Activity 1: Environment Setup 🔧

Objective🎯

In this activity, we will prepare your local machine and AWS account, creating a professional development environment ready for building serverless applications. Follow these steps carefully.

⚠️ Important Note: This is an Ephemeral Activity

Please be aware that this activity is **ephemeral**.
If you exit the activity and come back later, **all previous states and progress made in your CLAB terminal will be lost**, and you will have to start over from the beginning.
If you return, you must run the command below before starting again.

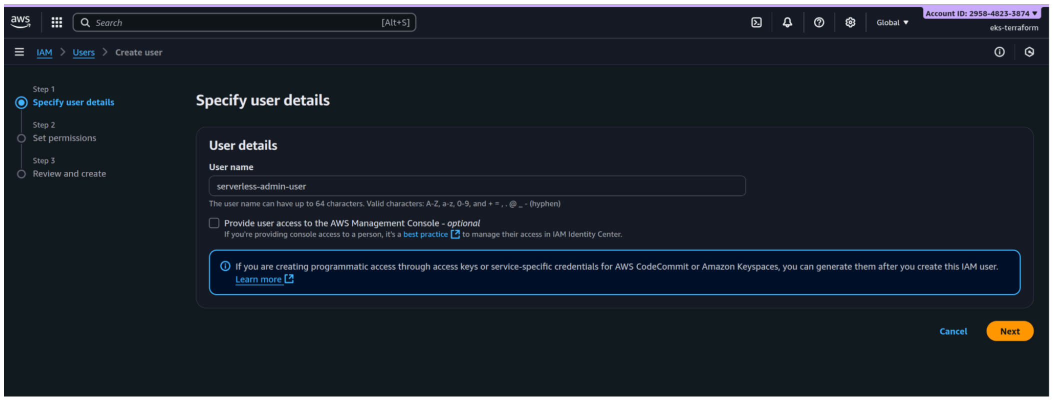
Shell
```shell
find . -maxdepth 1 ! -name 'problem_statement.txt' ! -name 'task.txt' ! -name
.git ! -name '.' -exec rm -rf {} +
```

Step 1: Create an AWS IAM User for Programmatic Access

The Serverless Framework needs programmatic permissions to create and manage resources in your AWS account. We will create a dedicated IAM user for this purpose.

1. **Navigate to the IAM Console:**
   - Sign in to your AWS Management Console.
   - In the search bar at the top, type `IAM` and select it from the services list.
2. **Create the User:**
   - In the IAM dashboard, click on **Users** in the left navigation pane, then click the **Create user** button.

- **User name:** Enter a descriptive name, like `serverless-admin-user`.
- Click **Next**.



Specify user details

User details

User name

serverless-admin-user

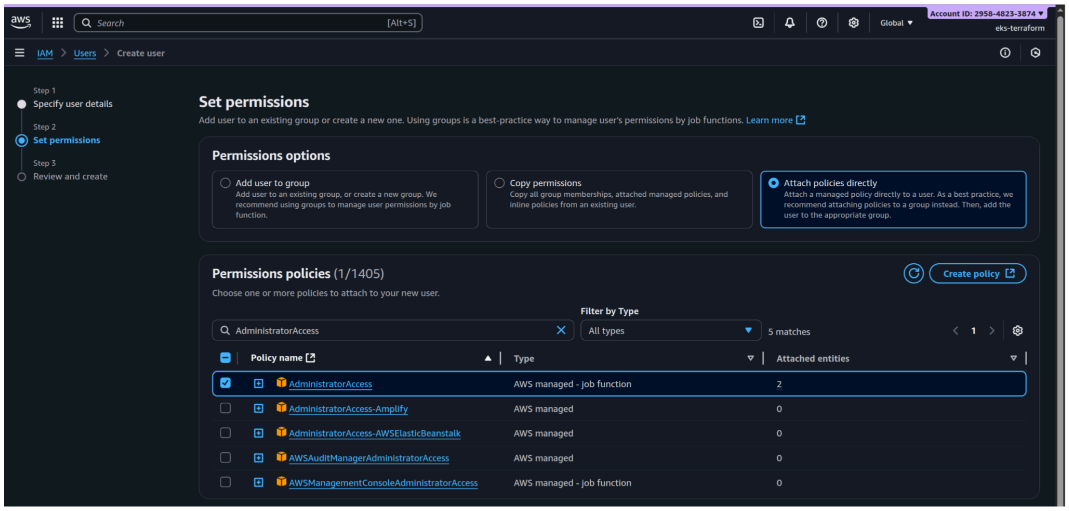
The user name can have up to 64 characters. Valid characters: A-Z, a-z, 0-9, and + = , . @ _ - (hyphen)

☐ Provide user access to the AWS Management Console - *optional*
If you're providing console access to a person, it's a best practice to manage their access in IAM Identity Center.

If you are creating programmatic access through access keys or service-specific credentials for AWS CodeCommit or Amazon Keyspaces, you can generate them after you create this IAM user.
Learn more

3. **Set Permissions:**

- Select **Attach policies directly**.
- In the search box for policies, type `AdministratorAccess`.
- Check the box next to the **AdministratorAccess** policy.
- Click **Next**.



Set permissions

Add user to an existing group or create a new one. Using groups is a best-practice way to manage user's permissions by job functions. Learn more

Permissions options

○ Add user to group
Add user to an existing group, or create a new group. We recommend using groups to manage user permissions by job function.

○ Copy permissions
Copy all group memberships, attached managed policies, and inline policies from an existing user.

● Attach policies directly
Attach a managed policy directly to a user. As a best practice, we recommend attaching policies to a group instead. Then, add the user to the appropriate group.

Permissions policies (1/1405)

Choose one or more policies to attach to your new user.

Filter by Type

AdministratorAccess | All types | 5 matches

	Policy name	Type	Attached entities
✓	AdministratorAccess	AWS managed - job function	2
☐	AdministratorAccess-Amplify	AWS managed	0
☐	AdministratorAccess-AWSElasticBeanstalk	AWS managed	0
☐	AWSAuditManagerAdministratorAccess	AWS managed	0
☐	AWSManagementConsoleAdministratorAccess	AWS managed - job function	0

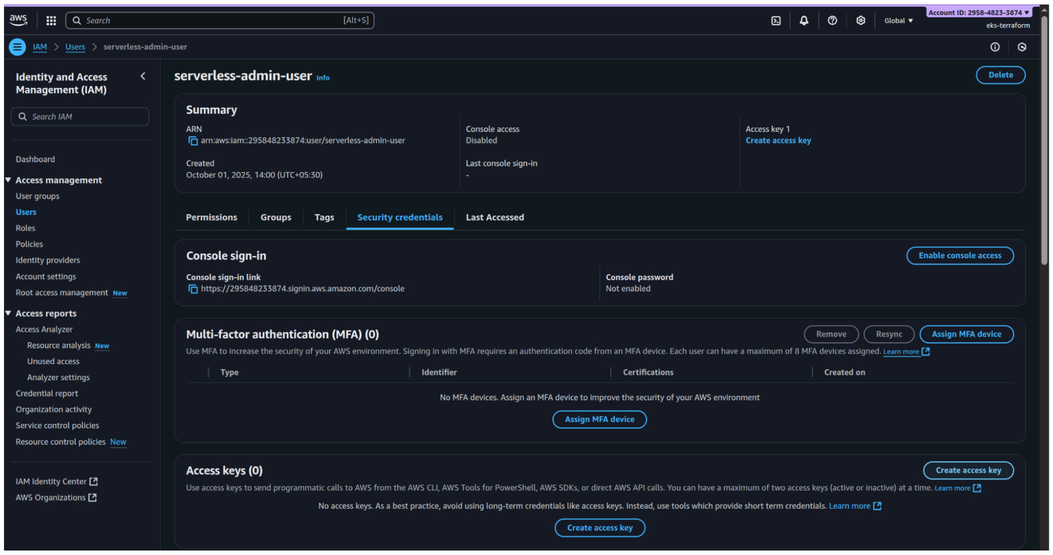
Note: For this educational lab, `AdministratorAccess` provides the simplest path to get started. In a real-world production environment, you should always follow the principle of least privilege and create a role with more restrictive, fine-grained permissions.

4. **Review and Create:**

- Review the user details and click **Create user**.

5. **Retrieve Your Access Keys:**

   - After the user is created, click on the user's name in the list.
   - Go to the **Security credentials** tab.
   - Scroll down to the **Access keys** section and click **Create access key**.
   - Select **Command Line Interface (CLI)** as the use case.
   - Acknowledge the recommendation and click **Next**.
   - (Optional) Set a description tag, like `Serverless Framework Lab Key`.
   - Click **Create access key**.
   - **This is the only time you will see the Secret access key.** Copy both the **Access key ID** and the **Secret access key** and save them somewhere secure on your machine. We will need them in the next step.



serverless-admin-user

Summary

ARN
arn:aws:iam::295848233874:user/serverless-admin-user

Created
October 01, 2025, 14:00 (UTC+05:30)

Console access
Disabled

Last console sign-in
-

Access key 1
Create access key

Console sign-in

Console sign-in link
https://295848233874.signin.aws.amazon.com/console

Console password
Not enabled

Multi-factor authentication (MFA) (0)

Use MFA to increase the security of your AWS environment. Signing in with MFA requires an authentication code from an MFA device. Each user can have a maximum of 8 MFA devices assigned.

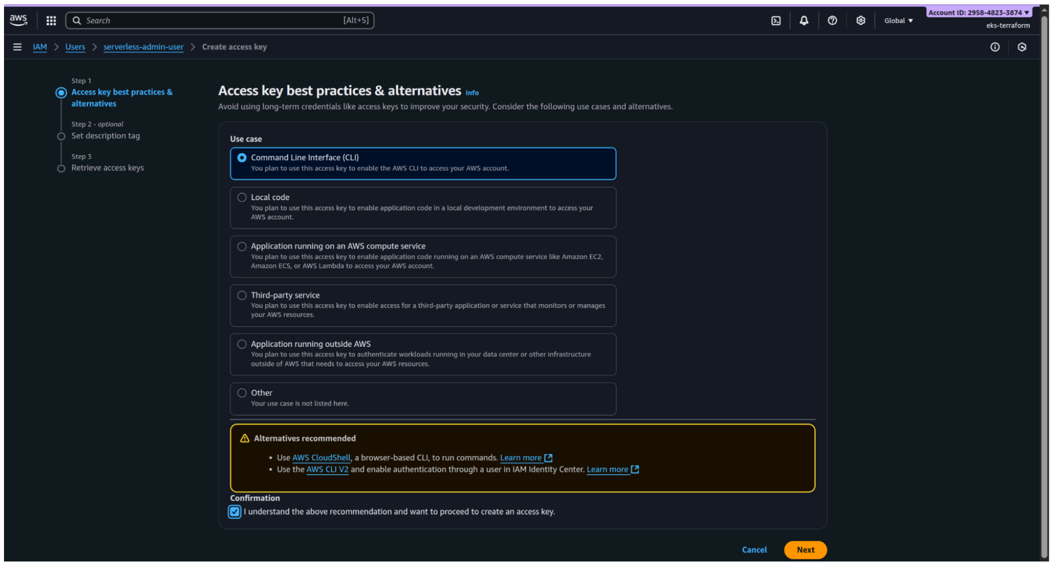
Type	Identifier	Certifications	Created on

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

Access keys (0)

Use access keys to send programmatic calls to AWS from the AWS CLI, AWS Tools for PowerShell, AWS SDKs, or direct AWS API calls. You can have a maximum of two access keys (active or inactive) at a time.

No access keys. As a best practice, avoid using long-term credentials like access keys. Instead, use tools which provide short term credentials.


Search [Alt+S]
Global
Account ID: 2958-4823-3874
eks-terraform
IAM > Users > serverless-admin-user > Create access key
Step 1
Access key best practices & alternatives
Step 2 - optional
Set description tag
Step 3
Retrieve access keys
Cancel
Next

Access key best practices & alternatives Info

Avoid using long-term credentials like access keys to improve your security. Consider the following use cases and alternatives.

Use case

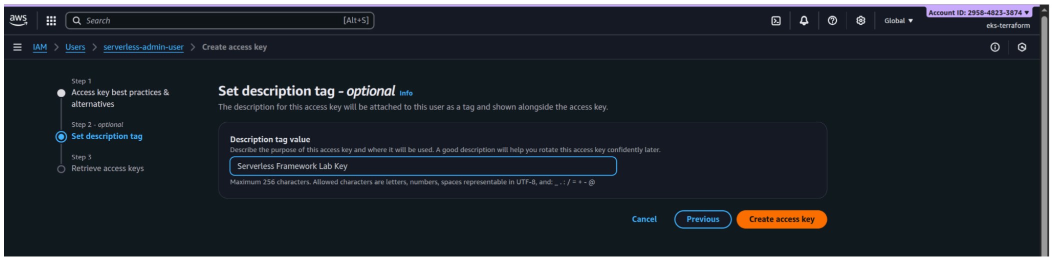
- [✓ Command Line Interface (CLI)] You plan to use this access key to enable the AWS CLI to access your AWS account.
- Local code: You plan to use this access key to enable application code in a local development environment to access your AWS account.
- Application running on an AWS compute service: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

Alternatives recommended

- Use AWS CloudShell, a browser-based CLI, to run commands. Learn more
- Use the AWS CLI V2 and enable authentication through a user in IAM Identity Center. Learn more

Confirmation

✓ I understand the above recommendation and want to proceed to create an access key.



Set description tag - *optional* Info

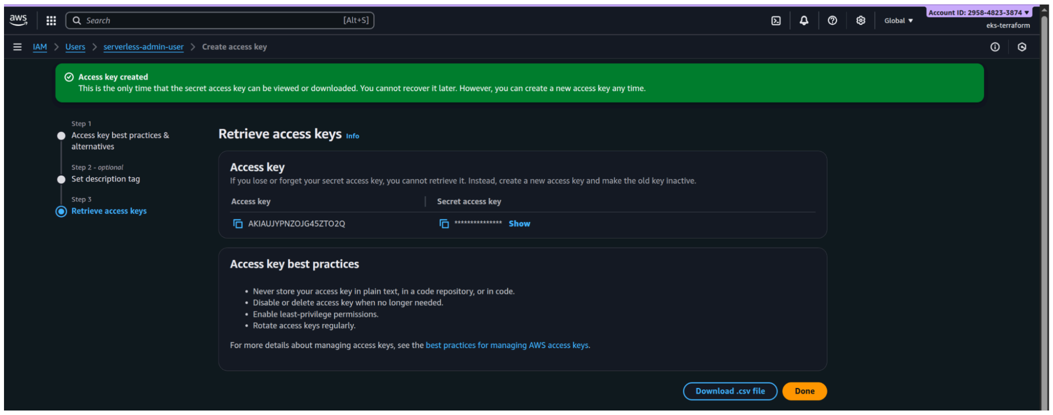
The description for this access key will be attached to this user as a tag and shown alongside the access key.

Description tag value

Describe the purpose of this access key and where it will be used. A good description will help you rotate this access key confidently later.

Serverless Framework Lab Key

Maximum 256 characters. Allowed characters are letters, numbers, spaces representable in UTF-8, and: _ . : / = + - @



Access key created

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

Retrieve access keys Info

Access key

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

Access key	Secret access key
AKIAUJYPNZOJG45ZTO2Q	*************** Show

Access key best practices

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

For more details about managing access keys, see the best practices for managing AWS access keys.

Step 2: Install and Configure the AWS CLI

The AWS Command Line Interface (CLI) is a tool that allows you to interact with AWS services from your terminal.

1. **Install the AWS CLI:**

- Open CLAB terminal.

- Check if AWS CLI already installed with `aws --version`.
- If you don't have it installed, follow the official instructions for your operating system: Installing the AWS CLI version 2.

2. **Configure a CLI Profile:**

- Open your terminal or command prompt.
- We will create a named profile to keep our lab credentials separate. This is a best practice for managing multiple projects or accounts.
- Run the following command:

```
Shell
aws configure --profile serverless-lab
```

- The CLI will now prompt you for four pieces of information. Use the credentials you saved in the previous step.
    - **AWS Access Key ID:** Paste the Access key ID.
    - **AWS Secret Access Key:** Paste the Secret access key.
    - **Default region name:** Enter a region to work in, for example, `ap-south-1`.
    - **Default output format:** You can leave this blank or type `json`.

  Your terminal should look something like this:

```
None
$ aws configure --profile serverless-lab
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]: ap-south-1
Default output format [None]: json

# After setup verify
$ aws configure list --profile serverless-lab
```

Note: Aws saves all the configs in `~/.aws/config` and `~/.aws/credentials` files.

Step 3: Install Python

Our guided project will use Python for the AWS Lambda functions, so you'll need a recent version of Python installed on your machine.

1. **Download and Install Python:**

    - Navigate to the official Python downloads page: python.org/downloads/.
    - Download and run the installer for a version of Python 3.9 or higher, as these are well-supported by AWS Lambda.
    - Or follow below instructions:

```
Shell
apt install -y python3 python3-pip python3-venv
```

2. **Best Practice: Python Virtual Environments:**

    - It is highly recommended to use a virtual environment for each of your Python projects to manage dependencies. While not strictly required for this lab to function, it's a critical skill. You can create one for our lab project later, but it's good to know the commands now:

```
Shell
# Create a virtual environment named 'venv' in your project folder
python3 -m venv venv

# Activate the virtual environment
source venv/bin/activate
```

Step 4: Install Node.js and `nvm`

The Serverless Framework is a **Node.js** application, so we need Node.js installed. We'll use `nvm` (Node Version Manager) to install it, as this tool makes it easy to manage different Node.js versions.

1. **Install nvm:**
    - Open a new CLAB terminal.
    - Check nvn is already installed with `nvm -v`
    - Run the installation script from the official `nvm` repository. The exact command can be found here: nvm GitHub Repository. Typically, it's a `curl` or `wget` command.
    - After running the script, close and reopen your terminal.
2. **Install and Use Node.js:**
    - Now, use `nvm` to install the latest Long-Term Support (LTS) version of Node.js:

```Shell
nvm install --lts
```

- Tell `nvm` to use this version in your current shell:

```Shell
nvm use --lts
```

Step 5: Install the Serverless Framework

With Node.js and its package manager (`npm`) installed, we can now install the Serverless Framework.

1. **Install via npm:**

- In your terminal, run the following command to install the framework:

```shell
npm install -g serverless
```

Step 6: Connect to the Serverless Dashboard

The Serverless Dashboard provides a web interface to monitor, manage, and gain insights into your deployed services.

1. **Log in from the CLI:**
    - Run the login command in your terminal:

```shell
serverless login
```

    - This will automatically open a new tab in your web browser.
    - Choose to register or log in. It's recommended to sign up with GitHub for ease.
    - Once you've authenticated, you can return to your terminal.

✅ Step 7: Verification

Let's quickly verify that everything is installed and configured correctly.

1. **Check AWS CLI Configuration:**

```shell
aws sts get-caller-identity --profile serverless-lab
```

This command should return the `UserId`, `Account`, and `Arn` of the IAM user you created, confirming your credentials are correct.

UserId → The unique identifier for the IAM user or assumed role.
Account → The AWS account ID you're operating under.
Arn → The Amazon Resource Name of the caller (could be a user, assumed role, etc.).
STS → AWS Security Token Service. It's an AWS service that issues temporary, limited-privilege credentials for IAM users
get-caller-identity → a special STS API call that simply returns details about *who you are authenticated as* when making the call.

2. **Check Python Version:**

```shell
# On macOS, Linux, or WSL
python3 --version
```

This should show a version of 3.9 or higher.

3. **Check Node.js and npm Versions:**

```shell
node -v
npm -v
```

This should output the versions of Node.js and npm.

4. **Check Serverless Framework Version:**

```shell
serverless --version
```

This will display the framework version and confirm it's installed correctly.

Your environment is now fully configured and ready for building!

═══ END OF DOCUMENT ═══