

# Security Learning Hub

## AWS Identity and Access Management (IAM)

### Overview

Here we'll cover what AWS Identity and Access Management (IAM) is. With AWS Identity and Access Management (IAM), you can specify who or what can access services and resources in AWS, centrally manage fine-grained permissions, and analyze access to refine permissions across AWS. We'll cover what IAM users, groups, roles, and policies are and I will show how to audit these resources. While this will not be an exhaustive look at the topic, it will get you started on the journey to learning more.

### AWS IAM Users and Groups

IAM can be used to create users, groups, roles, and policies. We're going to start here with creating users and groups. The first thing you do when you create an AWS account is create a root user for the account. The root account has full access over all aspects of the account. It's an IAM best practice to create individual users and to not use the Root account after initial creation. Here is a brief overview of what users, groups, and policies are in AWS:

- Root User - initial user created when opening an AWS account - this is mandatory.
- IAM users are individual accounts (principals).
- IAM accounts can be authenticated with passwords (AWS Console) or API keys (Programmatic access).
- Groups are collections of users. Users can be members of up to 10 groups.
- You can assign permissions to all members of the group by attaching IAM policies (policies are permissions you're providing).
- Up to 5000 individual user accounts can be created. Users have no permissions by default.

When creating groups, consider the job functions for the groups. For example, people in HR will need access to different resources than those in an Administrators group. Keep in mind that users can be a part of 10 Groups total. After you decide the function of the group, you will need to attach a policy to it. Keeping in mind the security principle of least privilege access, apply the policy that provides this so that all the users in the group are provided the adequate level of access necessary to complete their work.

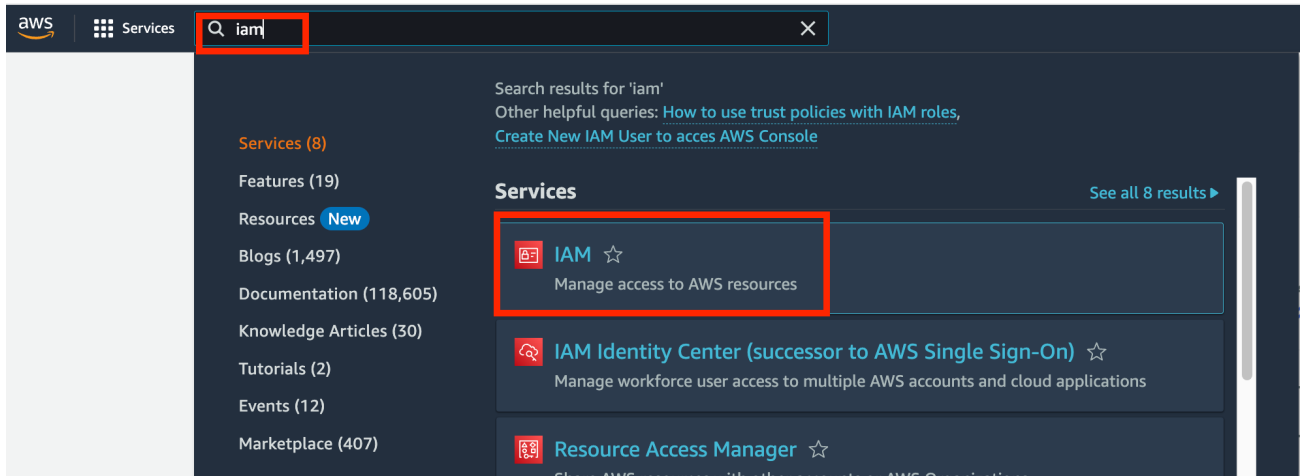
Note: By default, AWS implements the principle of least privilege access on resources. All access is denied by default and needs to be explicitly allowed. IAM is the gatekeeper that validates who

you are assesses whether you should be allowed to access a specific resource. Users, groups, and roles define the identity and policies define the permissions.

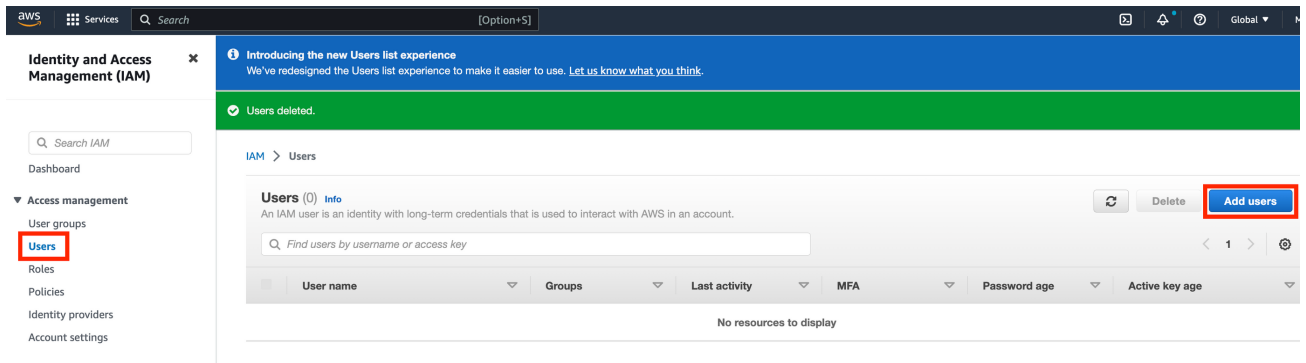
While we are not going to go through the process of creating the root account, there is a walkthrough of how to do this on the AWS site here

<<https://aws.amazon.com/premiumsupport/knowledge-center/create-and-activate-aws-account/>>. Now, with the root account already created, let's get to creating 2 new users (Operations-Eddie and HR-Edna) and 2 new groups (Operations and HR) where we will place our users in to.

1. Type in **IAM** in the search menu and select **IAM** under Services.



2. On the left, choose **Users** and then select the **Add users** option.



3. Type in the new username. **Operations-Eddie** is the account we are creating here. We will not enable AWS Management Console access for this account and we'll walk through the process of creating access keys for programmatic access later in this tutorial.

IAM > Users > Create user

Step 1  
Specify user details

Step 2  
Set permissions

Step 3  
Review and create

### Specify user details

**User details**

User name

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

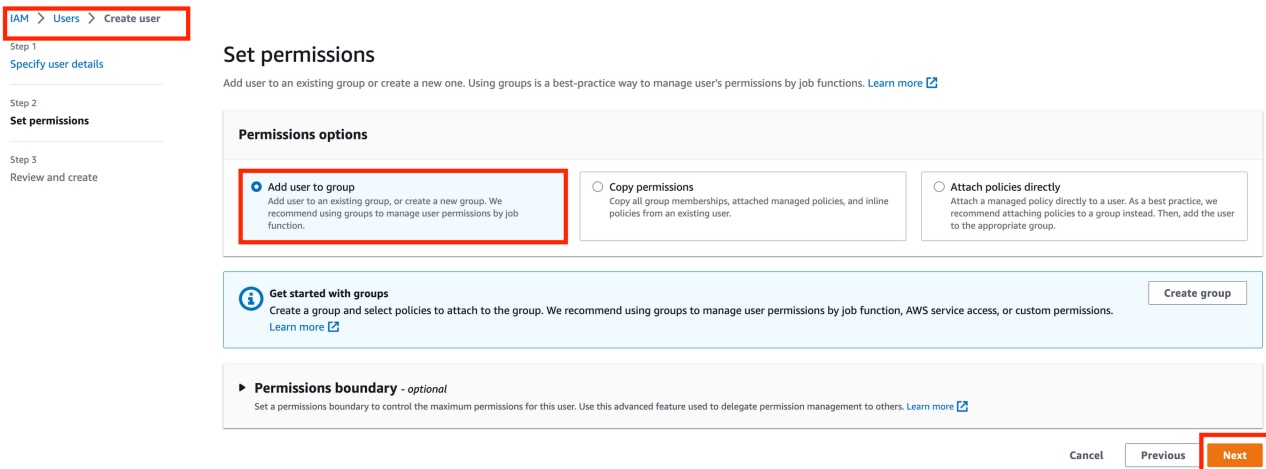
Enable console access - *optional*  
Enables a password that allows users to sign in to the AWS Management Console.

**We will only provide programmatic access for these two accounts (I will show how to generate the command line credentials shortly) . If we desire to provide GUI access via the AWS Management Console, we select this option.**

For programmatic access, you can generate access keys after you create the user. [Learn more](#)

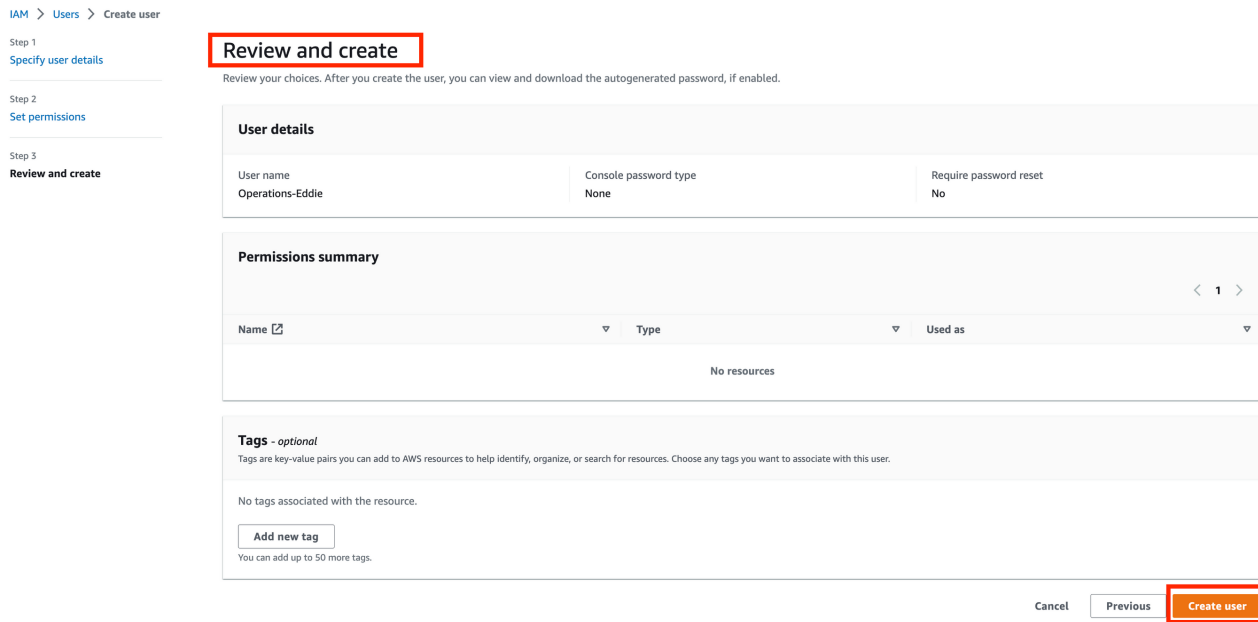
Cancel **Next**

4. Since we do not have any groups yet created, we will add them later. In the meantime, select **Add user to group** and select next.

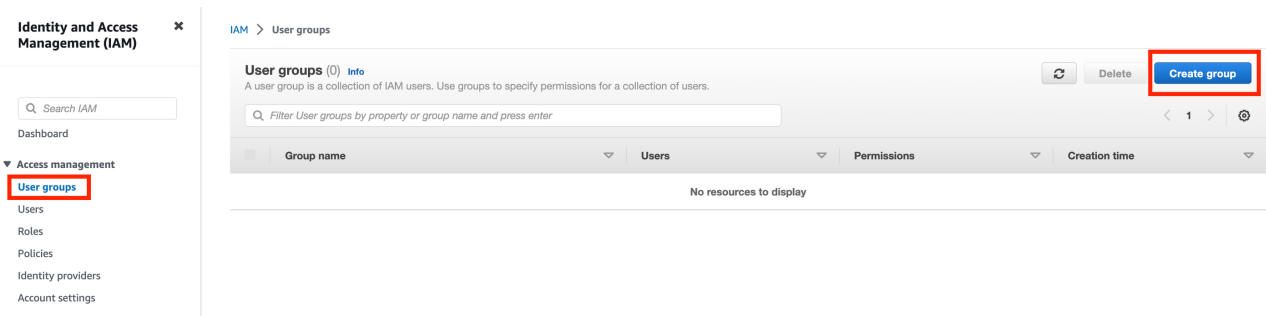


5. At this point you will be provided a summary for the user. If everything looks good, choose **Create user**.

Note: I will not walk through the process of creating the additional user



6. Now let's create the group and add our users to each respective group. Now select **User groups** from the left hand menu and choose **Create group**.



7. Here we will create the group name of **Operations** and add **Operations-Eddie** to the group.

**Identity and Access Management (IAM)**

Search IAM

Dashboard

Access management

- User groups
- Users
- Roles
- Policies
- Identity providers
- Account settings

Access reports

- Access analyzer
- Archive rules
- Analizers
- Settings
- Credential report
- Organization activity
- Service control policies (SCPs)

### Create user group

**Name the group**

User group name  
Enter a meaningful name to identify this group.

Operations

Maximum 128 characters. Use alphanumeric and +,=,\*,@ characters.

**Add users to the group - Optional** (Selected 1/2) Info

An IAM user is an entity that you create in AWS to represent the person or application that uses it to interact with AWS. A user can belong to up to 10 groups.

Search

User name	Groups	Last activity	Creation time
<input type="checkbox"/> HR-Edna	0	None	Now
<input checked="" type="checkbox"/> Operations-Eddie	0	None	1 minute ago

8. While we will delve in to policies later in this article, we will attach the AWS managed **AmazonS3FullAccess** policy to the Operations group. This will effectively provide full access permission to S3 for members of this group. You can simulate and audit this access via multiple tools (e.g. Access Analyzer, Access Advisor, Simulate, etc.) within AWS IAM if you desire to assess this and tighten up permissions in the future.

### Attach permissions policies - Optional

(Selected 2/810) Info

You can attach up to 10 policies to this user group. All the users in this group will have permissions that are defined in the selected policies.

Filter policies by property or policy name and press enter. 9 matches

"S3" Clear filters

Policy name	Type	Description
<input type="checkbox"/> AmazonDMSRedshiftS3Role	AWS managed	Provides access to manage S3 settings for Redshi...
<input checked="" type="checkbox"/> AmazonS3FullAccess	AWS managed	Provides full access to all buckets via the AWS Ma...

#### AmazonS3FullAccess

Provides full access to all buckets via the AWS Management Console.

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

While I did not provide a full walkthrough for each user and group, you got to see a single example for each. Let's take a look at how to provide programmatic access and then move on to policies!

1. Select the user you need to create programmatic access for. This will allow the user to use the command line interface to interact with the services you provisioned access to.

IAM > Users > HR-Edna

## HR-Edna

Delete

### Summary

ARN  
/HR-Edna

Console access  
Disabled

Access key 1  
Not enabled

Created  
February 13, 2023, 15:15 (UTC-05:00)

Last console sign-in  
-

Access key 2  
Not enabled

Permissions | Groups (1) | Tags | **Security credentials** | Access Advisor

2. Scroll down on the page and find **Access keys** and select it.

### 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. [Learn more](#)

Create access key

#### No access keys

As a best practice, avoid using long-term credentials like access keys. Instead, use tools which provide short term credentials. [Learn more](#)

Create access key

3. You can see AWS suggests alternative methods for accessing resources at the bottom of the page, but in this example we will choose **Command Line Interface (CLI)** and select **Next**.

## Access key best practices & alternatives

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

### 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 enable an application running on an on-premises host, or to use a local AWS client or third-party AWS plugin.

### 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](#)

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

Cancel

Next

4. I did not define a tag on the next screen and selected **Create access key**

### Set description tag - optional

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.

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

Cancel Previous **Create access key**

5. At this time choose to **Download .csv file** which contains the credentials. You can also view the secret access key - it will not be visible in the **AWS Console** from this point forward, which is why it is important to save the csv file unless you plan to use the credentials yourself. You may be asking yourself - since this file contains sensitive credentials, why would I want to save it locally non-encrypted! Good point! This is why using MFA and other technologies is important to making sure you have a secure setup. While this is not a discussion I will make here, there are other methods to provision access that are more secure.

### Retrieve access keys

**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
AKIAIOSFODNN7EXAMPLEQ5S	***** <a href="#">Show</a>

**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](#).

Download .csv file

Done

## [AWS IAM Roles](#)

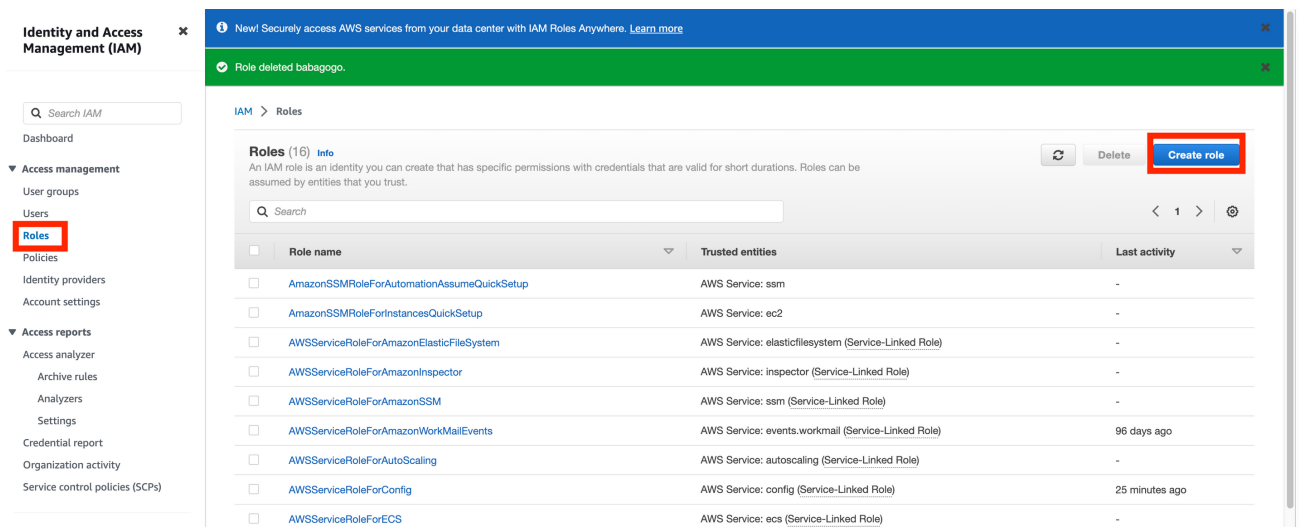
An IAM role is an IAM identity that has specific permissions tied to it. Roles are assumed by users, applications, and services. Once assumed, the identity becomes the role to gain the role's permissions. This eliminates the need to assign an individual all the permissions for the task and

instead they can be assigned a specific role that contains all the necessary permissions and use it instead to accomplish what they need to.

- Helpful to give temporary permissions for a specific task
- Allow a user/application to perform many actions in a different account
- Permissions expire over time
- Common roles: EC2 Instance Roles
- Common roles: Lambda Function Roles
- Common roles: Roles for CloudFormation

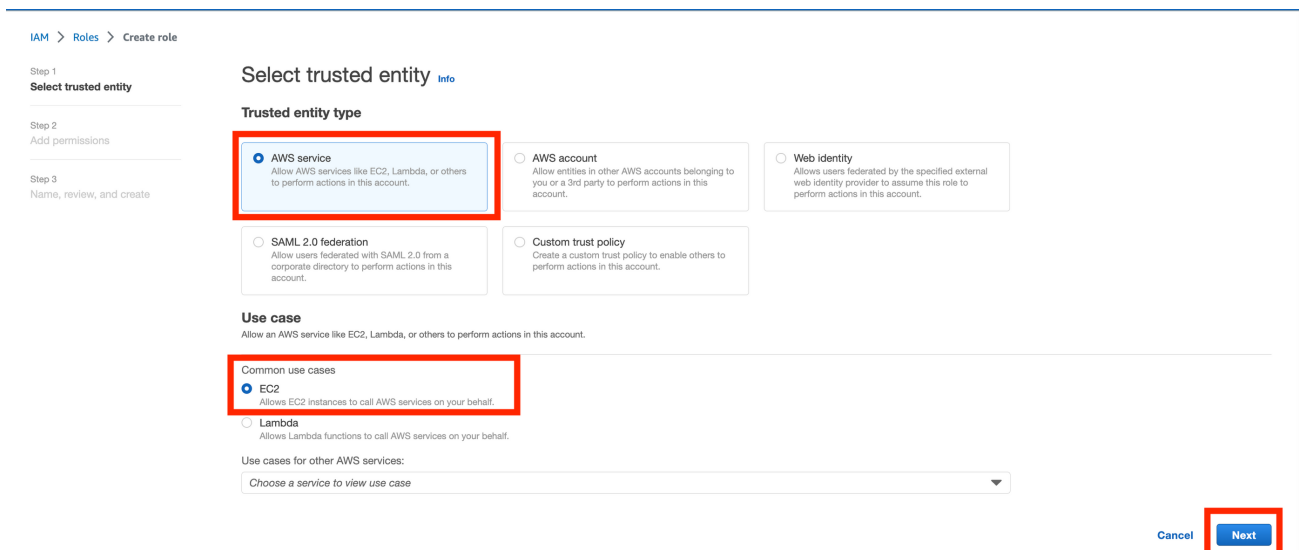
I'll walk through creating a role and showing how to apply it to an EC2 instance. When users connect or use that instance, they will be able to access the resources permissions were granted for without having to enter their aws credentials in the EC2 instance to do so. Let's go!

## 1. Let's browse to **Roles** in IAM and **Create role**



The screenshot shows the AWS IAM console interface. On the left is a navigation menu with 'Roles' highlighted. The main content area shows a list of roles with columns for 'Role name', 'Trusted entities', and 'Last activity'. A 'Create role' button is highlighted with a red box in the top right corner of the main content area.

## 2. Out next screen shows the options we have available. We will choose **AWS service** and **EC2**.



The screenshot shows the 'Create role' wizard in the AWS IAM console, specifically Step 1: 'Select trusted entity'. The 'AWS service' option is selected and highlighted with a red box. Below it, the 'EC2' option under 'Common use cases' is also highlighted with a red box. A 'Next' button is highlighted with a red box in the bottom right corner.

### 3. We will grant multiple permissions to this role.

IAM > Roles > Create role

Step 1  
Select trusted entity

Step 2  
Add permissions

Step 3  
Name, review, and create

#### Add permissions [Info](#)

Permissions policies (Selected 2/816) [Info](#)

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

Filter policies by property or policy name and press enter. 9 matches

"iam" X Clear filters

<input type="checkbox"/>	Policy name <a href="#">↗</a>	Type	Description
<input type="checkbox"/>	<a href="#">IAMQuickSightListIAM</a>	AWS m...	Allow QuickSight to list IAM entities
<input type="checkbox"/>	<a href="#">IAMSelfManageServiceSpecificCreden...</a>	AWS m...	Allows an IAM user to manage their own Service Specific Credentials.
<input type="checkbox"/>	<a href="#">IAMFullAccess</a>	AWS m...	Provides full access to IAM via the AWS Management Console.
<input type="checkbox"/>	<a href="#">IAMUserChangePassword</a>	AWS m...	Provides the ability for an IAM user to change their own password.
<input checked="" type="checkbox"/>	<a href="#">IAMReadOnlyAccess</a>	AWS m...	Provides read only access to IAM via the AWS Management Console.
<input type="checkbox"/>	<a href="#">IAMUserSSHKeys</a>	AWS m...	Provides the ability for an IAM user to manage their own SSH keys.
<input type="checkbox"/>	<a href="#">IAMAccessAdvisorReadOnly</a>	AWS m...	This policy grants access to read all access information provided by IAM access advisor such as service last accessed inform...
<input type="checkbox"/>	<a href="#">IAMAccessAnalyzerReadOnlyAccess</a>	AWS m...	Provides read only access to IAM Access Analyzer resources
<input type="checkbox"/>	<a href="#">IAMAccessAnalyzerFullAccess</a>	AWS m...	Provides full access to IAM Access Analyzer

Set permissions boundary - optional [Info](#)

Set a permissions boundary to control the maximum permissions this role can have. This is not a common setting, but you can use it to delegate permission management to others.

Cancel Previous **Next**

### 4. Give a name to the new role.

IAM > Roles > Create role

Step 1  
Select trusted entity

Step 2  
Add permissions

Step 3  
Name, review, and create

#### Name, review, and create

##### Role details

Role name

Enter a meaningful name to identify this role.

Maximum 64 characters. Use alphanumeric and '+', '@', '-' characters.

Description

Add a short explanation for this role.

Allows EC2 instances to call AWS services on your behalf.

Maximum 1000 characters. Use alphanumeric and '+', '@', '-' characters.

##### Step 1: Select trusted entities

```
1- {
2-   "Version": "2012-10-17",
3-   "Statement": [
4-     {
5-       "Effect": "Allow",
6-       "Action": [
7-         "sts:AssumeRole"
8-       ],
9-       "Principal": {
10-        "Service": [
11-          "ec2.amazonaws.com"
12-        ]
13-      }
14-    }
15-  ]
16- }
```

Step 2: Add permissions

### 5. Here we can see the 2 permission sets we provided to the Role. Now we will **Create rule** and then move to EC2 to attach the role to an instance.

Step 2: Add permissions

Permissions policy summary

Policy name <a href="#">↗</a>	Type	Attached as
<a href="#">IAMReadOnlyAccess</a>	AWS managed	Permissions policy
<a href="#">AmazonS3FullAccess</a>	AWS managed	Permissions policy

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 tag

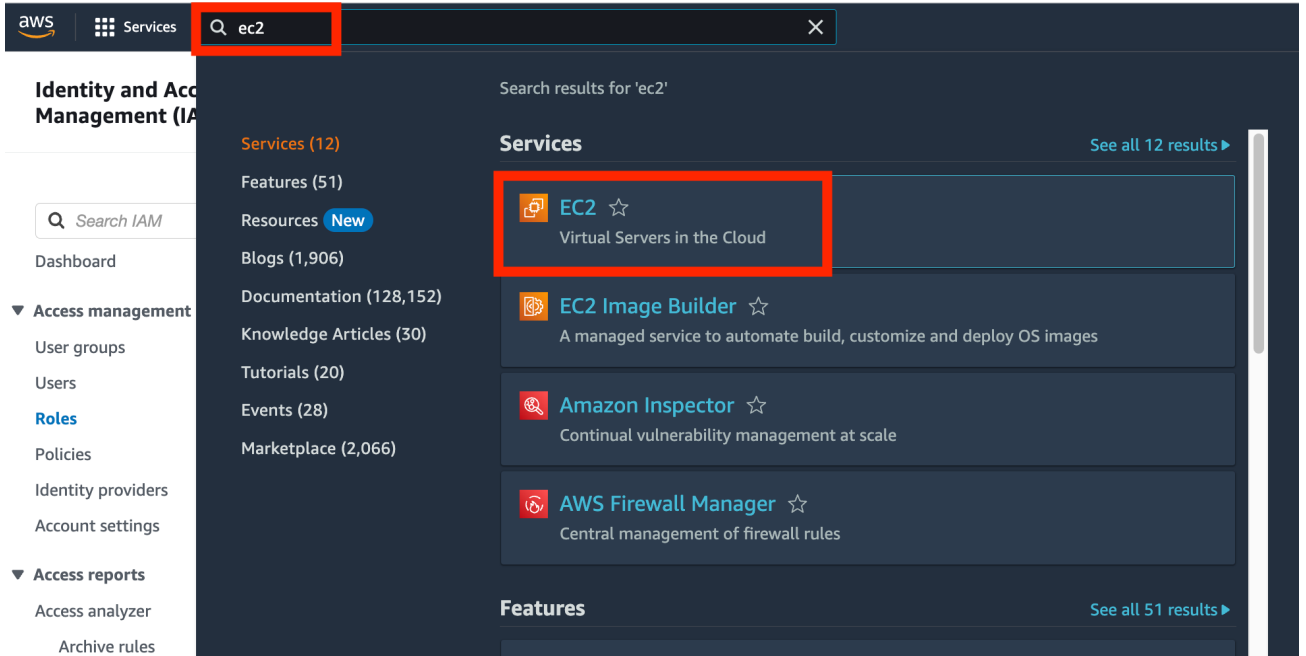
You can add up to 50 more tags.

Cancel

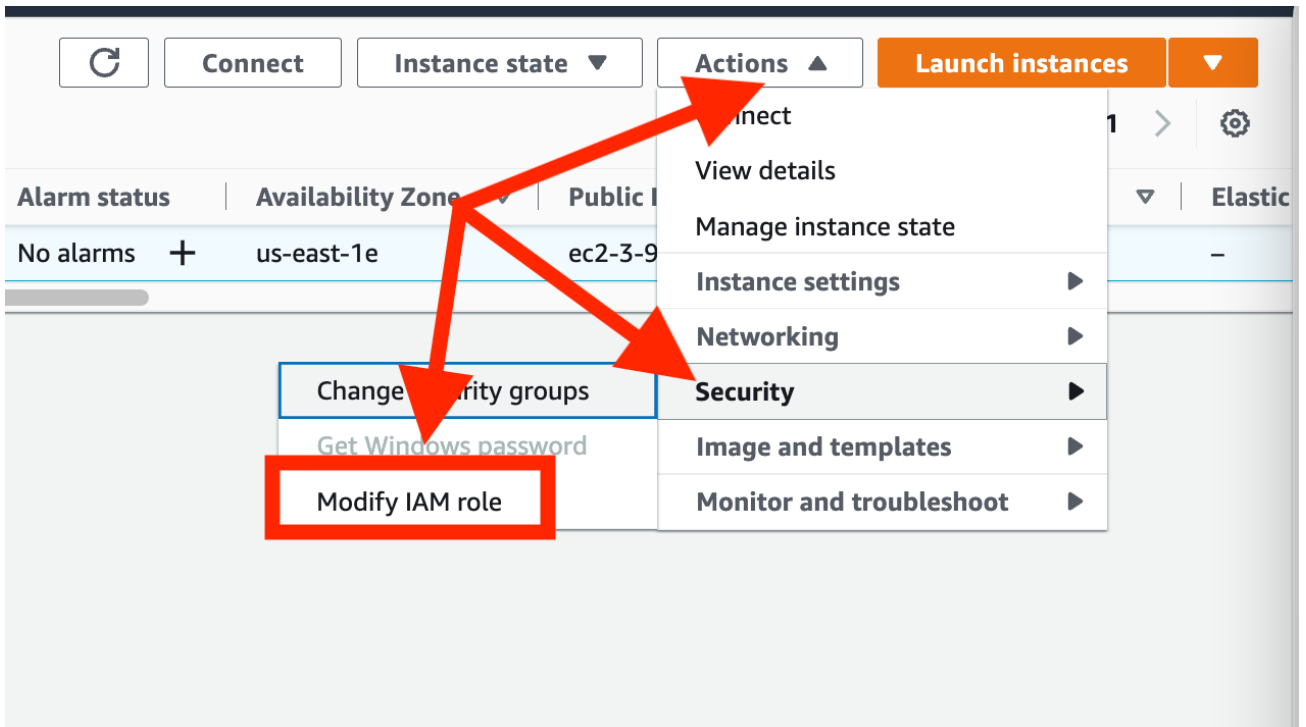
Previous

Create role

## 6. Browse to EC2



7. Select the instance you want to apply the role to. Here is how you attach it to the EC2 instance.



8. Now that we are complete, let's use the built-in SSH Connect to connect and run some commands on the instance. I first want to show that no access credentials have been entered in the Instance. However, we are able to list users and list buckets while logged in with the ssh credentials of ec2-user. So anyone who logs in to this instance with those credentials - or if another account is created on the instance - they will be able to run the same commands. The final command I am denied access. The reasons for this is that I deleted the s3 permissions in the role and this effectively removed those permissions immediately. Keep in mind this can be a very risky action if you do not understand what the permissions govern in the role - if this was an automated order process being done via a web front end on the ec2 instance, this action could prevent orders from running or data from being retrieved. I just want to mention this as it could cause problems in a production environment.

```

  _ | _ | _ |
  _ | ( _ | _ | )
  _ | \ _ | _ |

Amazon Linux 2 AMI

https://aws.amazon.com/amazon-linux-2/
16 package(s) needed for security, out of 16 available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-62-198 ~]$ aws configure
AWS Access Key ID [None]: ^C
[ec2-user@ip-172-31-62-198 ~]$ aws iam list-users --output text|cut -f 6
HR-Debbie
[ec2-user@ip-172-31-62-198 ~]$ aws s3 ls
2020-10-25 01:13:33
2020-06-22 02:29:17
2020-06-22 02:34:23
2022-02-02 21:22:20

[ec2-user@ip-172-31-62-198 ~]$ aws s3 ls
An error occurred (AccessDenied) when calling the ListBuckets operation: Access Denied
[ec2-user@ip-172-31-62-198 ~]$
```

## [AWS IAM Policies](#)

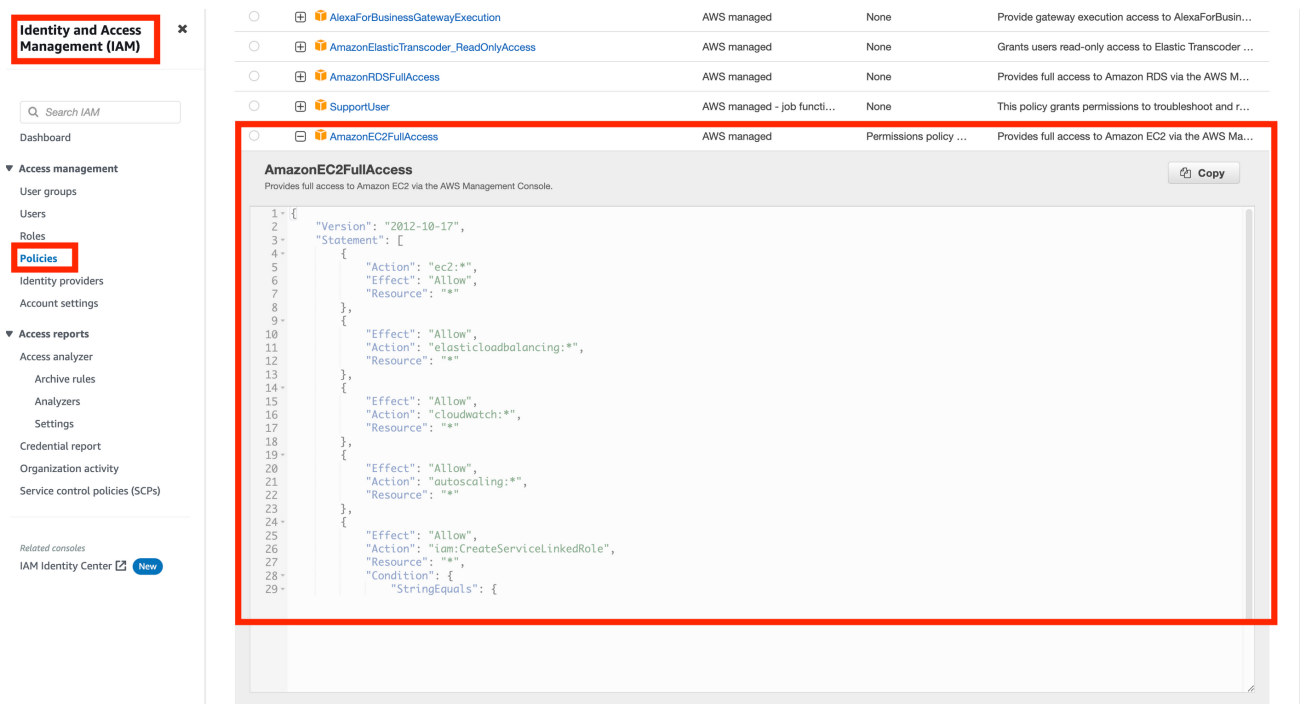
Here we will go in to deeper detail on AWS identity-based policies (which are attached to an IAM user, group, or role) and explain the nuances of a policy. An AWS policy is a document which defines one or more permissions and define who has access, as well as what actions can be done and under which conditions. Let's first start with the couple different types of policies we will show in the images (additional examples are provided, below) :

1. **AWS Managed Policies** -these policies are created and maintained by AWS.
  - a. Purpose built for specific job actions/functions.
  - b. Can change at any time - customer cannot change policy.

2. **Customer Managed Policies** - these policies are created and maintained by customer.

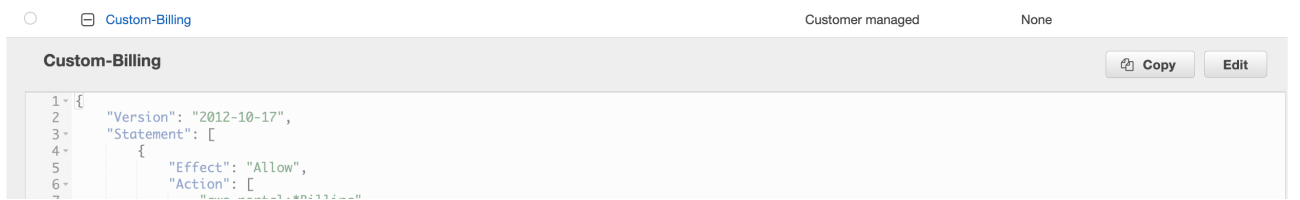
- a. You define what you would like to allow and disallow.
- b. I will often copy an AWS Managed Policy and modify it for my own needs.

1. In order to access the policies, go in to **IAM** and select **Policies** from the left hand menu as highlighted below. I've highlighted the **AmazonEC2FullAccess** policy - many additional policies are available here for use. The majority are **AWS Managed Policies**, unless you have created a lot of **Customer Managed Policies**.



2. Below I share 2 different policies. The first is a customer managed policy for billing and the 2nd example is an **AWS managed policy** which has some different actions defined from my custom policy. Let's go over some of the components in this JSON policy:

- **Version:** 2012-10-17. This is the current version of the policy language. This should not be changed - if it is, it will break the policy.
- **Statement:** The Statement element is the main element for a policy. This element is required. The Statement element can contain a single statement or an array of individual statements.
- **Effect:** This element is required and specifies whether the statement results in an **Allow** or an explicit **Deny**.
- **Action:** The Action element describes the specific action or actions that will be allowed or denied.
- **Resource:** The Resource element specifies the object or objects that the statement covers. This can be defined for specific ARNs - S3 bucket, object in a bucket, etc.
- **Note:** This is a very brief overview (see link **AWS IAM JSON policy elements reference** for many more examples and further elaboration)



```

7     "aws-portal:*Billing",
8     "aws-portal:*Usage",
9     "aws-portal:*PaymentMethods",
10    "billingconductor:*",
11    "organizations:ListAccounts",
12    "pricing:DescribeServices"
13  ],
14  "Resource": "*"
15 }
16 ]
17 }

```

Billing AWS managed - job function    Permissions policy (1)    Grants permissions 1

**Billing** Copy

Grants permissions for billing and cost management. This includes viewing account usage and viewing and modifying budgets and payment methods.

```

1- {
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Effect": "Allow",
6       "Action": [
7         "aws-portal:*Billing",
8         "aws-portal:*Usage",
9         "aws-portal:*PaymentMethods",
10        "budgets:ViewBudget",
11        "budgets:ModifyBudget",
12        "ce:UpdatePreferences",
13        "ce:CreateReport",
14        "ce:UpdateReport",
15        "ce>DeleteReport",
16        "ce:CreateNotificationSubscription",
17        "ce:UpdateNotificationSubscription",
18        "ce>DeleteNotificationSubscription",
19        "cur:DescribeReportDefinitions",
20        "cur:PutReportDefinition",
21        "cur:ModifyReportDefinition",
22        "cur>DeleteReportDefinition",
23        "purchase-orders:*PurchaseOrders"
24      ],
25      "Resource": "*"
26    }
27  ]
28 }

```

Here are some other examples of policies that I did not go into:

- **Organization Service Control Policy (SCP):** Defines the max permissions that can be granted by any identity or resource-based policy within the entire org
- **Resource-based policy:** Attached to a resource like S3, EC2, etc
- **Identity-based policy:** Attached to an identity (shown here)
- **Inline Policies:** applied to one group or one user. This is embedded in an IAM identity (a user, group, or role). Usually you want to avoid using these, but in some cases these are useful.
- **Session policy:** Limited permissions for a session
- **Permissions boundary:** Defines the maximum permissions that can be granted to a specific principal
- **Trust policy:** is also an example of a resource- based policy

The options that can be used in policies can be quite vast! I will revisit this in the future where I will go in to more detail and depth on configuring and using policies. For now, let's move on to viewing some of things we can determine from the command line.

## [AWS IAM - View Details from Command Line](#)

Here are some basic commands with highly redacted content.

List users and groups:

- `aws iam list-users --output text|cut -f 6`
- `aws iam list-groups --output text|cut -f 5`

```

[securitylearninghub >aws iam list-users --output text --profile seclearninghub|cut -f 6
betty
US 5

```

```
HR-Edna  
Operations-Eddie
```

```
securitylearninghub >aws iam list-groups --output text --profile seclearninghub|cut -f 5  
HR  
Operations
```

To generate and then obtain a credential report:

- `aws iam generate-credential-report`
- `aws iam get-credential-report --output text --query Content|base64 -d`

For a much more exhaustive list of useful commands, see my website reference here:

<https://www.securitylearninghub.com/#!aws-cli.md>

## [AWS IAM - Best Practices](#)

Here are some IAM best practices to keep in mind as you move forward with using AWS IAM:

- Lock away your AWS account root user access keys
- Create individual IAM users
- Use groups to assign permissions to IAM users
- Grant least privilege
- Get started using permissions with AWS managed policies
- Use customer managed policies instead of inline policies
- Use access levels to review IAM permissions
- Configure a strong password policy for your users
- Enable MFA
- Use roles for applications that run on Amazon EC2 instances
- Use roles to delegate permissions
- Do not share access keys
- Rotate credentials regularly
- Remove unnecessary credentials
- Use policy conditions for extra security
- Monitor activity in your AWS account - think CloudTrail and Access Analyzer

## [Conclusion](#)

We covered the basics of AWS IAM and went over how to view accounts and groups via the AWS Command Line. AWS IAM is an exhaustive topic and significantly more complicated than the brief look we had of it here. There are resources I provided in order for your to further your learning on this topic so enjoy reviewing those. Hope you enjoyed this article - please contact us at **admin[at]securitylearninghub[dot]com** with any comments or questions.

## [References](#)

### **AWS IAM Policies and permissions:**

[https://docs.aws.amazon.com/IAM/latest/UserGuide/access\\_policies.html](https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies.html)

**AWS IAM Policy Evaluation:**

[https://docs.aws.amazon.com/IAM/latest/UserGuide/reference\\_policies\\_evaluation-logic.htm](https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_evaluation-logic.htm)

**AWS Amazon Resource Names (ARNs)** :<https://docs.aws.amazon.com/general/latest/gr/aws-arns-and-namespaces.html>

**AWS IAM JSON policy elements reference:**

[https://docs.aws.amazon.com/IAM/latest/UserGuide/reference\\_policies\\_elements.html](https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_elements.html)

**AWS Command Line - Useful Commands:** <https://www.securitylearninghub.com/#!aws-cli.md>