

Setting up AWS

If you're already an expert in AWS, what you'll need is

- IAM credentials (and stored locally too)
- A bucket (and region)

.and you can skip the rest of this section and go directly to Writing our function

If like me however, you're mostly new to managing AWS, not only can it be extremely overwhelming, but the help is also overwhelming too. In this section I'll briefly walk you through .what you need to get the above set up

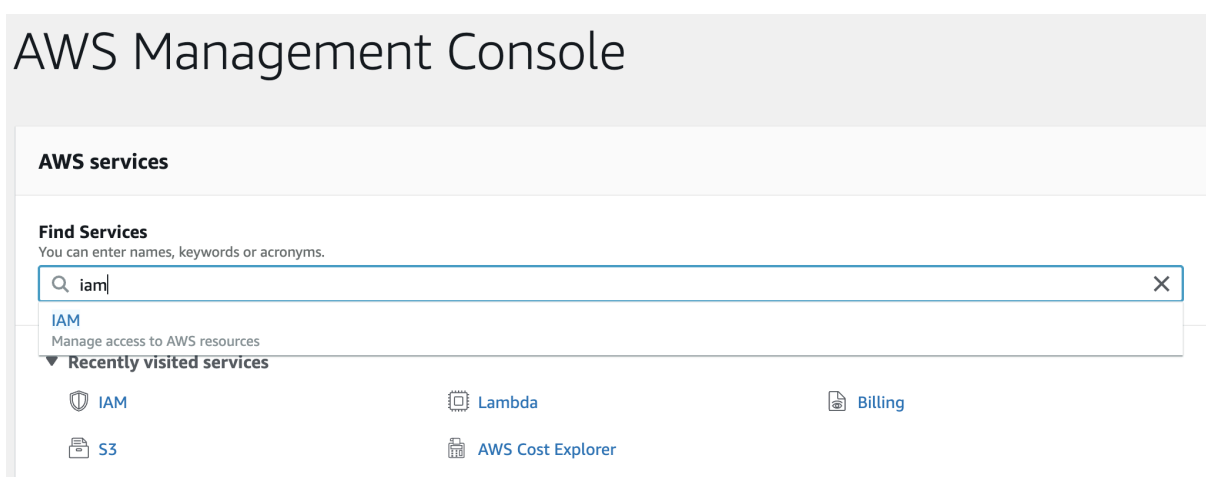
Step 1 - Create an AWS account

You need to provide credit card details, but stick to the free tier and you should be fine. In addition, make sure you set up some alerts in case you suddenly go beyond what free provides. For these demos !though you most certainly won't hit anything beyond free

While you may want to download AWS CLI for certain management aspects, note that Kotless .doesn't need it

Step 2 - Create IAM credentials

This is required by Kotless (Terraform actually) to deploy your functions. To do this, go to the AWS Management Console (make sure you're logged in) and search for IAM



Once in the IAM section, proceed to create a new user account by clicking on Users

Identity and Access Management (IAM)

Dashboard

▼ Access management

Groups

Users

Roles

Policies

Identity providers

Account settings

and then the **Add User** button

Add user

Delete user

🔍 Find users by username or access key

This takes you through a series of steps to provide information for the new user. It's important to remember here the name we give the user, which is the same one which will be defined on our local `.machine` for Kotless to use. For this example, I've named it **my.kotless.user**

Add user

1 2 3 4 5

Set user details

You can add multiple users at once with the same access type and permissions. [Learn more](#)

User name*

[+ Add another user](#)

Select AWS access type

Select how these users will access AWS. Access keys and autogenerated passwords are provided in the last step. [Learn more](#)

- Access type* ☒ **Programmatic access**
Enables an **access key ID** and **secret access key** for the AWS API, CLI, SDK, and other development tools.
- ☐ **AWS Management Console access**
Enables a **password** that allows users to sign-in to the AWS Management Console.


.Make sure **Programmatic access** is ticked


In the next step we're going to define permissions. Obviously this needs to be fine-tuned based on what's needed. For now we're going to give full Admin


Add user

1 2 3 4 5

Set permissions

 Add user to group

 Copy permissions from existing user

 Attach existing policies directly

Create policy ↺

Filter policies ▾

Showing 528 results

	Policy name ▾	Type	Used as
<input checked="" type="checkbox"/>	AdministratorAccess	Job function	Permissions policy (1)
<input type="checkbox"/>	AlexaForBusinessDeviceSetup	AWS managed	None
<input type="checkbox"/>	AlexaForBusinessFullAccess	AWS managed	None
<input type="checkbox"/>	AlexaForBusinessGatewayExecution	AWS managed	None

The next step we'll skip (as it's to define tags), leading us to the final step which is to review and .create the user

Add user

- 1
- 2
- 3
- 4
- 5

Review

Review your choices. After you create the user, you can view and download the autogenerated password and access key.

User details

User name	my.kotless.user
AWS access type	Programmatic access - with an access key
Permissions boundary	Permissions boundary is not set

Permissions summary

The following policies will be attached to the user shown above.

Type	Name
Managed policy	AdministratorAccess

Once done, you'll be prompted with the user along with two values: **Access key ID** and **Secret Access key**

Add user

- 1
- 2
- 3
- 4
- 5

✔

Success
You successfully created the users shown below. You can view and download user security credentials. You can also email users instructions for signing in to the AWS Management Console. This is the last time these credentials will be available to download. However, you can create new credentials at any time.

Users with AWS Management Console access can sign-in at:

 **Download .csv**

	User	Access key ID	Secret access key
▶	✔ my.kotless.user	<div></div>	<div></div> <div>Hide</div>

Step 3 - Create an S3 bucket

.Go back to the AWS Management Console and search for S3

Amazon S3

Buckets (1)
Buckets are the fundamental container in Amazon S3 for data storage. For others to access the objects in your buckets, you'll need to explicitly grant them permissions. [Learn more](#)

Copy ARN

Empty

Delete

Create bucket

Find bucket by name

< 1 > ⚙

Click on **Create bucket** providing a name and region. Again, keep track of these two values as we'll use them later

Create bucket

General configuration

Bucket name

Bucket name must be unique and must not contain spaces or uppercase letters. [See rules for bucket naming](#)

Region

Bucket settings for Block Public Access

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to this bucket or objects within, you can customize the individual settings below to suit your specific storage use cases. [Learn more](#)

☒ **Block all public access**

Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.

- ☒ **Block public access to buckets and objects granted through *new* access control lists (ACLs)**
S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.
- ☒ **Block public access to buckets and objects granted through *any* access control lists (ACLs)**
S3 will ignore all ACLs that grant public access to buckets and objects.
- ☒ **Block public access to buckets and objects granted through *new* public bucket or access point policies**
S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.
- ☒ **Block public and cross-account access to buckets and objects through *any* public bucket or access point policies**
S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.

► Advanced settings

Cancel

Create bucket

.Leave all other options as default

Step 4 - Store the credentials locally

Kotless is going to need access to the credentials created, and we need to somehow provide these. These are stored in the user directory (on macOS/Linux this would be ~/.aws and on Windows in the %home directory)

Create a file name ~/.aws/credentials and type in the following contents

```
[profile my.kotless.user]
{aws_access_key_id={the_access_key_id
{aws_secret_access_key={the_secret_access_key
```

Notice how the profile name matches the name of the IAM credential we created earlier

And that's it. We're now ready to write our function and deploy with Kotless

Important - When you set up your AWS account, the system itself asks you to follow a series of good practices, such as removing root access, setting up MFA, defining groups with restricted permissions, etc. It's important to go back and do this at some point. I'm avoiding it in here cause I know HOW EXCITED YOU ARE TO SEE THIS WORK! So let's move on