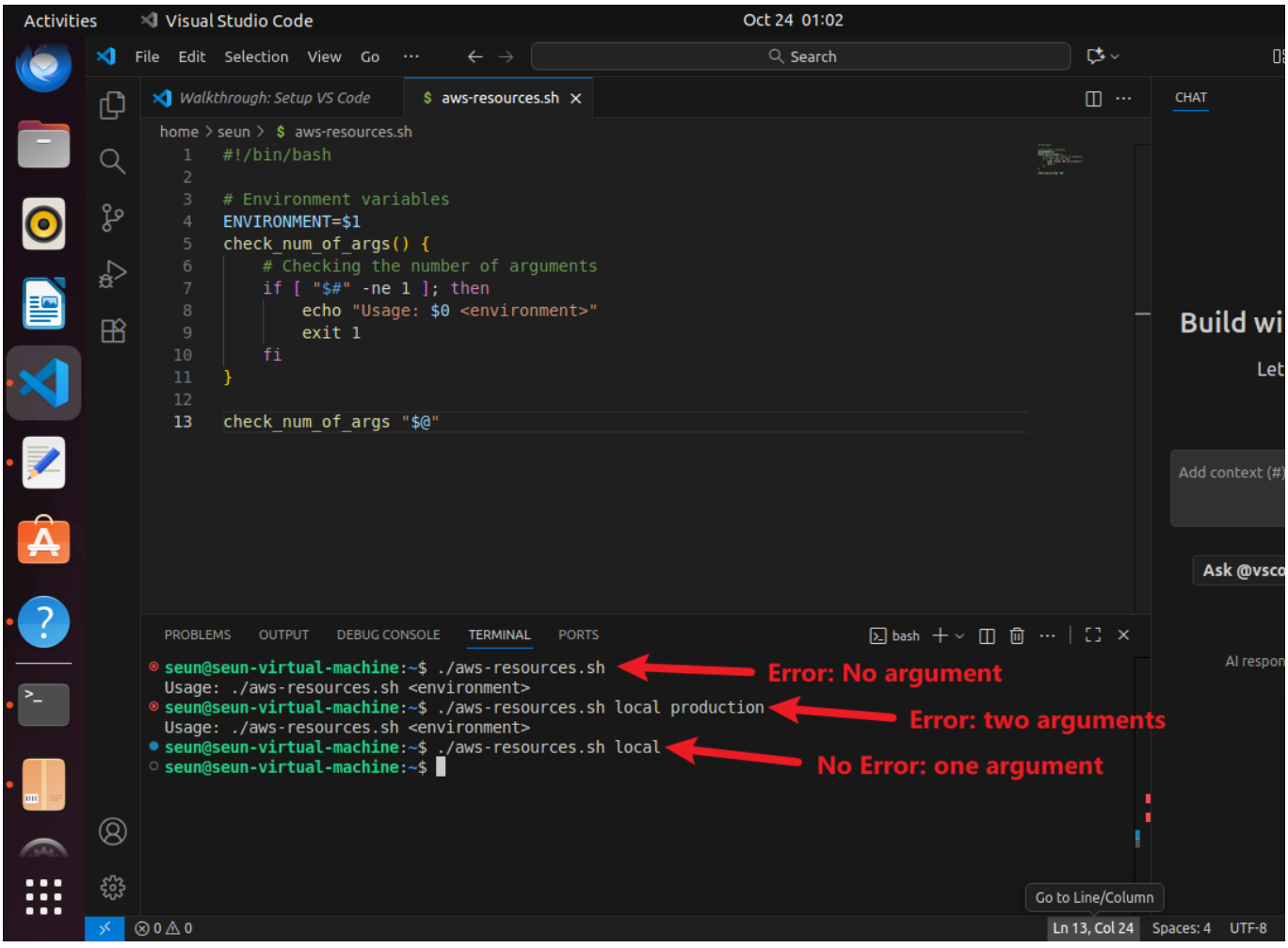
Step 60: Add Argument Check Function

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.

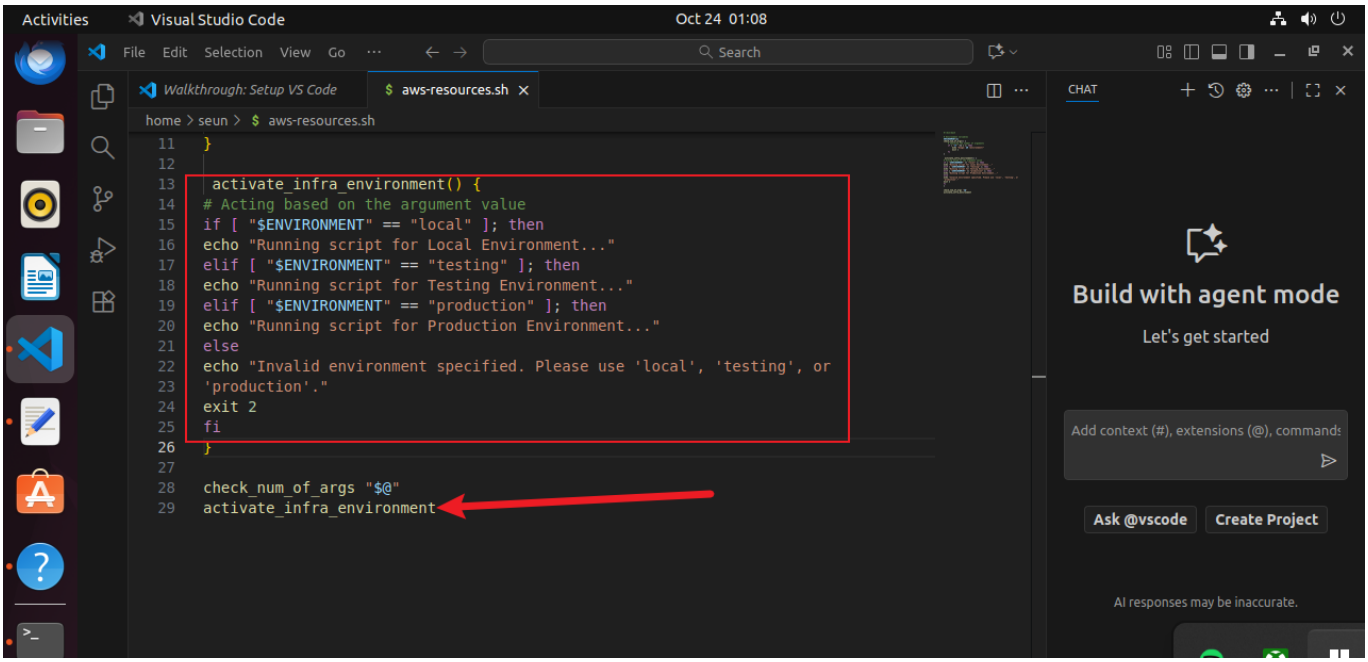


Step 61: Add Environment Activation Function

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
```

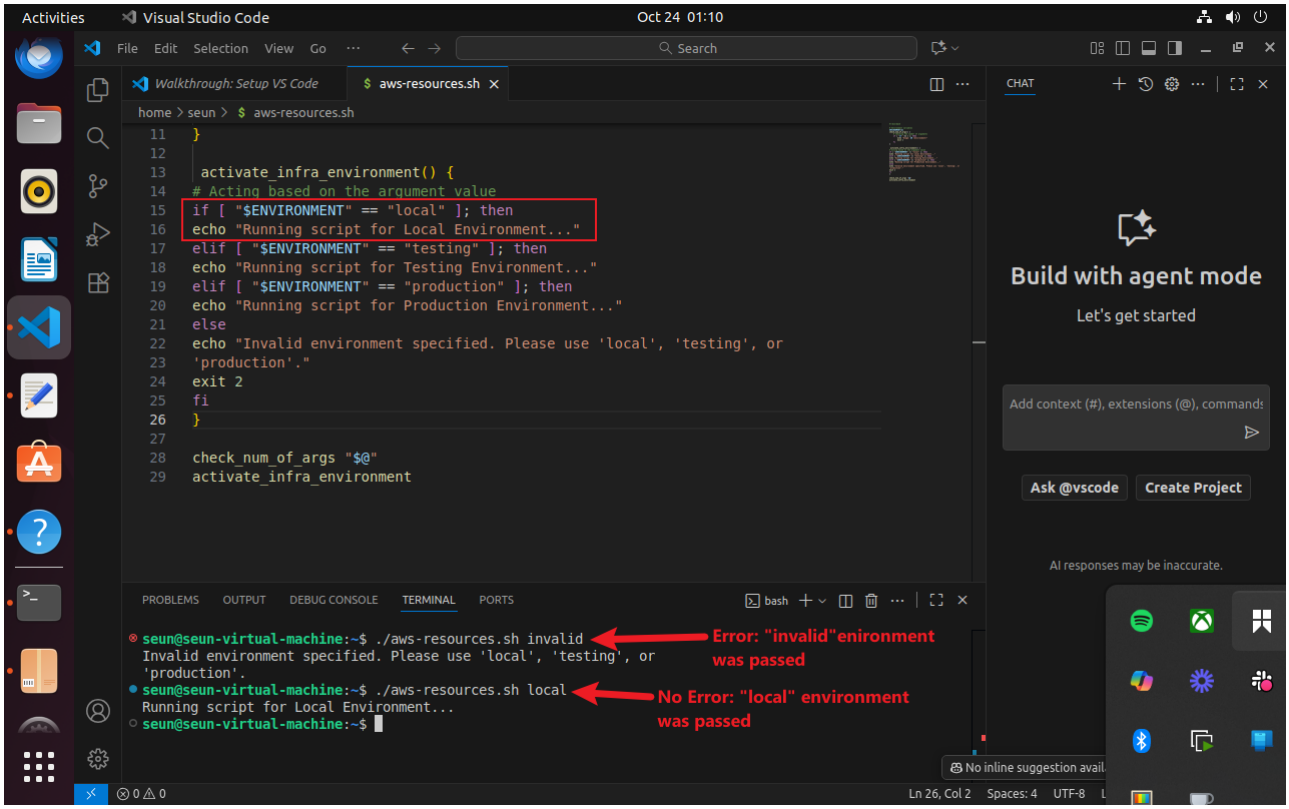
```
echo "Running script for Testing Environment..."
elif [ "$ENVIRONMENT" == "production" ]; then
echo "Running script for Production Environment..."
else
echo "Invalid environment specified. Please use 'local', 'testing', or
'production'."
exit 2
fi
}
activate_infra_environment
```



Step 62: Test Environment Activation

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

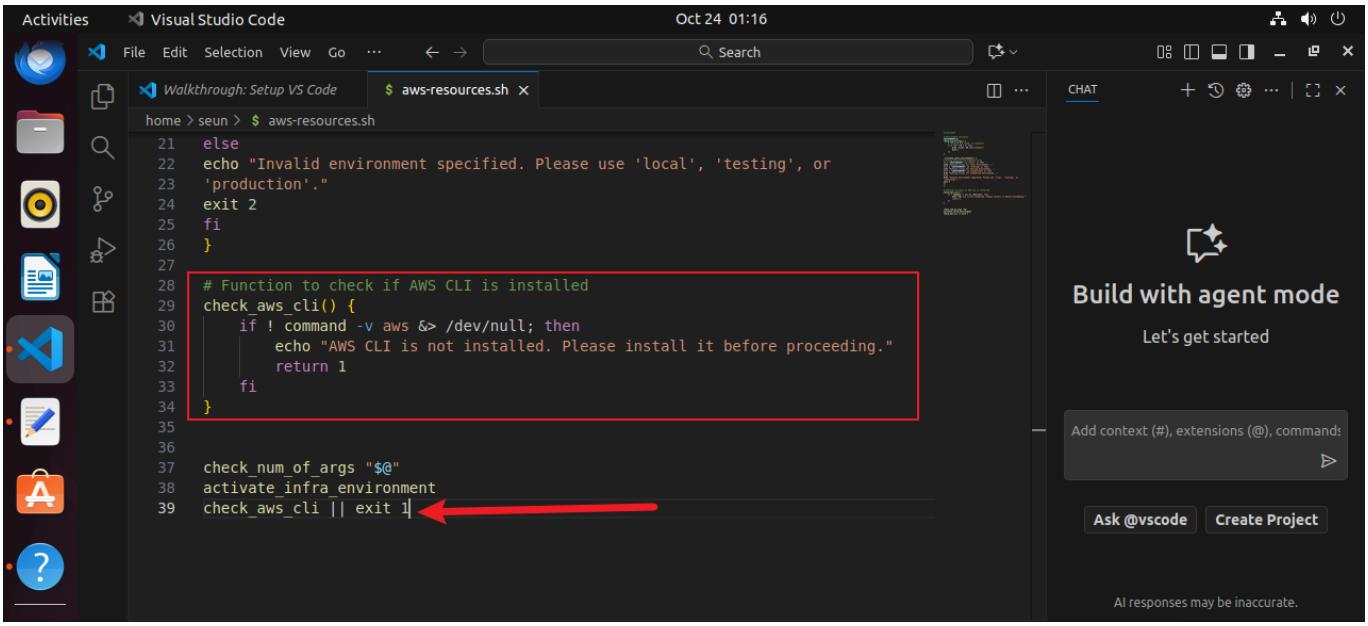
- Run with "local" environment (success) `./aws-resources.sh local`



Step 63: Add AWS CLI Check Function

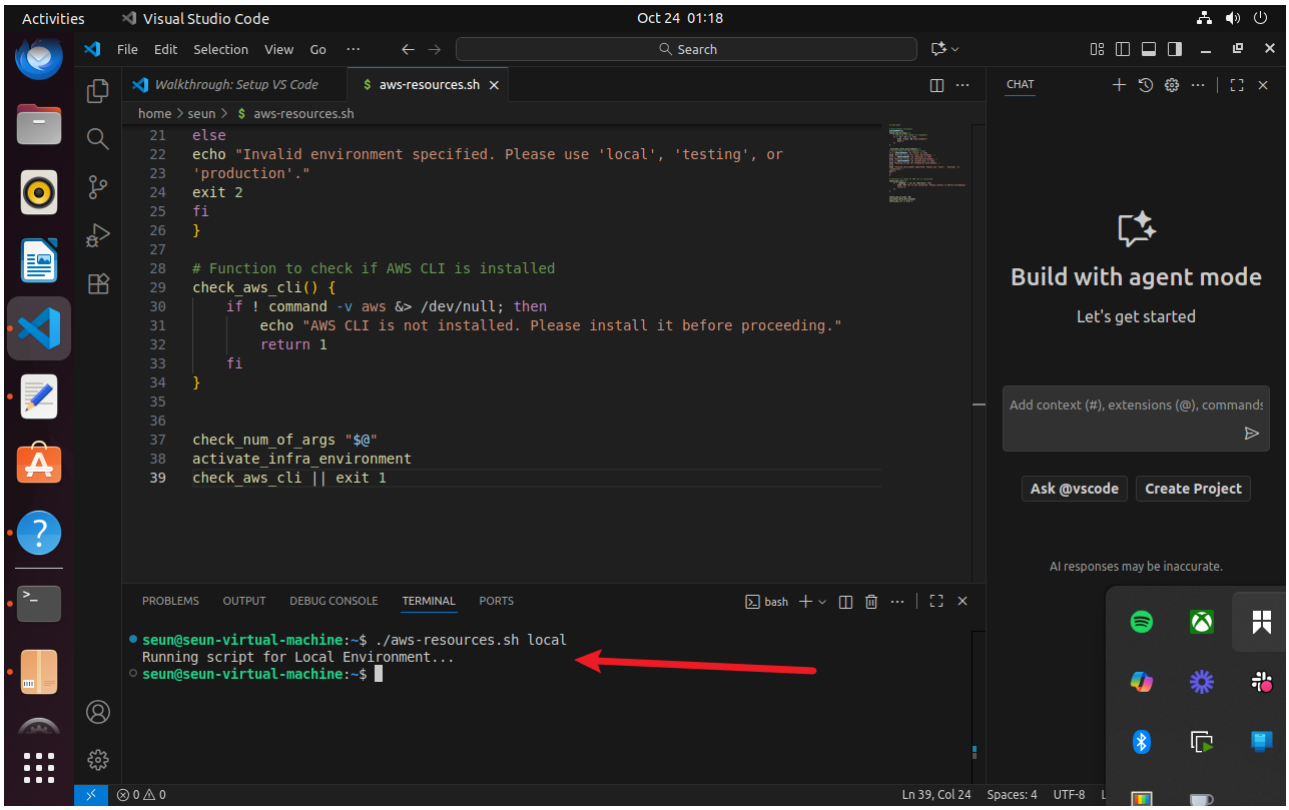
Add the function and call it:

```
# 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
```



Step 64: Run AWS CLI Check

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

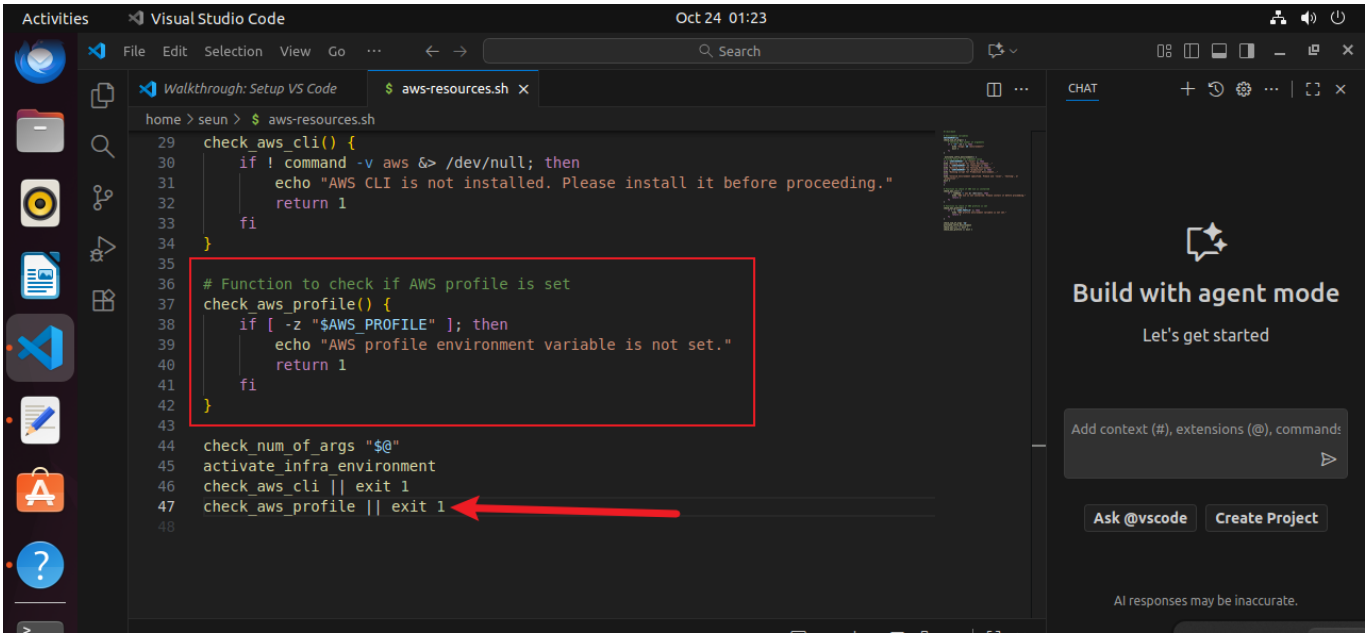


Step 65: Add AWS Profile Check Function

Add the function and call it:

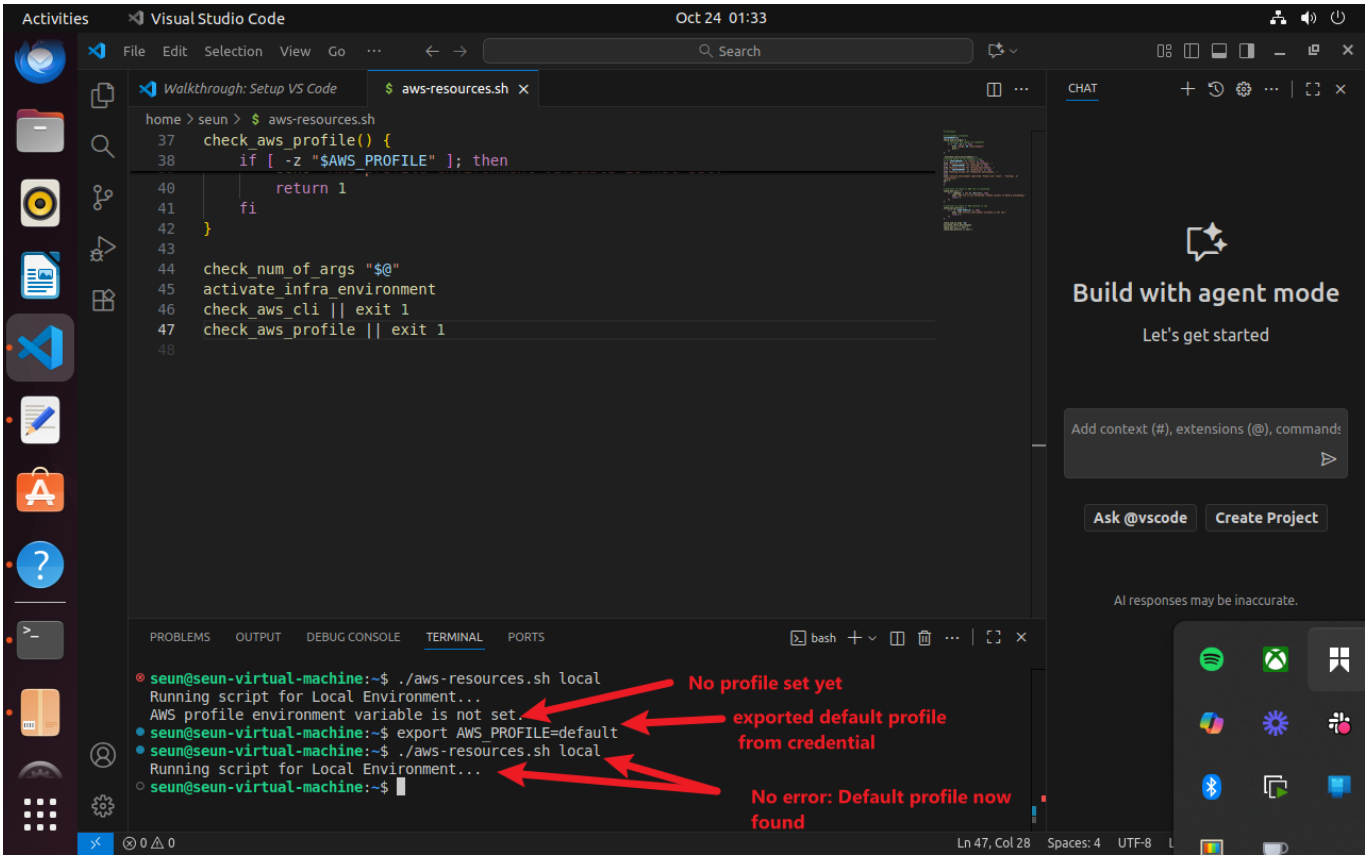
```
# Function to check if AWS profile is set
check_aws_profile() {
    if [ -z "$AWS_PROFILE" ]; then
        echo "AWS profile environment variable is not set."
        return 1
    fi
```

```
}

check_aws_profile || exit 1
```



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)

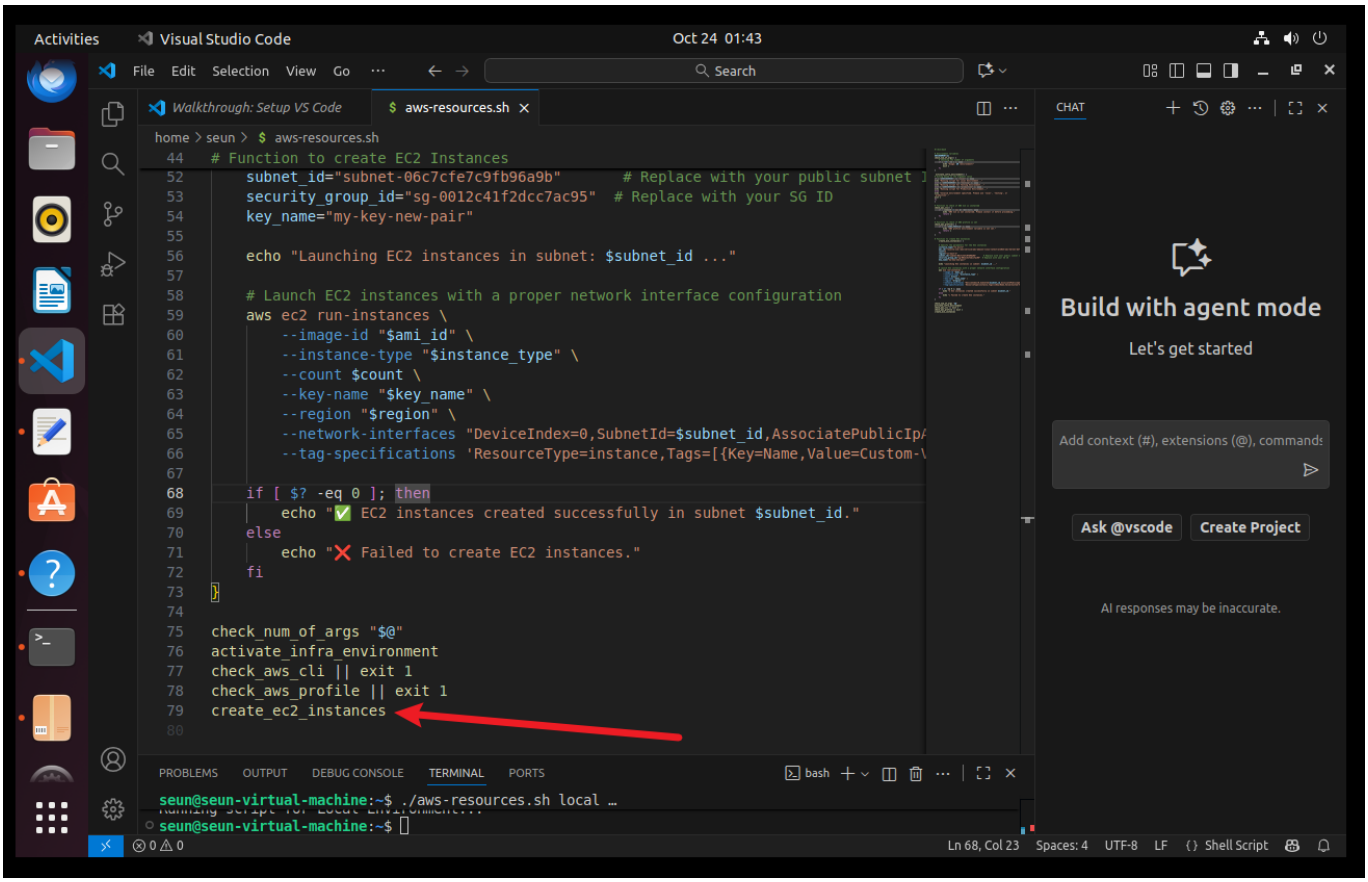


Step 67: Add EC2 Instance Creation Function

Add the function and call it:

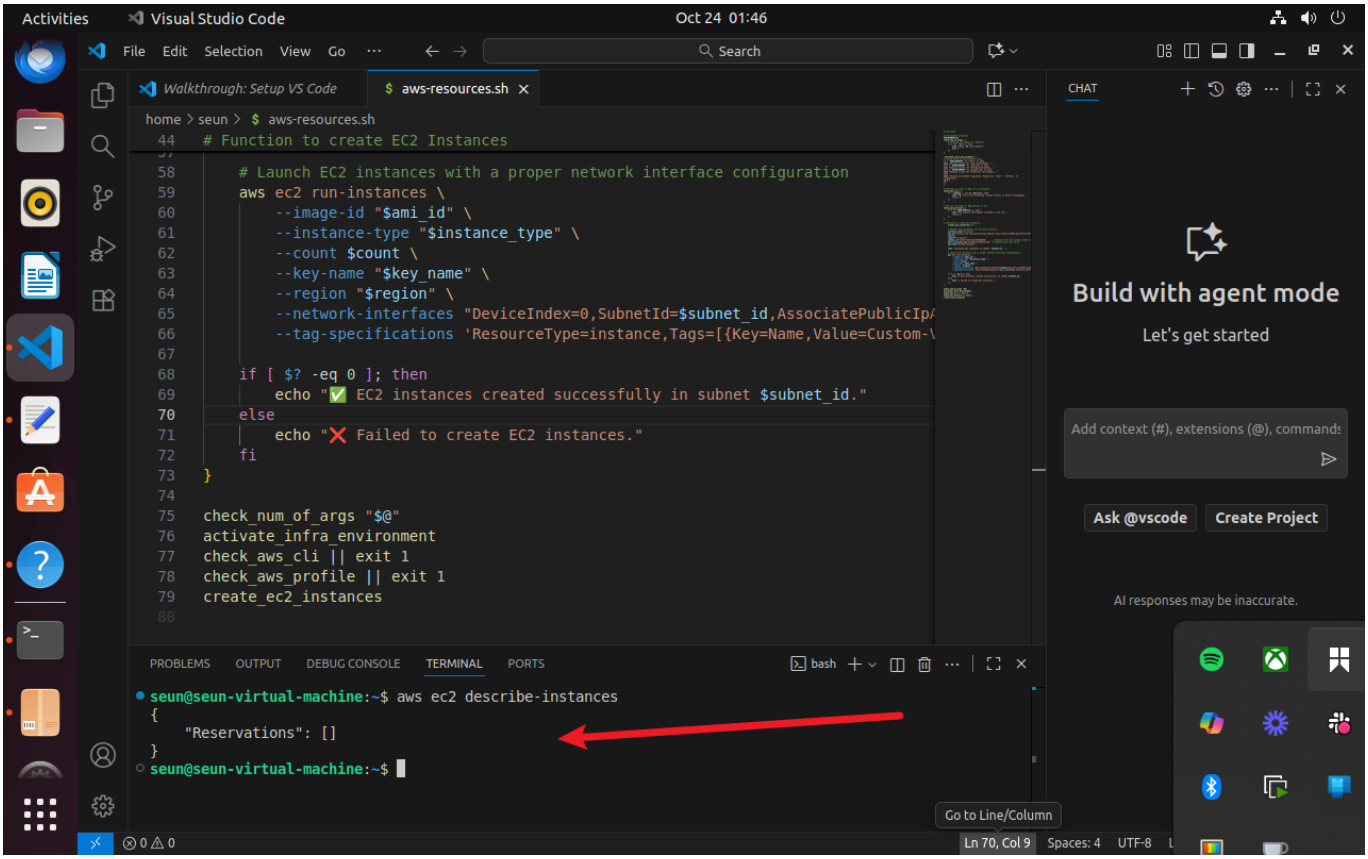
```
# Function to create EC2 Instances
    create_ec2_instances() {

    # 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.



Step 68: Verify No Existing EC2 Instances

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

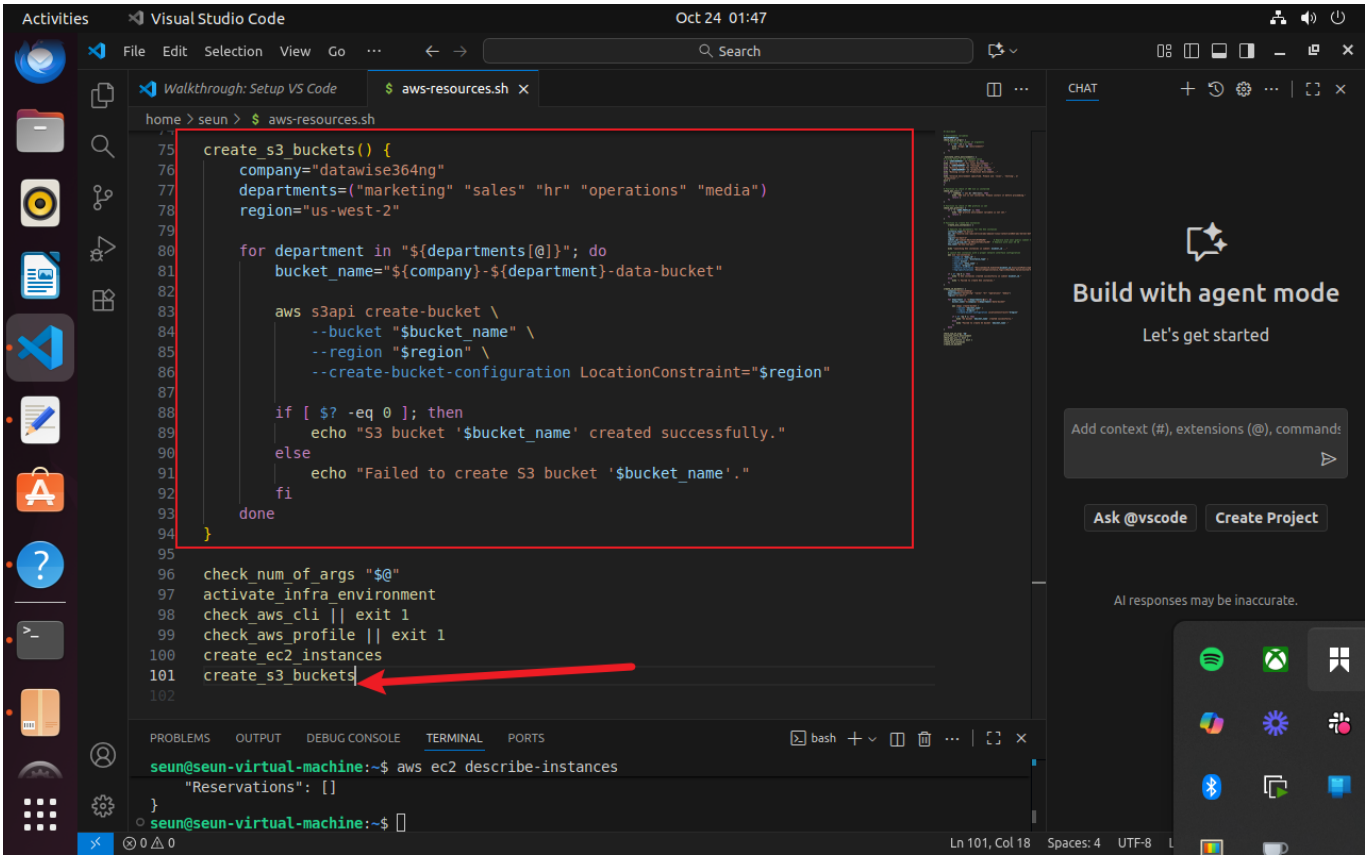


Step 69: Add S3 Bucket Creation Function

Add the function and call it:

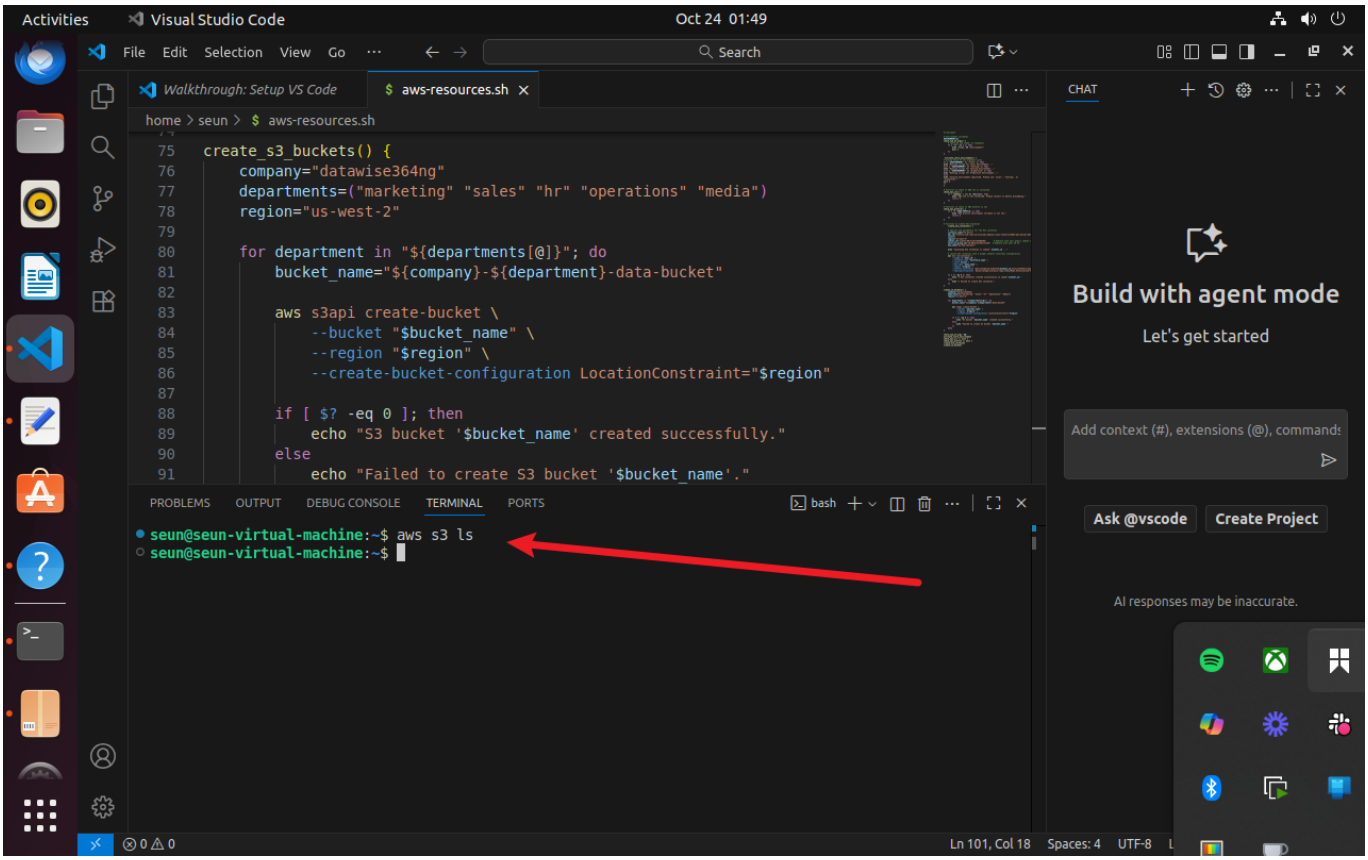
```
create_s3_buckets() {
    company="datawise364ng"
    departments=("marketing" "sales" "hr" "operations" "media")
    region="us-west-2"

    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.



Step 70: Verify No Existing S3 Buckets

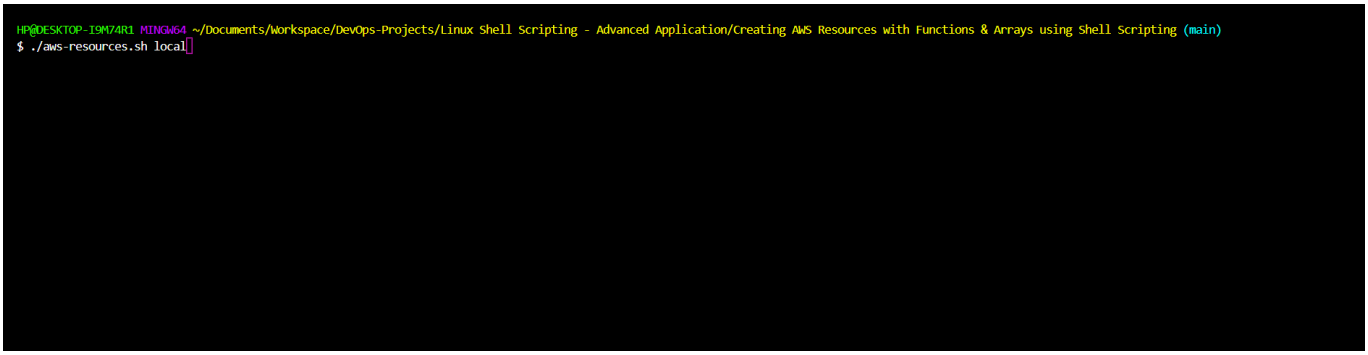
- Before execution, note there are currently no S3 buckets (check via `aws s3 ls` or console)



Phase 10: Test Script and Troubleshoot Permissions

Step 71: Run Script to Create Resources

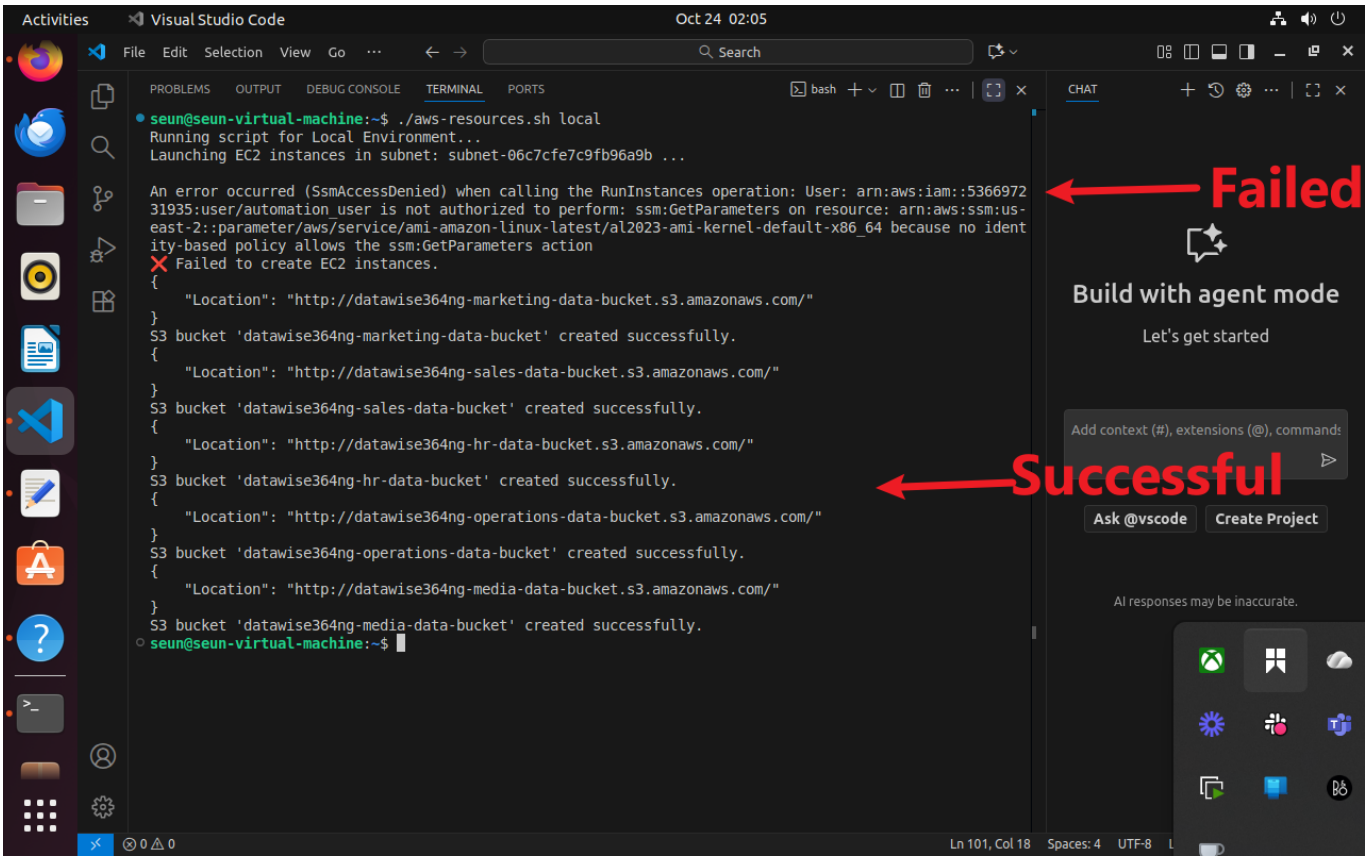
- Execute `./aws-resources.sh local`



Step 72: Initial Execution Results

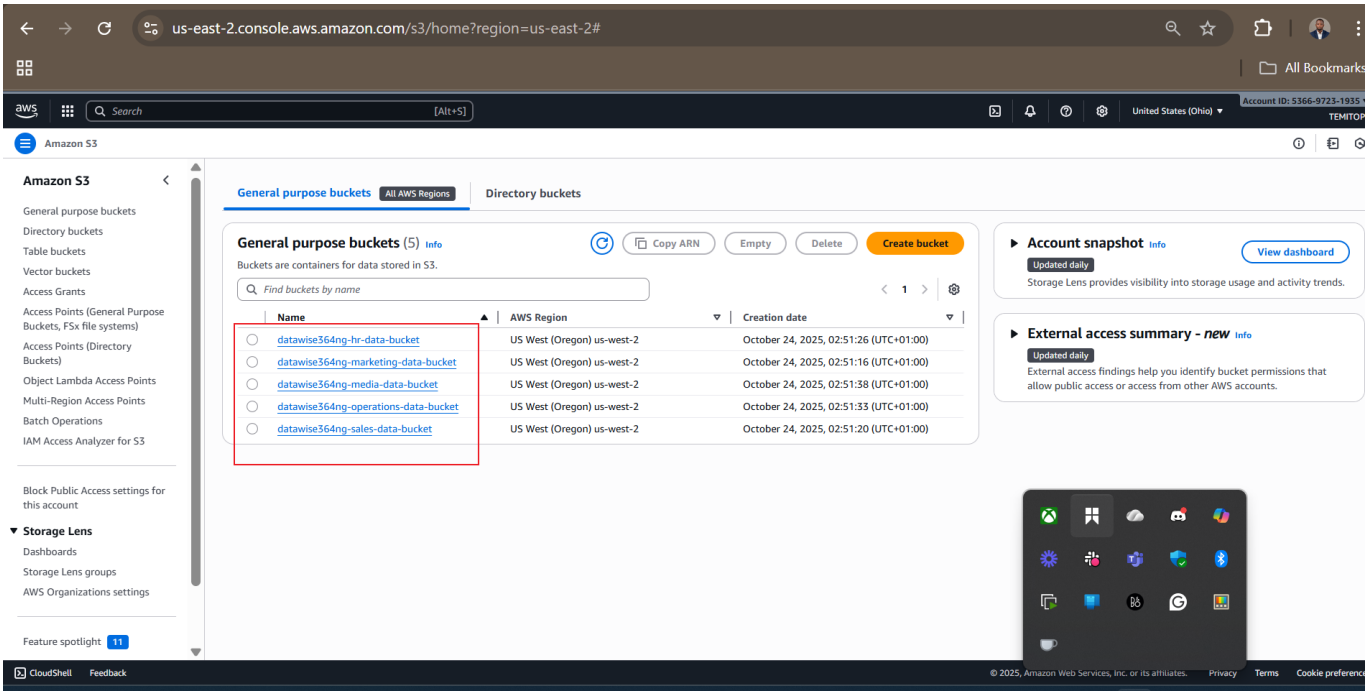
- S3 buckets created successfully
- EC2 creation denied due to SSM permissions error (e.g., access denied for ssm:GetParameter)

Purpose: Identifies missing permissions in the policy.



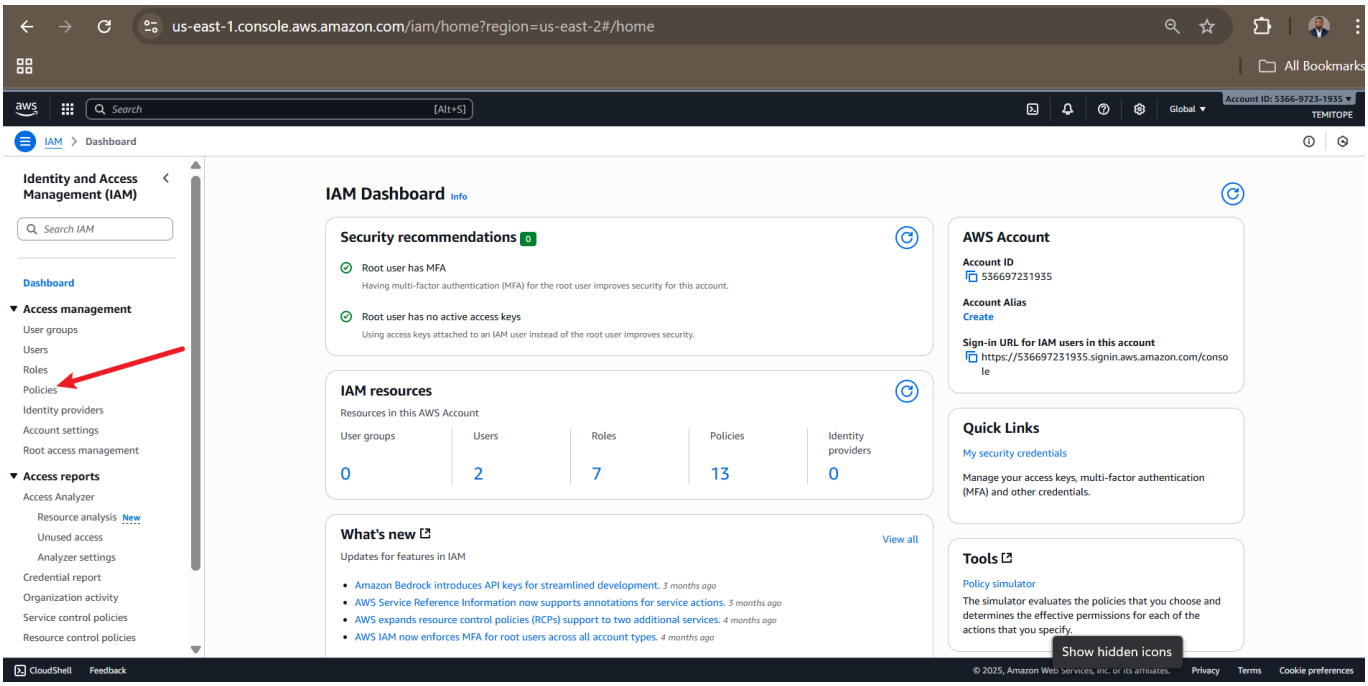
Step 73: Verify Created S3 Buckets

- Check the created buckets on the S3 console dashboard



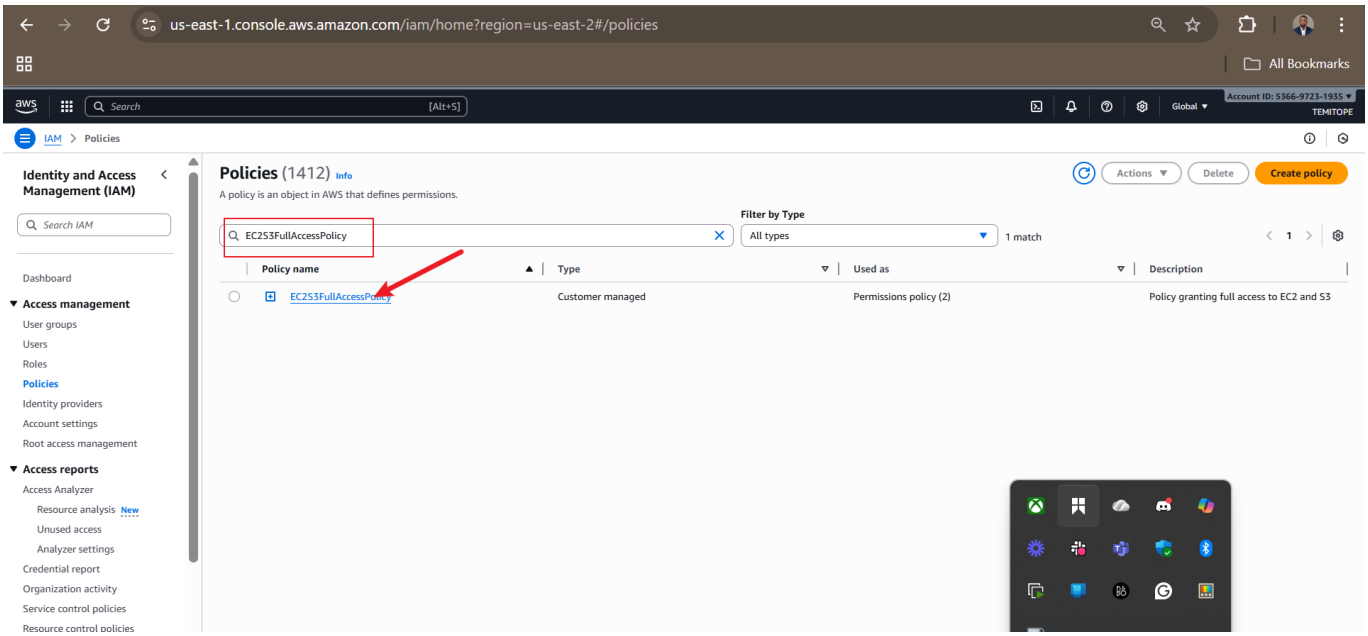
Step 74: Navigate Back to IAM Policies

- Back to IAM dashboard, click on Policies



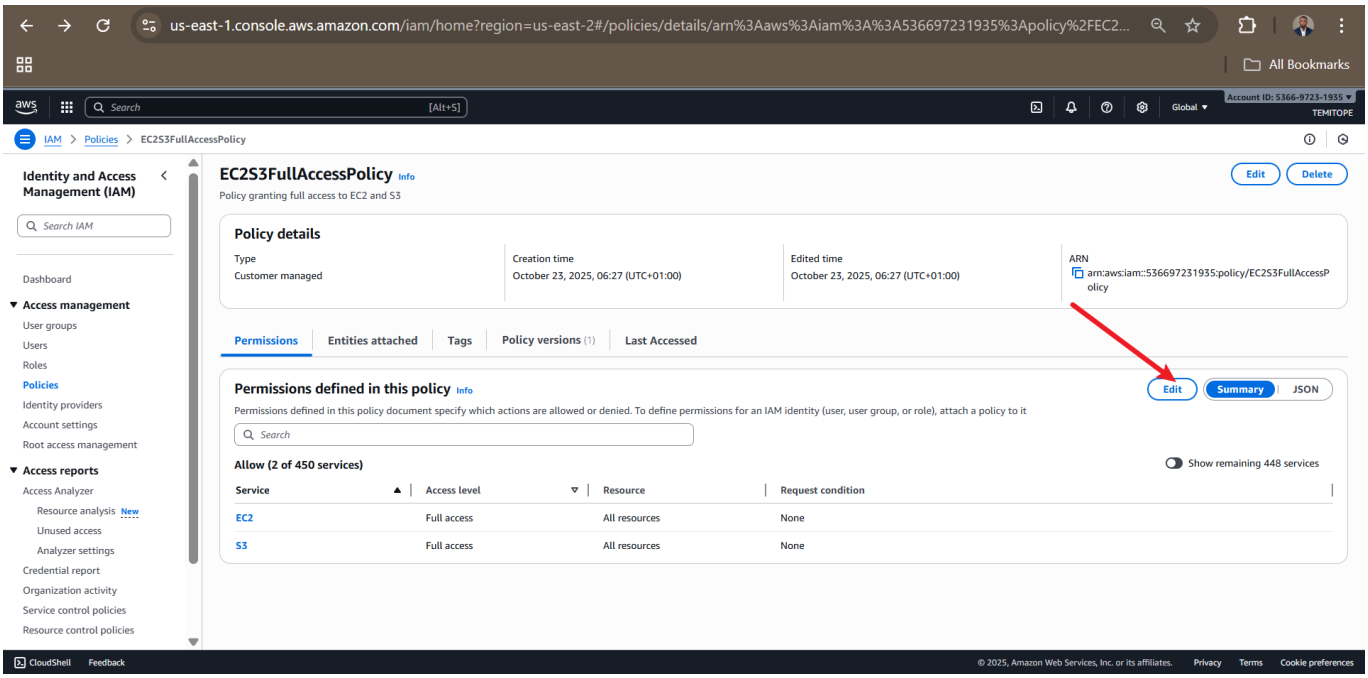
Step 75: Open Existing Policy

- Search for EC2S3FullAccessPolicy created initially and click to open it



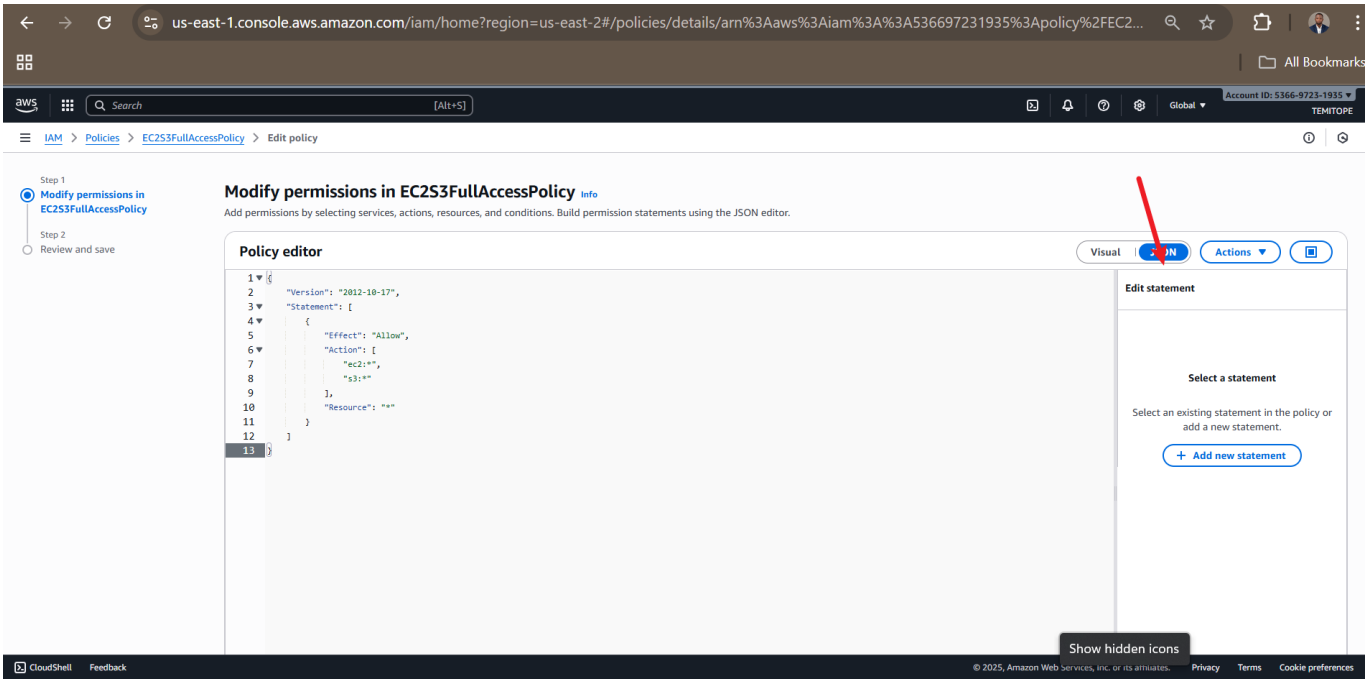
Step 76: Edit Policy

- On EC2S3FullAccessPolicy dashboard, in permissions section, click Edit



Step 77: Access JSON Editor

- Ensure JSON is selected and policy editor JSON shows



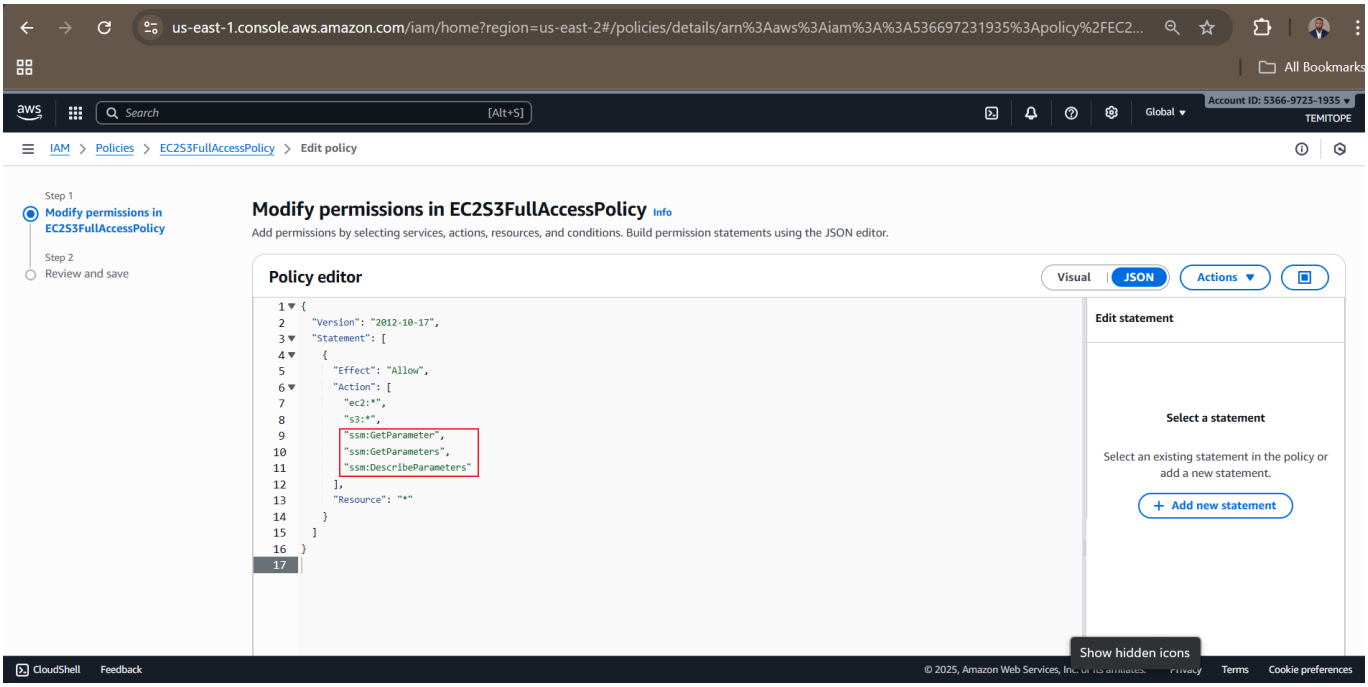
Step 78: Update JSON for SSM Permissions

Update the JSON by adding a new statement for SSM:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:*",
```

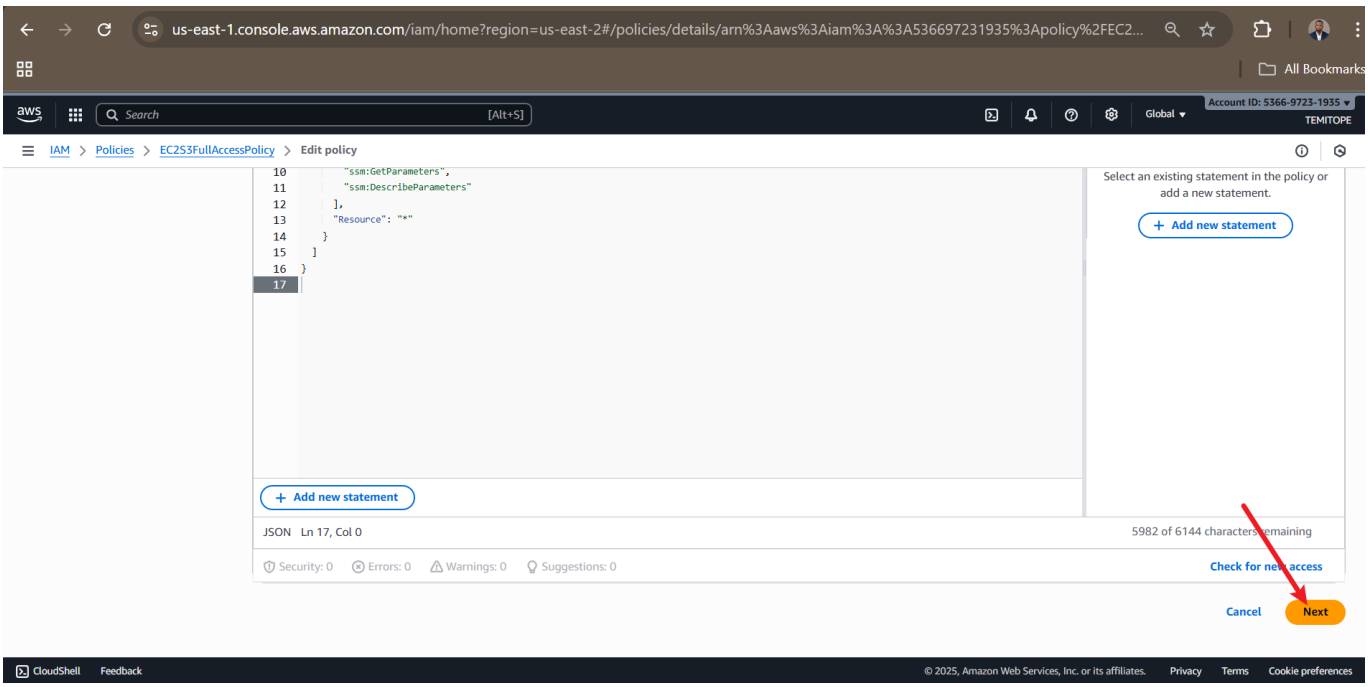
```
                "s3:*",
                "ssm:GetParameter",
                "ssm:GetParameters",
                "ssm:DescribeParameters"
            ],
            "Resource": "*"
        }
    ]
}
```

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



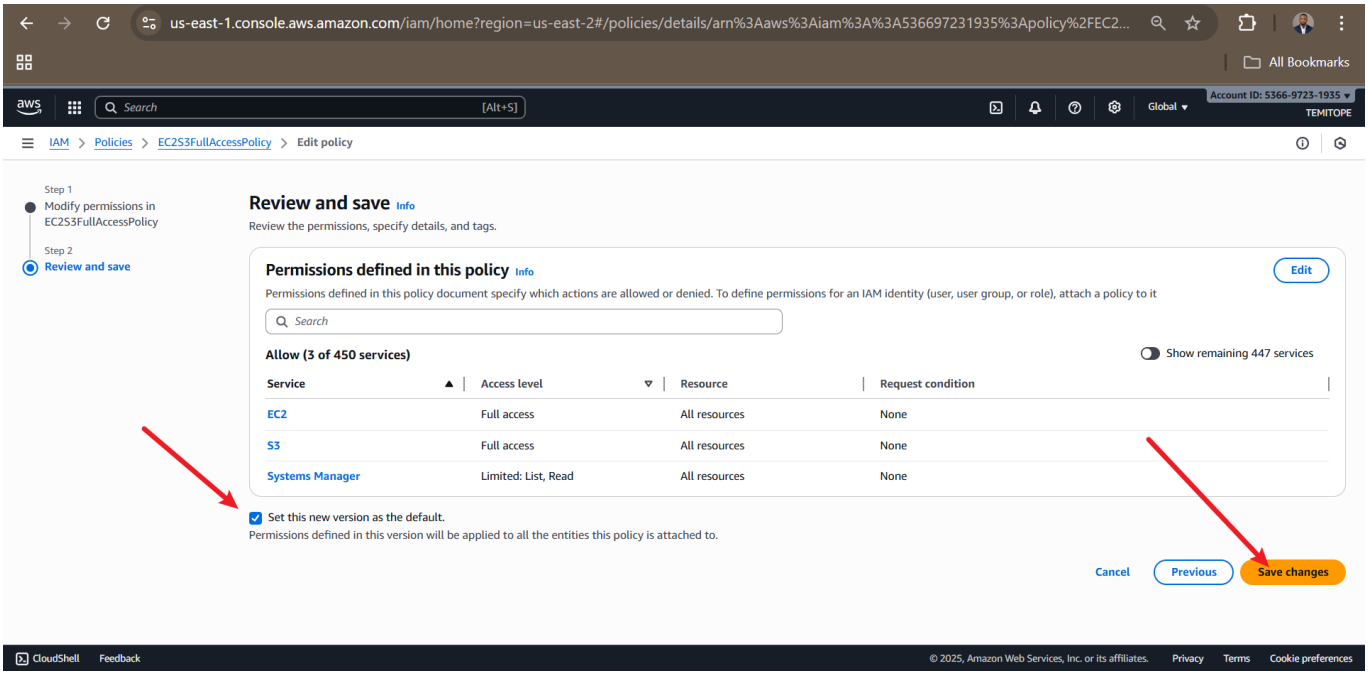
Step 79: Proceed with Edit

- Scroll down and click Next



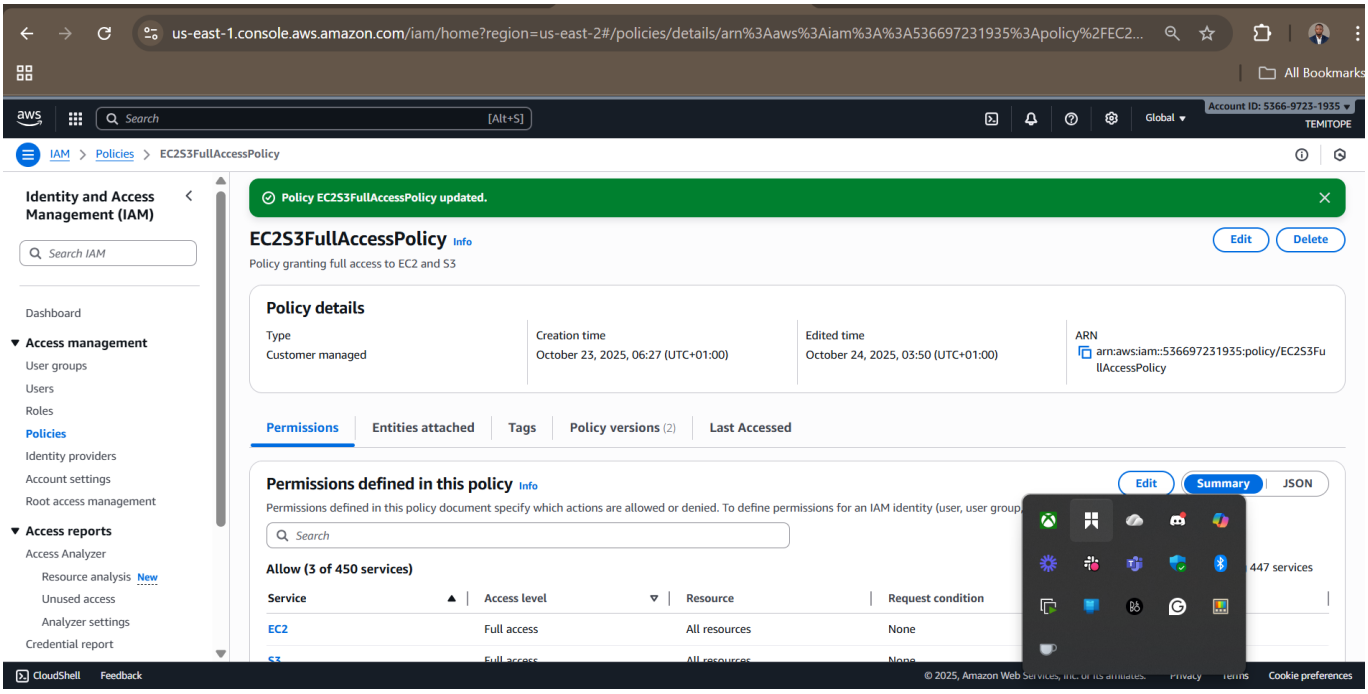
Step 80: Save Policy Changes

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



Step 81: Verify Policy Edit

- Policy edited successfully



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
```

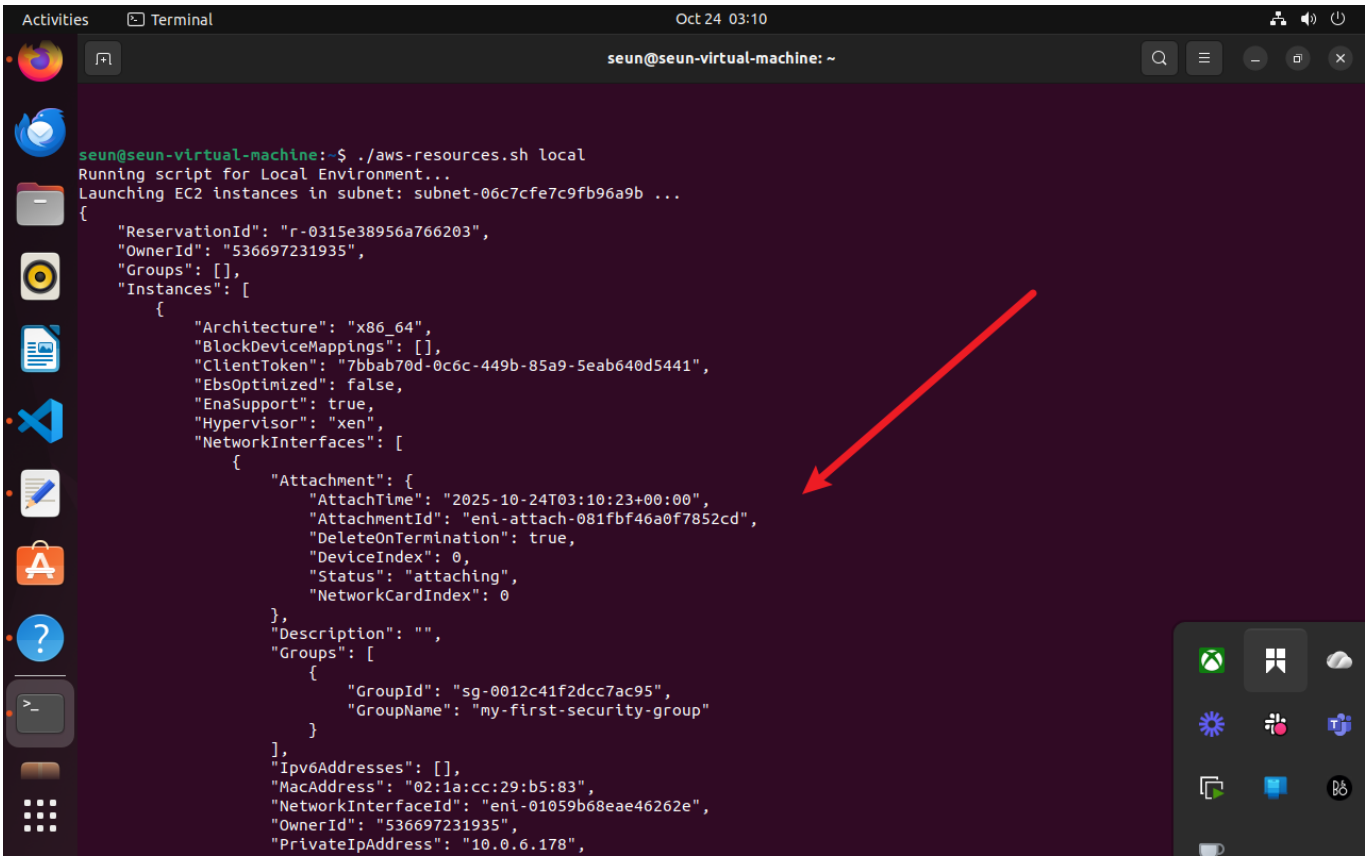
```
    echo "🧹 Deleting $bucket ..."
    aws s3 rb "s3://$bucket" --force
done
```

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

 Step 82

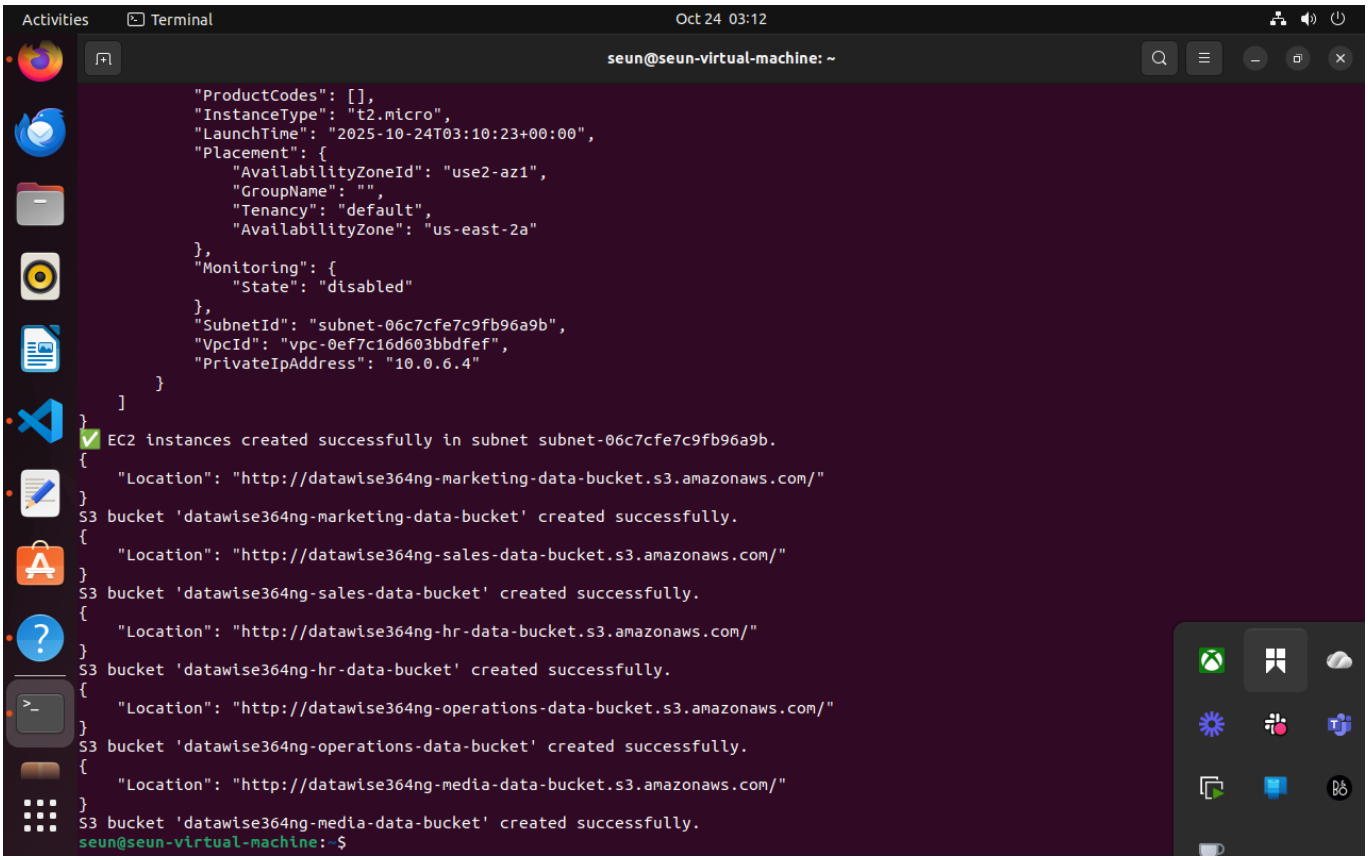
Step 83: Re-run Script for EC2

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



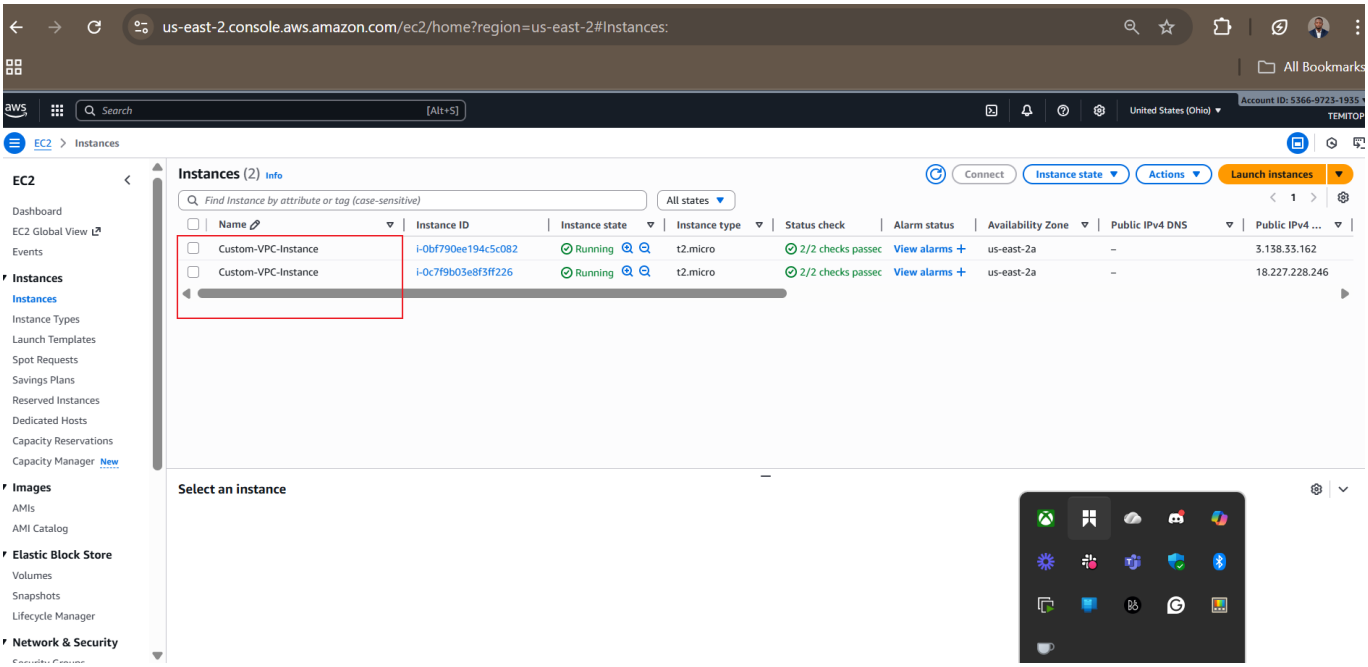
Step 84: Successful Resource Creation

- EC2 and S3 now created successfully with no error



Step 85: Verify Created EC2 Instances

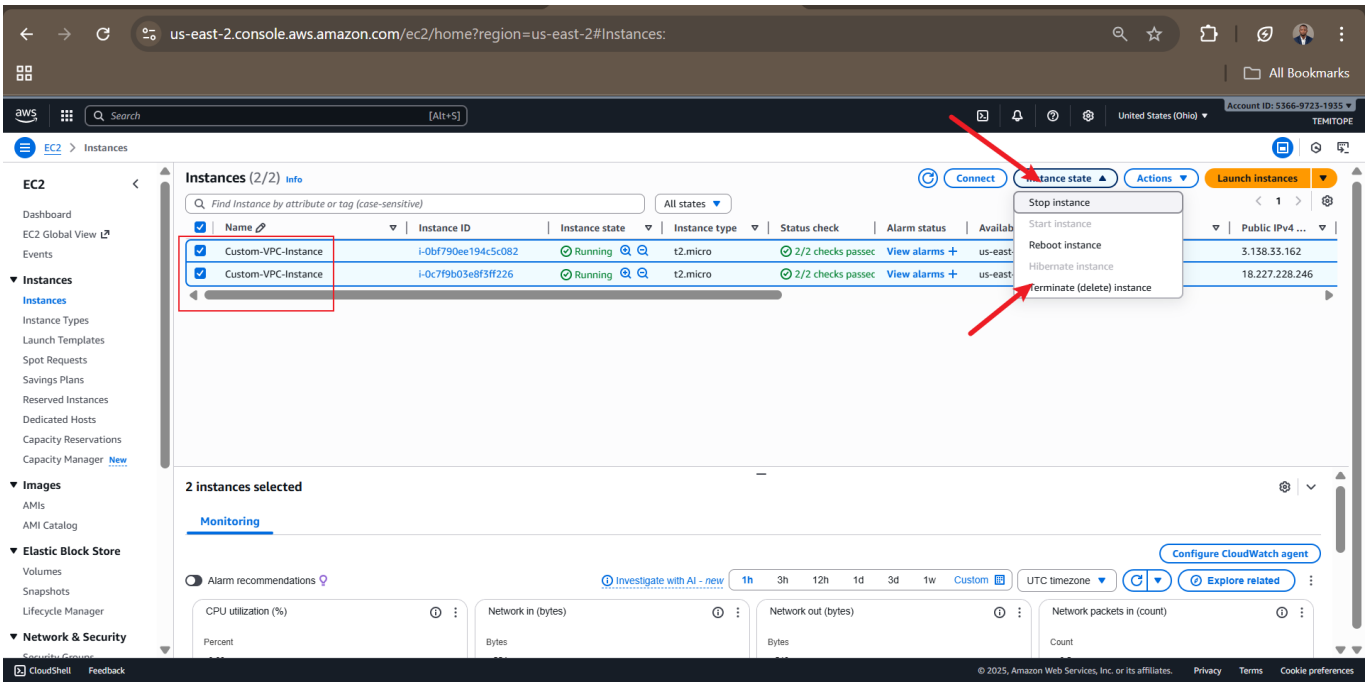
- Check created instances on EC2 instance dashboard



Phase 11: Clean Up Resources

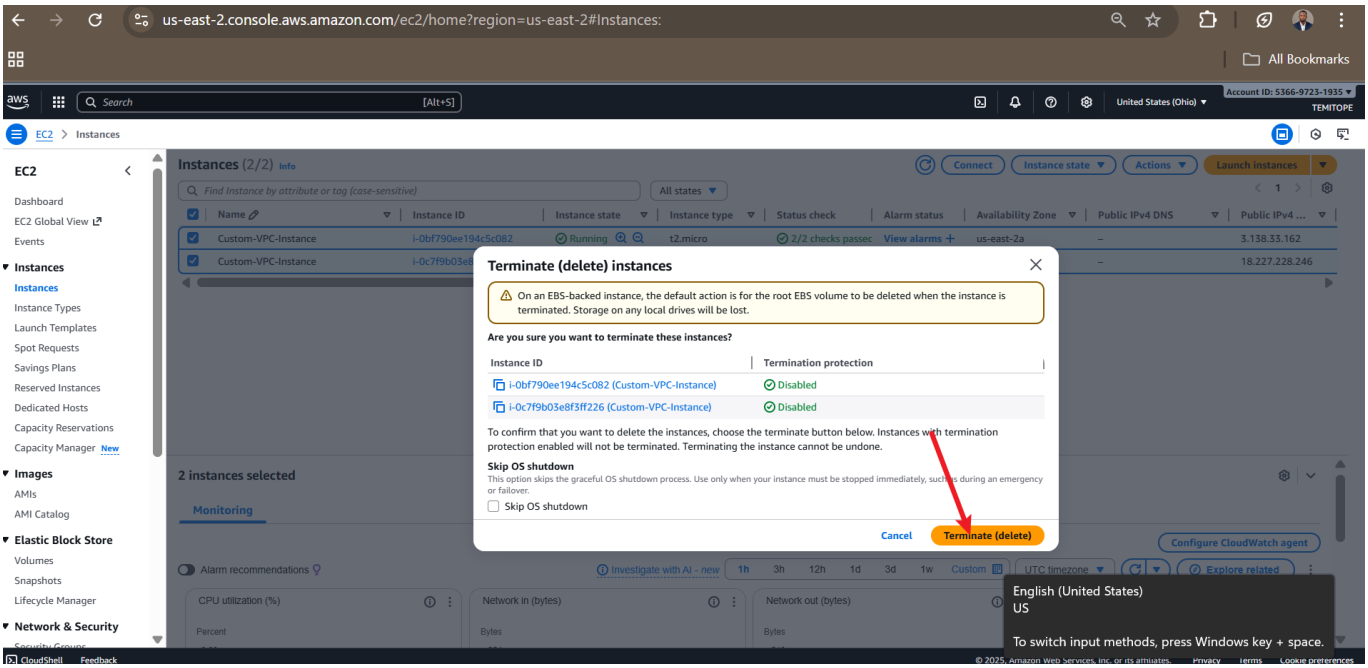
Step 86: Terminate EC2 Instances

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



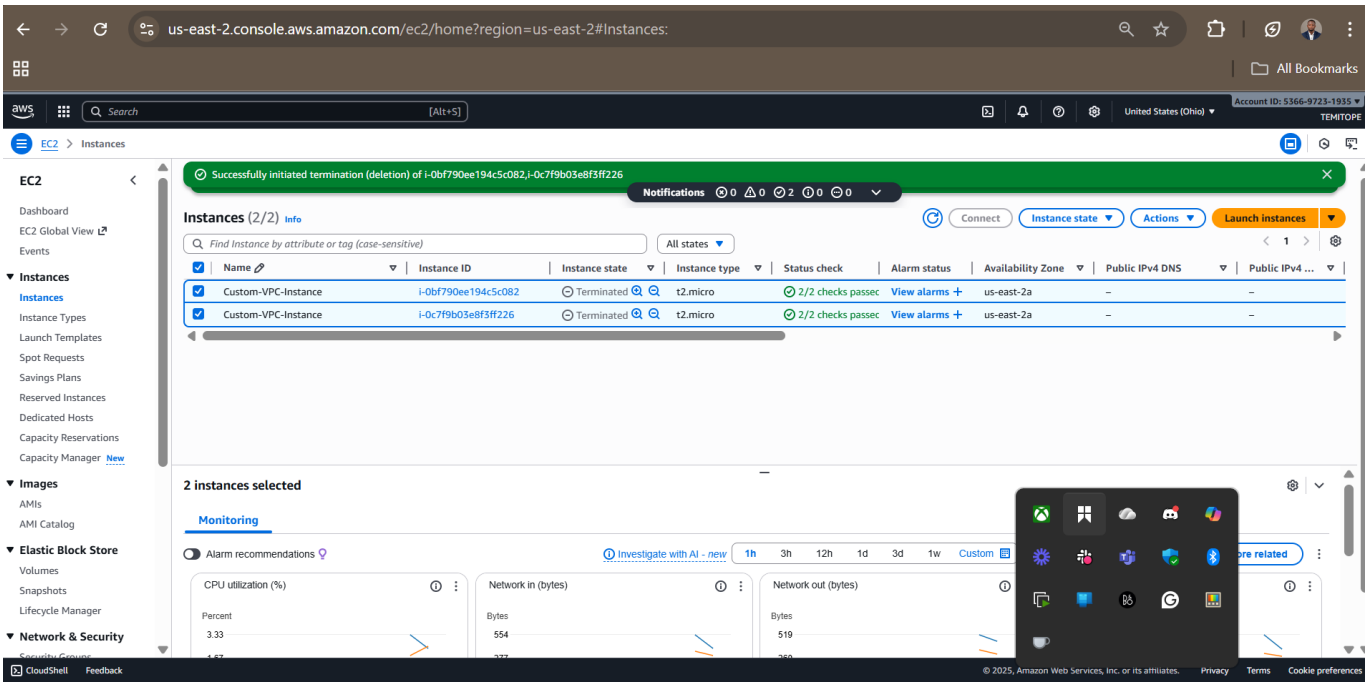
Step 87: Confirm Termination

- Click button to terminate and delete



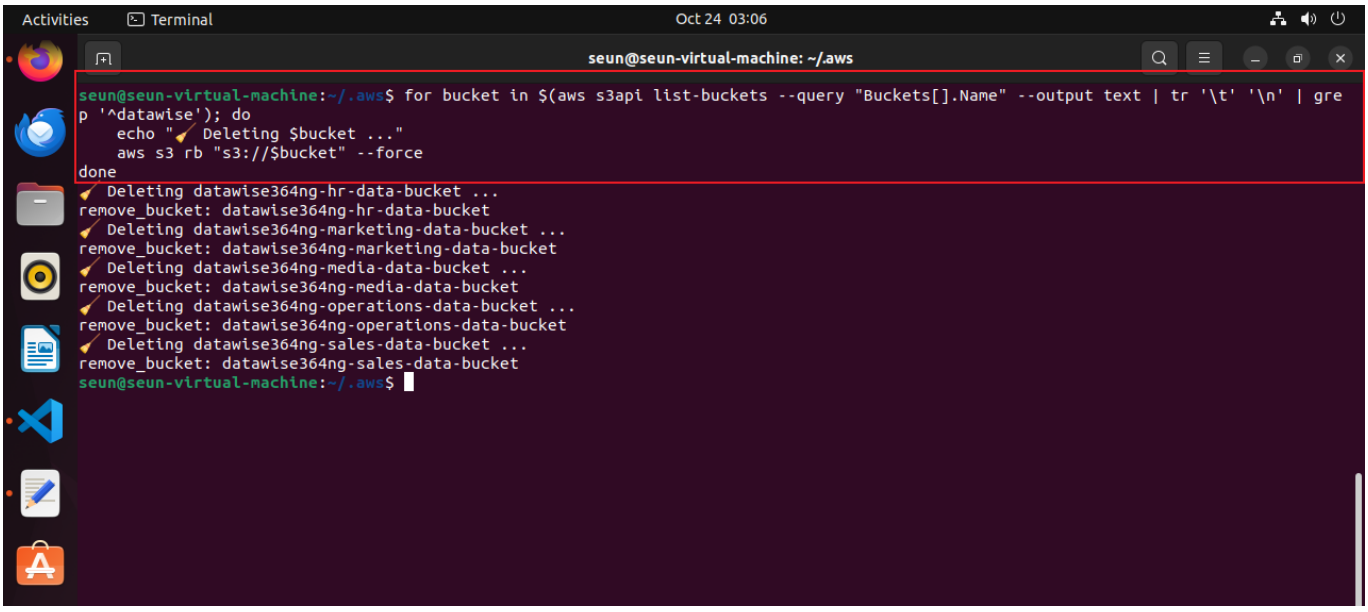
Step 88: Verify Termination

- Instances now terminated



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)



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