# [DEMO] Configuring AWS access credentials for CloudGoat

Christophe • September 4, 2023

For this course, I highly recommend that you create a separate AWS account. Either that or make sure that the account you plan on using HAS NO PRODUCTION RESOURCES!

This is extremely important, because the actions we are going to take in this course will not only be destructive, they're also going to create vulnerabilities in your AWS account. You do not want that running alongside any production resources.

Take the time to pause here and create a separate AWS account. Or if you've never created an account before, then you'll want to do that now.

You can follow the instructions from either of these pages to do that and it's pretty straight forward so I won't be showing it for the sake of time:

- https://docs.aws.amazon.com/accounts/latest/reference/manage-acct-creating.html
- https://repost.aws/knowledge-center/create-and-activate-aws-account

Once you have your new AWS account, we're ready to get started with this lesson.

The first step we want to take is to create a new IAM user. This user is going to be the user that creates the lab environment and resources for us. As a general best practice, we should never use the root user after we've created our account except to create other users that we will use for day-to-day administration. If you need more information on that, please refer to our Introduction to AWS Security course.

## Steps to creating an IAM user, policies, access keys

With that, pull up IAM from the AWS console and go to `Users`. We'll create a new user so that we can give them specific permissions.

Name this user whatever you'd like:

```
cloudgoat-admin
```

We do not need to provide console access so leave that unchecked.

Normally I'd recommend using groups to provide permissions but this is a throwaway example so we'll skip that step to save time.

`Create User`

We now need to provide permissions.

This is the policy that we'll use specifically for CloudGoat, which needs to be able to provision resources in your AWS account, and so we need to give admin privileges.

Go ahead and select the `AdministratorAccess` policy.

Select it and apply it to the user, and then create the user.

Now that our example user has this policy attached, we need to generate `Security Credentials` by going to the Users page, clicking on our new user, and going to the Security Credentials tab.

From there, scroll down until you see `Access Keys` and `Create access key`.

For this video, our use case will be to access the AWS API via the CLI, so we can select `Command Line Interface (CLI)`.

You'll get a warning asking you instead to use the `AWS CloudShell` or `AWS CLI V2`, but go ahead and click the checkbox confirming you understand and click on `Next`.

We don't need to set a description, so click on `Create access key`.

We now have our AWS access keys, so we need to copy them and configure them on our machine.

## Using the AWS CLI

My machine, by default, is using the AWS CLI v1, but there is an AWS CLI v2, which is recommended to use. I believe that v1 is no longer receiving updates, so you *should* upgrade. With that said, I think a lot of you are still running v1 and instead of spending time or forcing you to upgrade, I'm just going to stick to using v1. If you are following along on v2 and you run into issues, please let me know. Otherwise, everything should be very similar.

```
aws --version
aws-cli/1.27.74 Python/3.10.0 Darwin/22.5.0 botocore/1.21.65
```

If you don't have the AWS CLI installed at all on your machine, please refer to the instructions on this page and install it before moving one: https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html#getting-started-install-instructions

Once you have the CLI installed, let's configure our credentials by running:

```
aws configure
```

Paste in the access key ID first.

Then paste in your secret access key.

Select whatever default region you'd like — I like to stick to us-east-1.

And let's leave the default output format.

What this does is it creates or updates two files:

- `Credentials` file
- `Config` file

My `config` file looks like this:

```
vim ~/.aws/config

[profile christophe-cli]
region = us-east-1

[default]
region = us-east-1
```

As you can see, you can have multiple profiles set up. If you don't specify a profile when you're running commands, then AWS will use the `default` profile. Otherwise, you can pass in the profile name (like this: `aws iam list-groups --profile christophe-cli`) and it will use that profile.

My `credentials` file looks like this:

```
vim ~/.aws/credentials

[default]
aws_access_key_id = AKIAT6ZKEI3ERS4YCCQS
aws_secret_access_key = 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
```

Whenever you now execute a CLI command, AWS will look up the config file information, grab the default profile or whatever profile you specify, and then grab the corresponding credentials from the credentials file which has your access keys.

You'll notice I don't have credentials in this file for the `christophe-cli` profile, and that's because I prefer using a tool called the AWS Vault which stores your IAM credentials in your operating system's secure keystore and then generates temporary credentials instead of storing them in plaintext in the `credentials` file.

That is my preferred way of using IAM credentials for development, purposes, but I'm not going to force you to use that so I'm going to continue throughout this course without using it. Please feel free to use it though if you want.

Whatever method you choose, let's type in the command

```
aws iam list-groups
{
    "Groups": []
}
```

And you should see an empty group if you don't have any groups in this account, or you should see a list of groups being returned if you have created groups in this account. Either result is acceptable and proves that you've successfully set up your AWS CLI credentials.

If you get an error message back, then please double check the steps we just went through, and then ask for help if you get stuck!

Otherwise, we now have our admin level privileges for CloudGoat, but now we need to set up limited permissions for Pacu. So go ahead and complete this lesson and I'll see you in the next!

---

## Responses

**Wilson Daniele**
March 2, 2025

Hi,
I usually prefer using VMs for penetration testing, so I was thinking about setting up an environment like this:
VM1: Attacking Machine (e.g., Parrot, BlackArch, etc.) running Pacu
VM2: Target Machine (e.g., Debian) running LocalStack (https://www.localstack.cloud/) and CloudGoat

In this setup, I can use LocalStack to simulate AWS services, which avoids the risk of paying or exposing my actual AWS account. I haven't used LocalStack, Pacu, or CloudGoat before, so I plan to experiment and see how it works.
Let me know your thoughts!
Thanks

Reply

**Christophe**
March 2, 2025

VMs are a great fit for this so go for it! Even if you already have access to LocalStack, I would not recommend using it beyond just initially to get comfortable before actually doing it in AWS. LocalStack can be helpful to run some basic dev tests before pushing to AWS to prevent glaring mistakes from going out, but I would not use it for much more than that and definitely not for pentesting purposes. It will have lots of limitations and discrepancies because it's not going to be a high fidelity version of AWS, especially for IAM. At best you'll be limited in what you can do, at worst you'll get bad conclusions and learn bad habits that won't actually translate to AWS. Just my 2 cents!

Reply

**Wilson Daniele**
March 2, 2025

I understand what you mean. After hitting the two-hour limit (it was fun to test, but I had to focus on other things), I encountered several issues. Even though I managed to bypass or fix most of them, the problem with credentials not being recognized during terraform apply when creating the VPC made me reconsider.... it just wasn't worth it 😅

Reply