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.

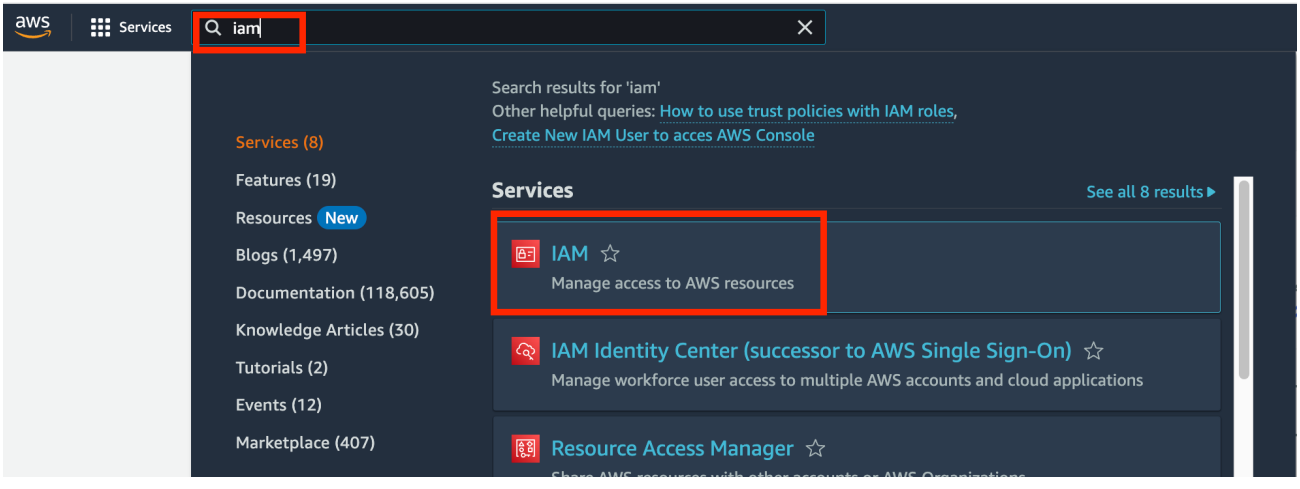
When creating groups, consider the job functions for the groups. For example, people in HR will need access to different resources than those in an Administrators group. Keep in mind that users can be a part of 10 Groups total. After you decide the function of the group, you will need to attach a policy 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

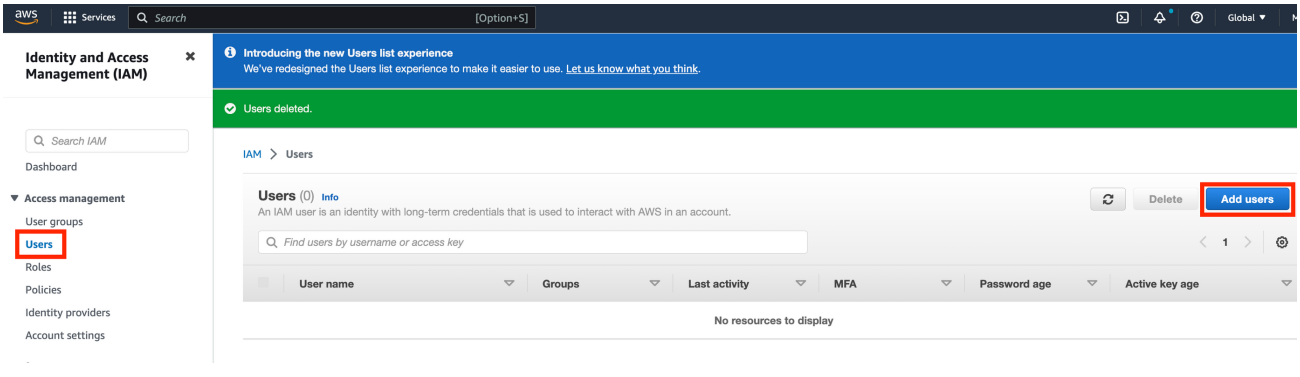
you are assesses whether you should be allowed to access a specific resource. Users, groups, and roles define the identity and policies define the permissions.

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.

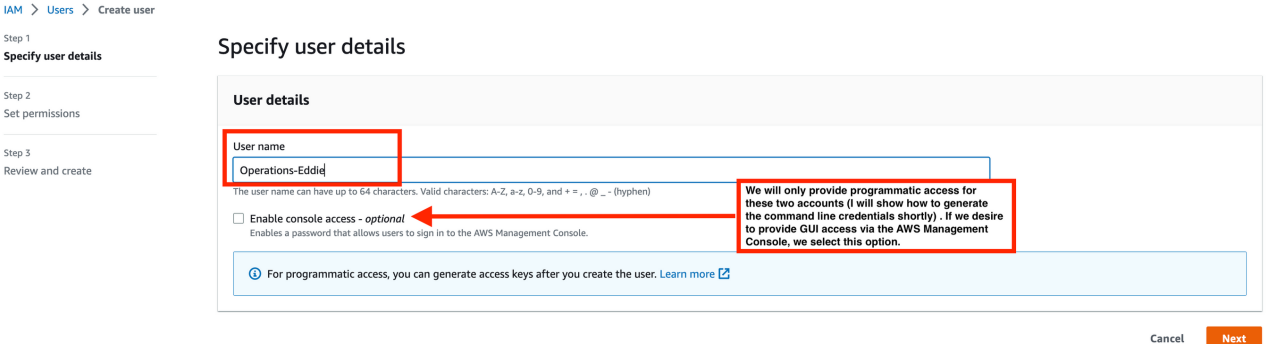
1. Type in **IAM** in the search menu and select **IAM** under Services.



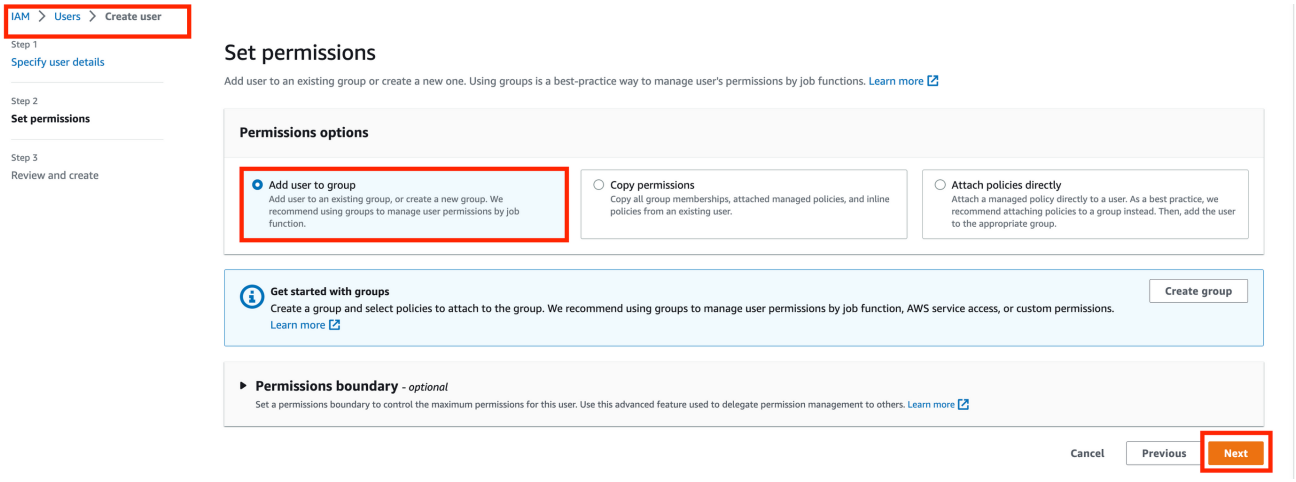
2. On the left, choose **Users** and then select the **Add users** option.



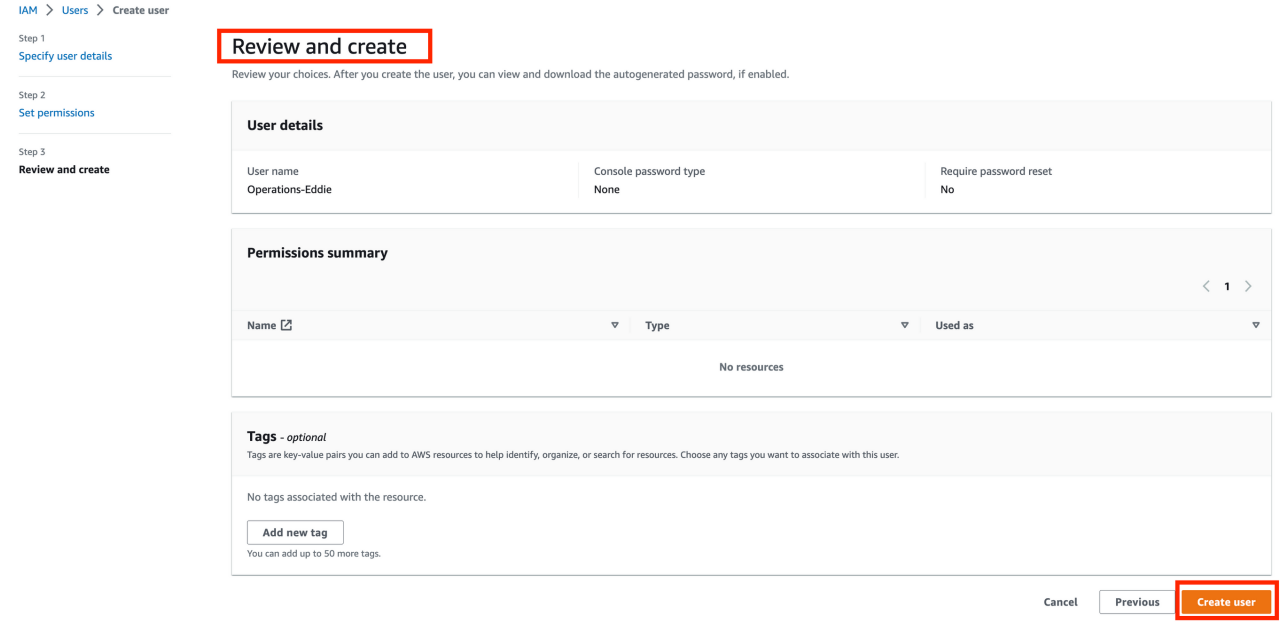
3. Type in the new username. **Operations-Eddie** is the account we are creating here. We will not enable AWS Management Console access for this account and we'll walk through the process of creating access keys for programmatic access later in this tutorial.



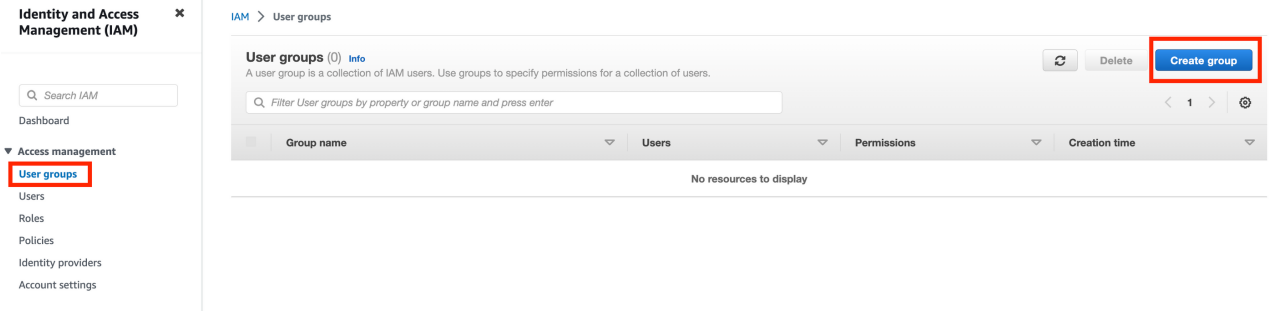
4. Since we do not have any groups yet created, we will add them later. In the meantime, select **Add user to group** and select next.



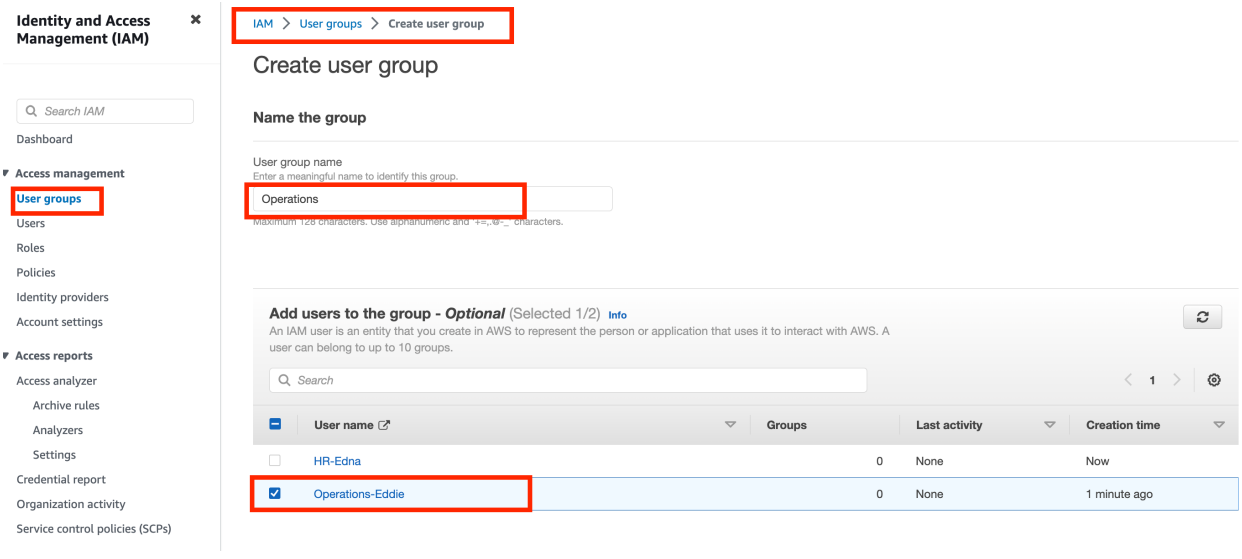
5. At this point you will be provided a summary for the user. If everything looks good, choose **Create user**.
   Note: I will not walk through the process of creating the additional user



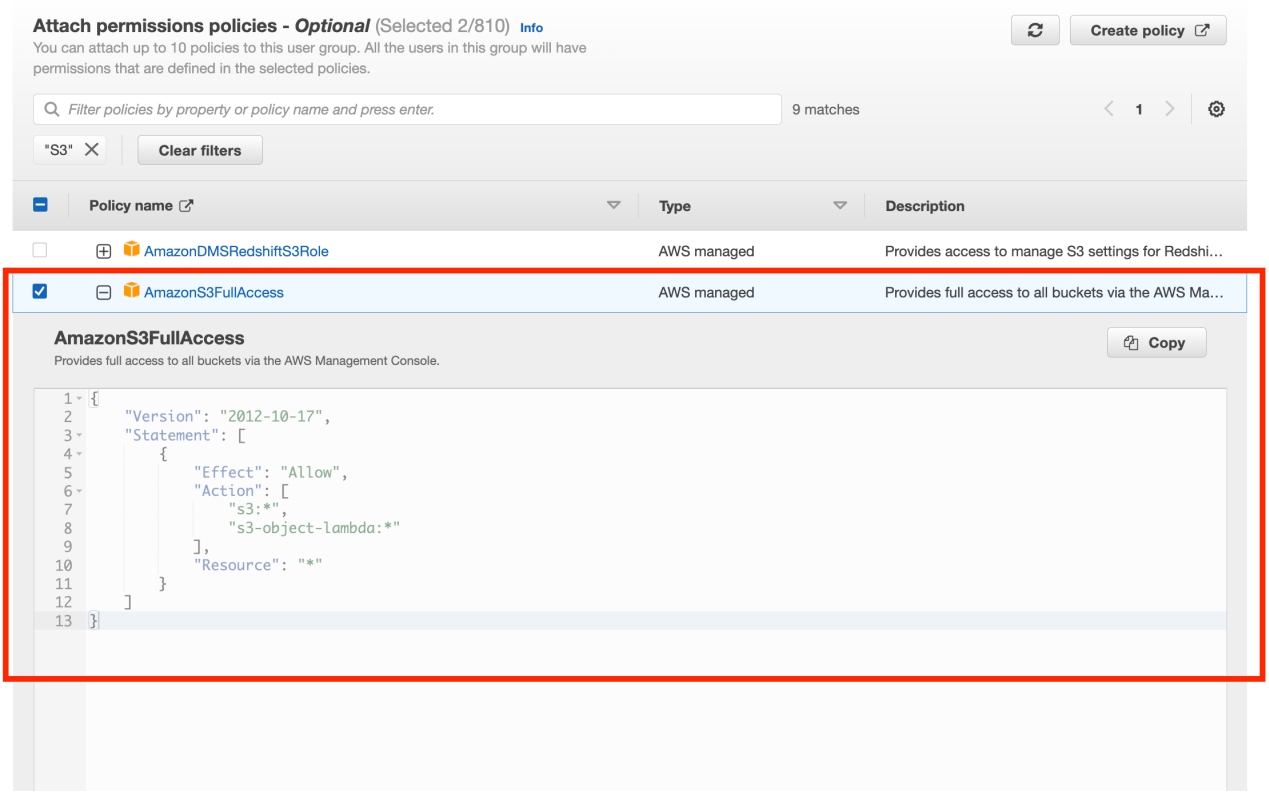
6. Now let's create the group and add our users to each respective group. Now select **User groups** from the left hand menu and choose **Create group**.



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

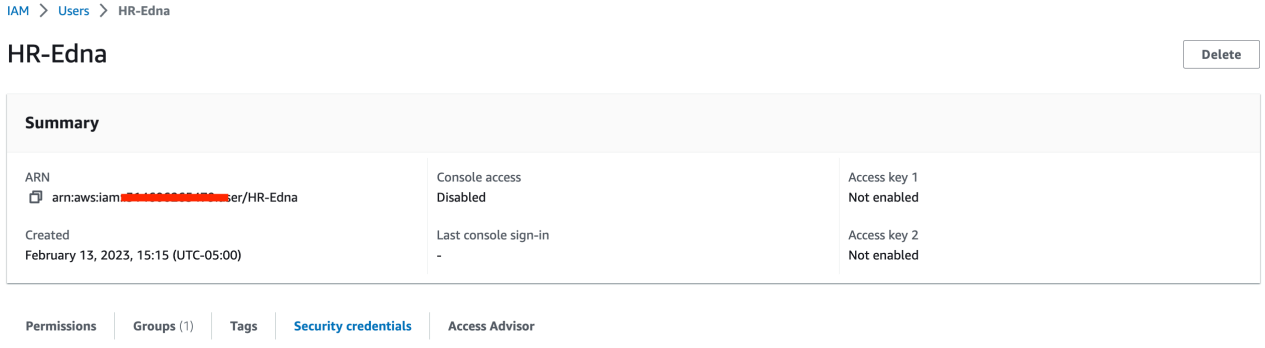


8. While we will delve in to policies later in this article, we will attach the AWS managed **AmazonS3FullAccess** policy to the Operations group. This will effectively provide full access permission to S3 for members of this group. You can simulate and audit this access via multiple tools (e.g. Access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

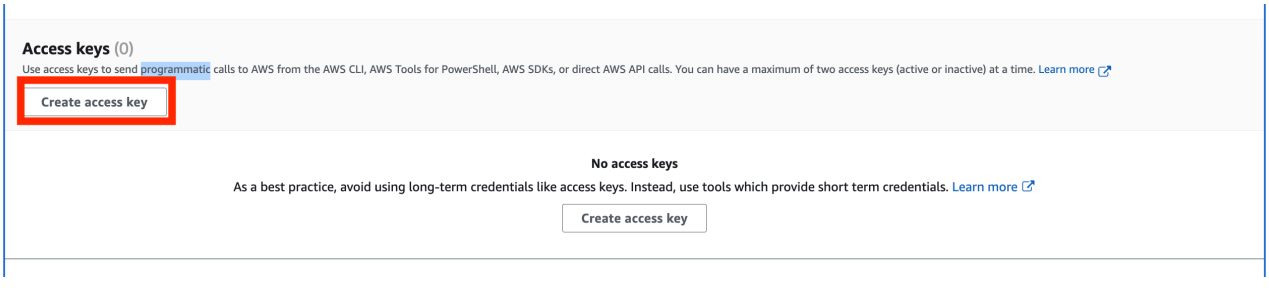


While I did not provide a full walkthrough for each user and group, you got to see a single example for each. Let's take a look at how to provide programmatic access and then move on to policies!

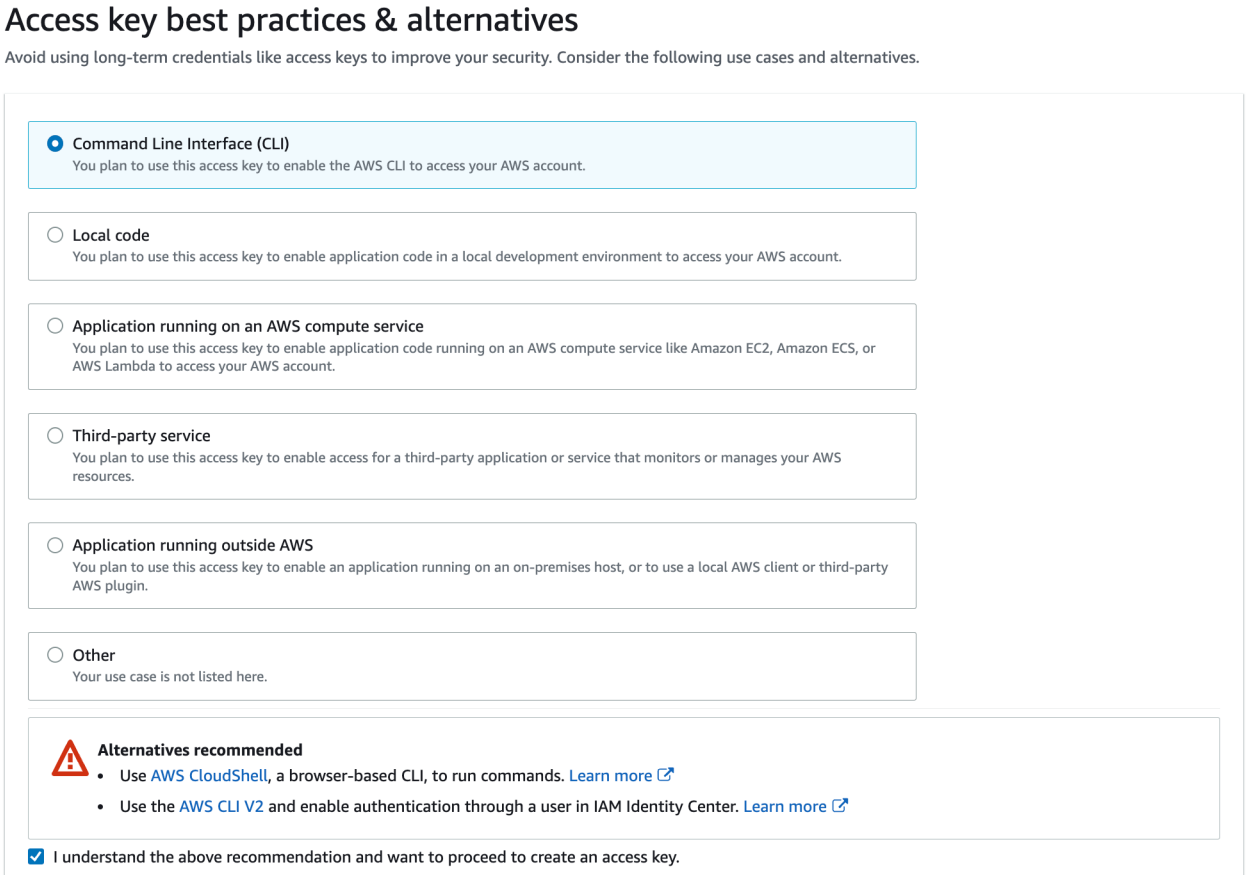
1. Select the user you need to create programmatic access for. This will allow the user to use the command line interface to interact with the services you provisioned access to.



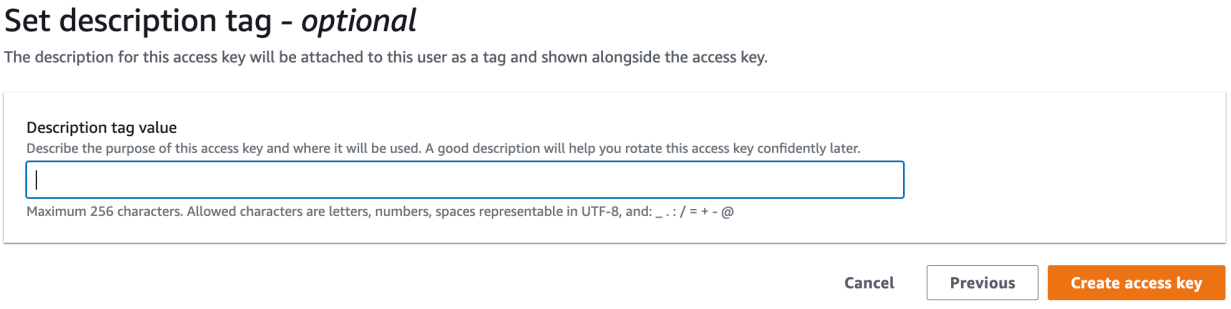
2. Scroll down on the page and find **Access keys** and select it.



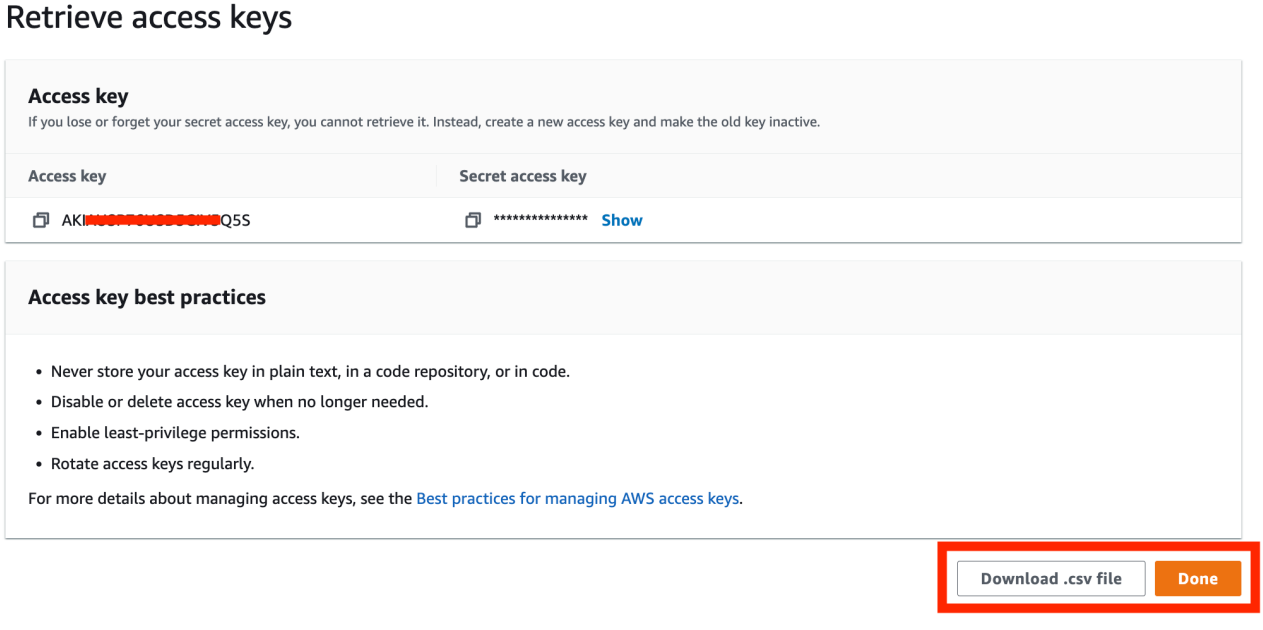
3. You can see AWS suggests alternative methods for accessing resources at the bottom of the page, but in this example we will choose **Command Line Interface (CLI)** and select **Next**.



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



AWS IAM Roles

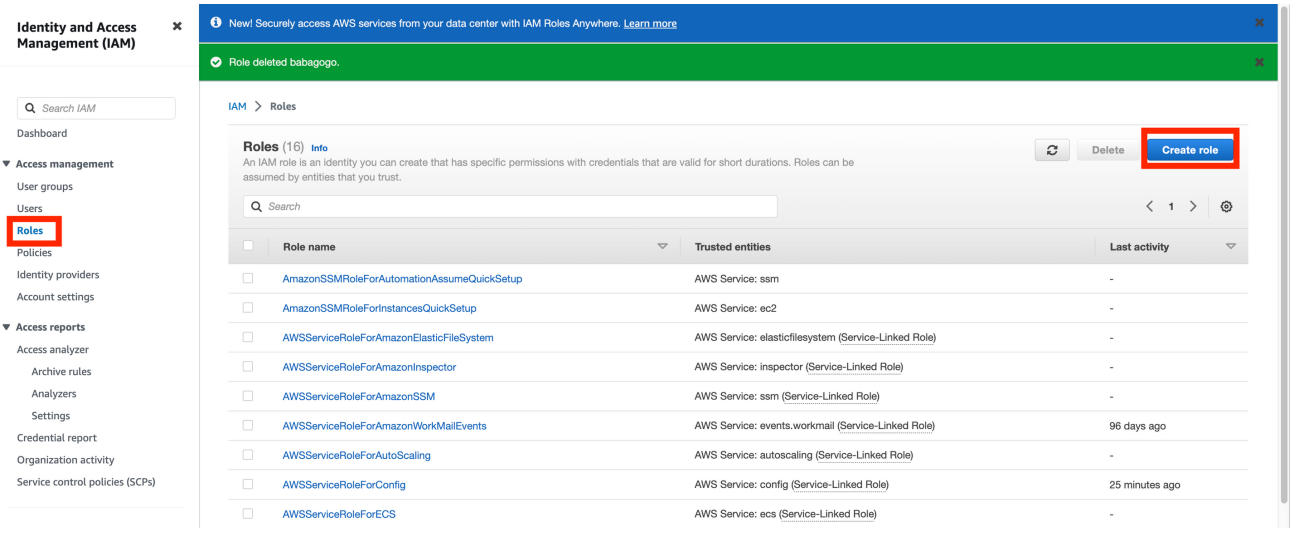
An IAM role is an IAM identity that has specific permissions tied to it. Roles are assumed by users, applications, and services. Once assumed, the identity becomes the role to gain the role's permissions. This eliminates the need to assign an individual all the permissions for the task and

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

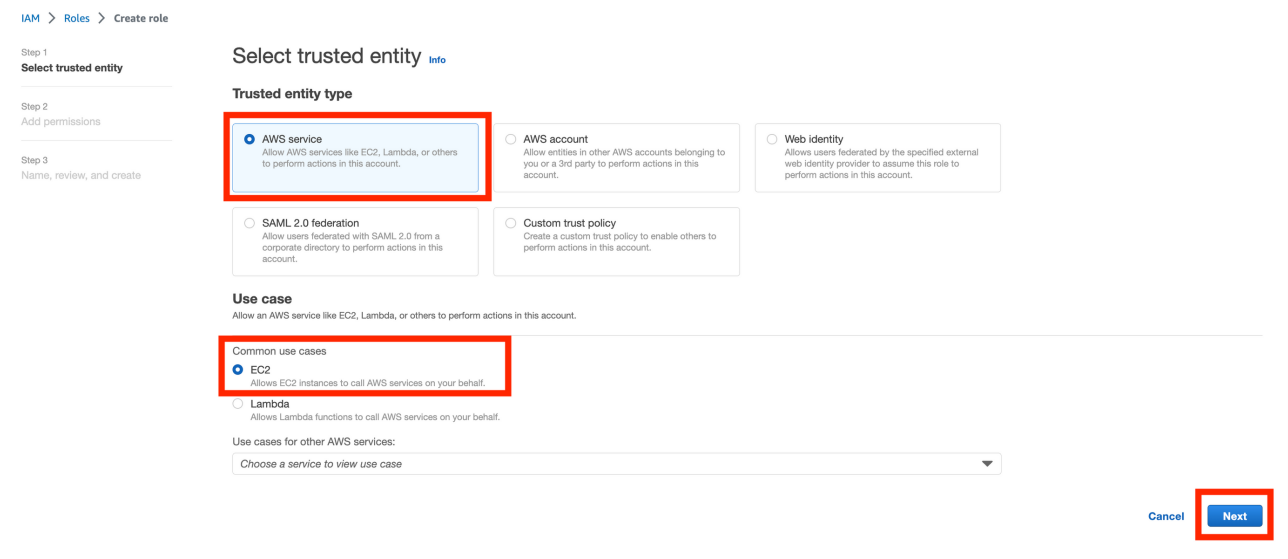
- Helpful to give temporary permissions for a specific task
- Allow a user/application to perform many actions in a different account
- Permissions expire over time
- Common roles: EC2 Instance Roles
- Common roles: Lambda Function Roles
- Common roles: Roles for CloudFormation

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!

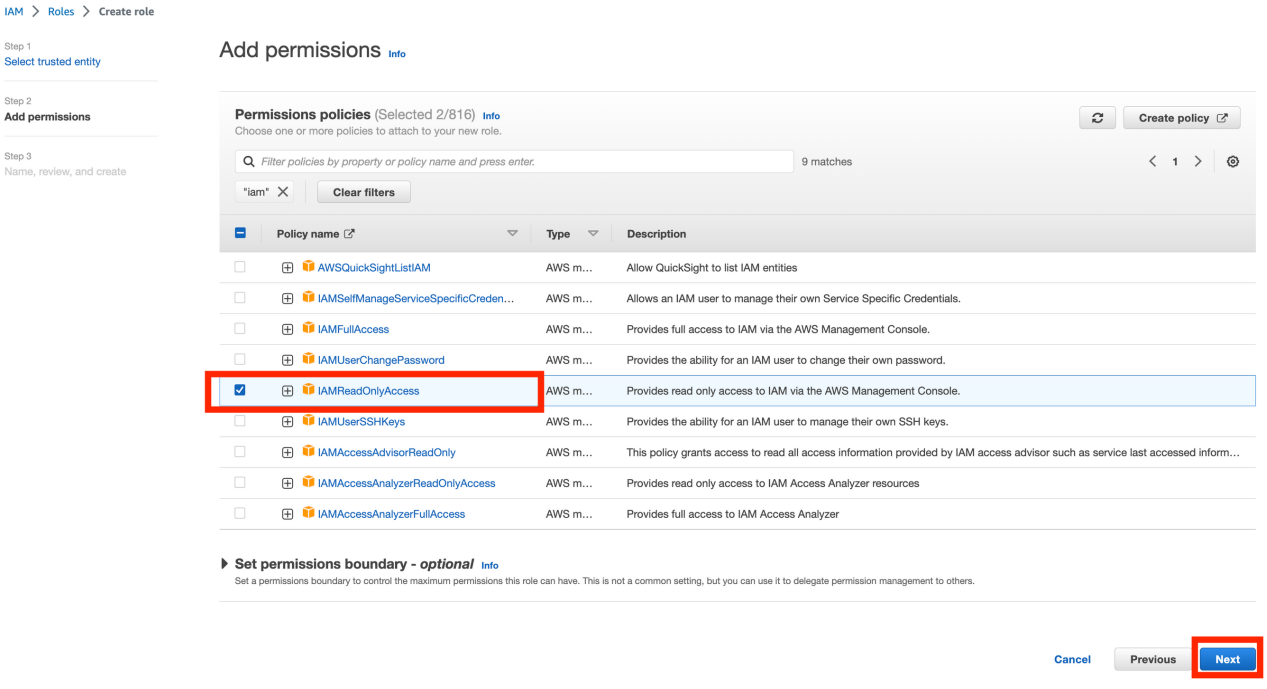
1. Let's browse to **Roles** in IAM and **Create role**



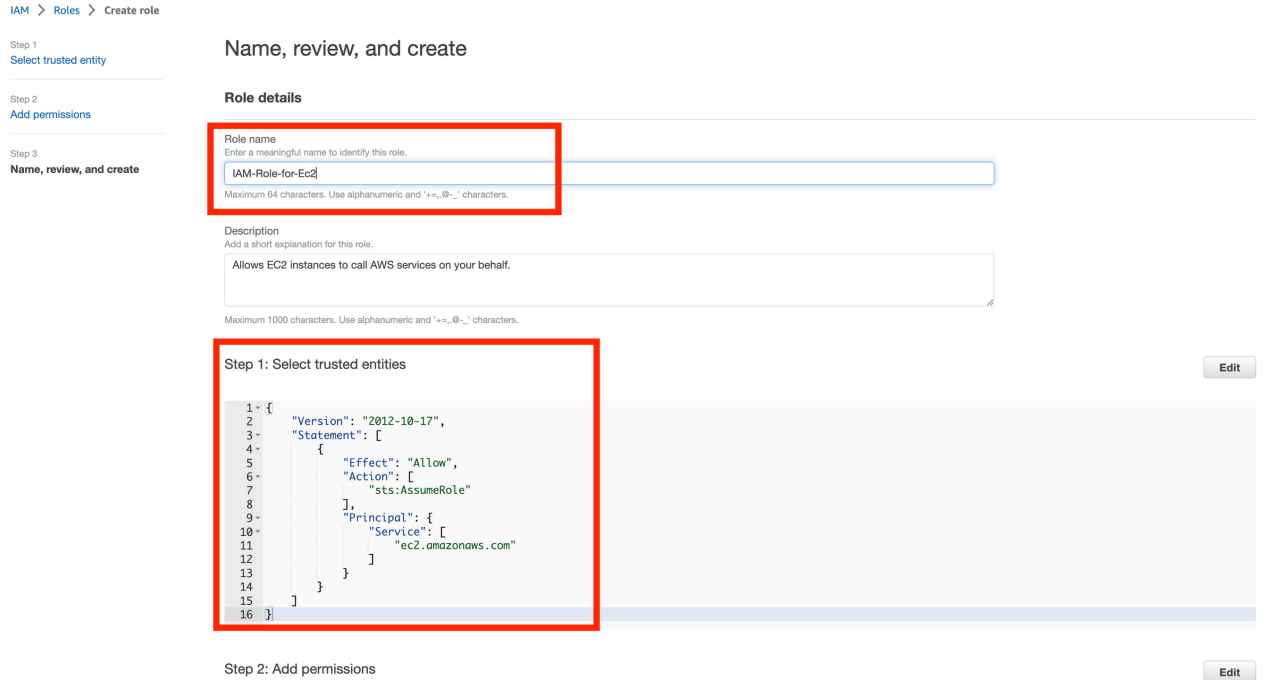
2. Out next screen shows the options we have available. We will choose **AWS service** and **EC2**.



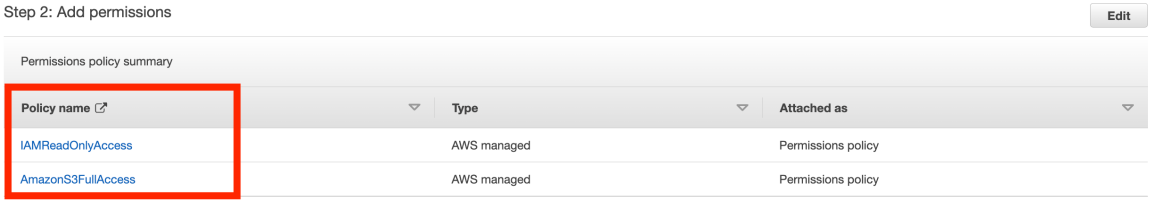
3. We will grant multiple permissions to this role.

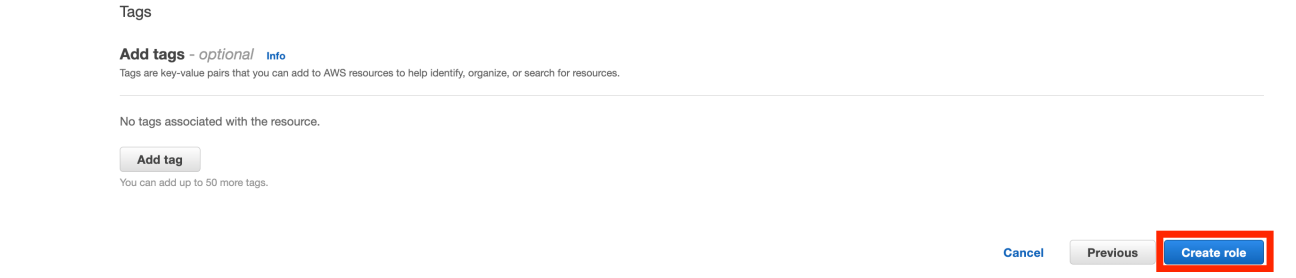


4. Give a name to the new role.

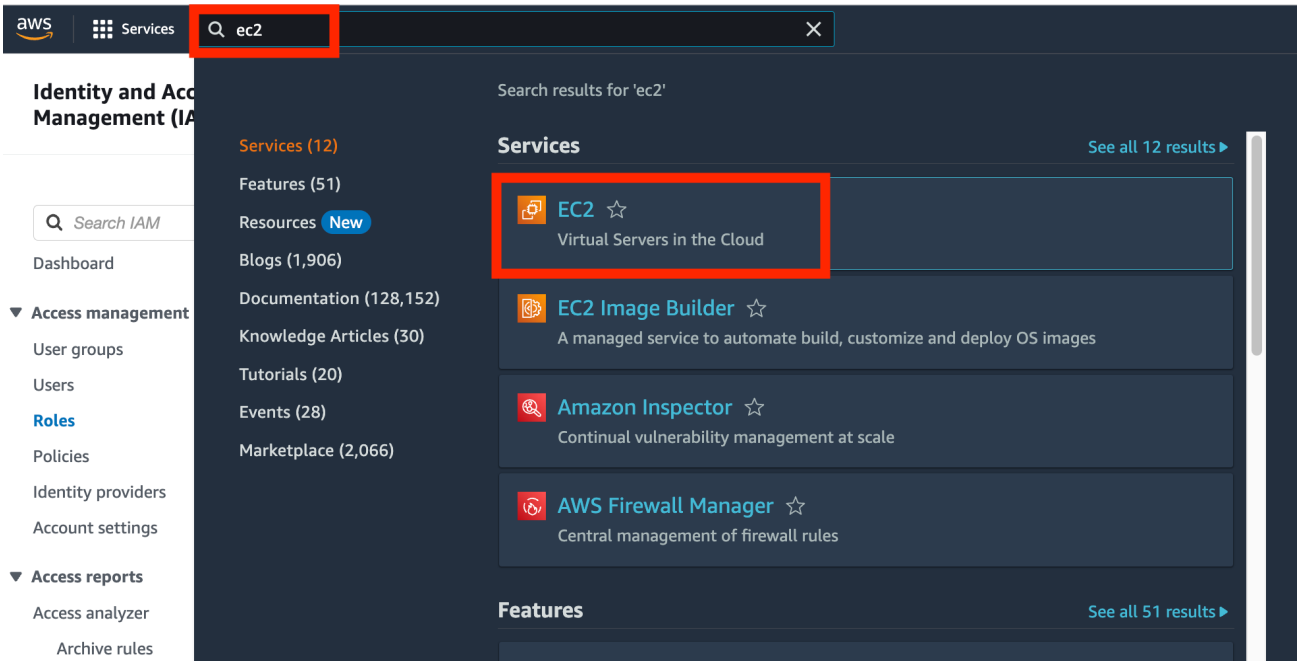


5. Here we can see the 2 permission sets we provided to the Role. Now we will **Create rule** and then move to EC2 to attach the role to an instance.

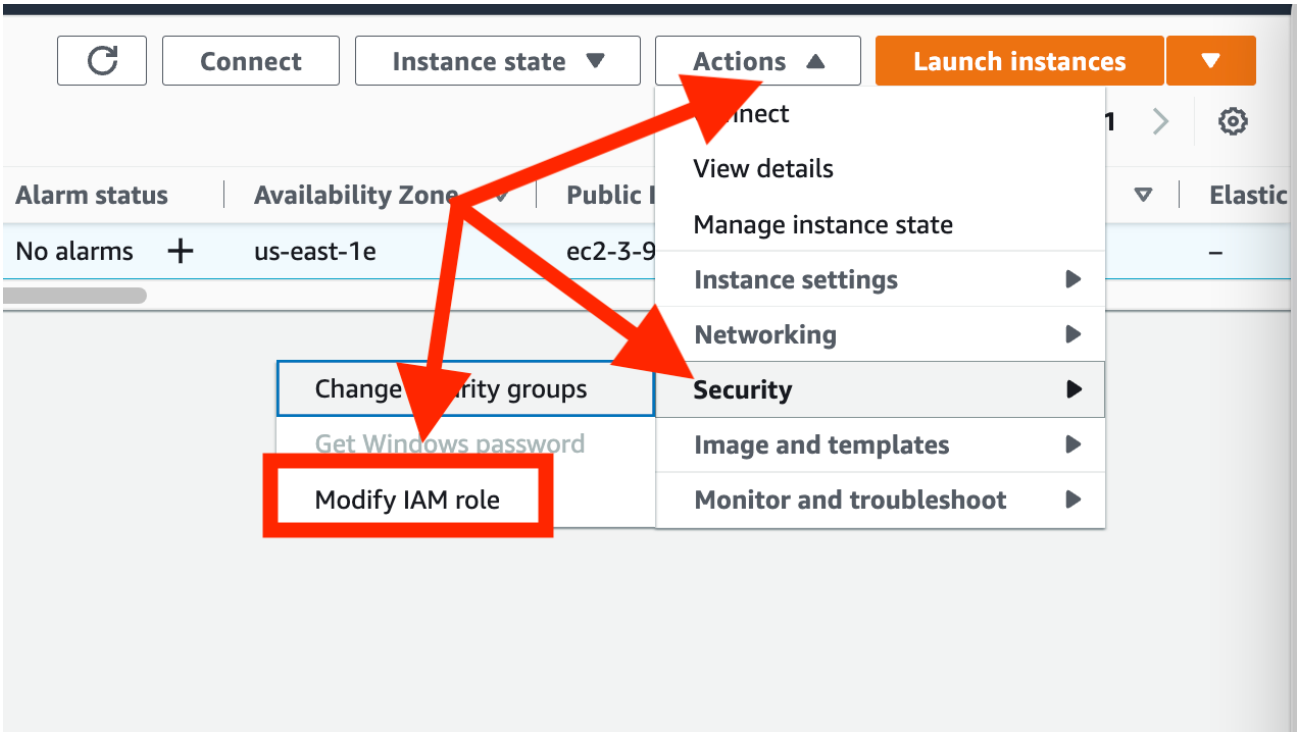




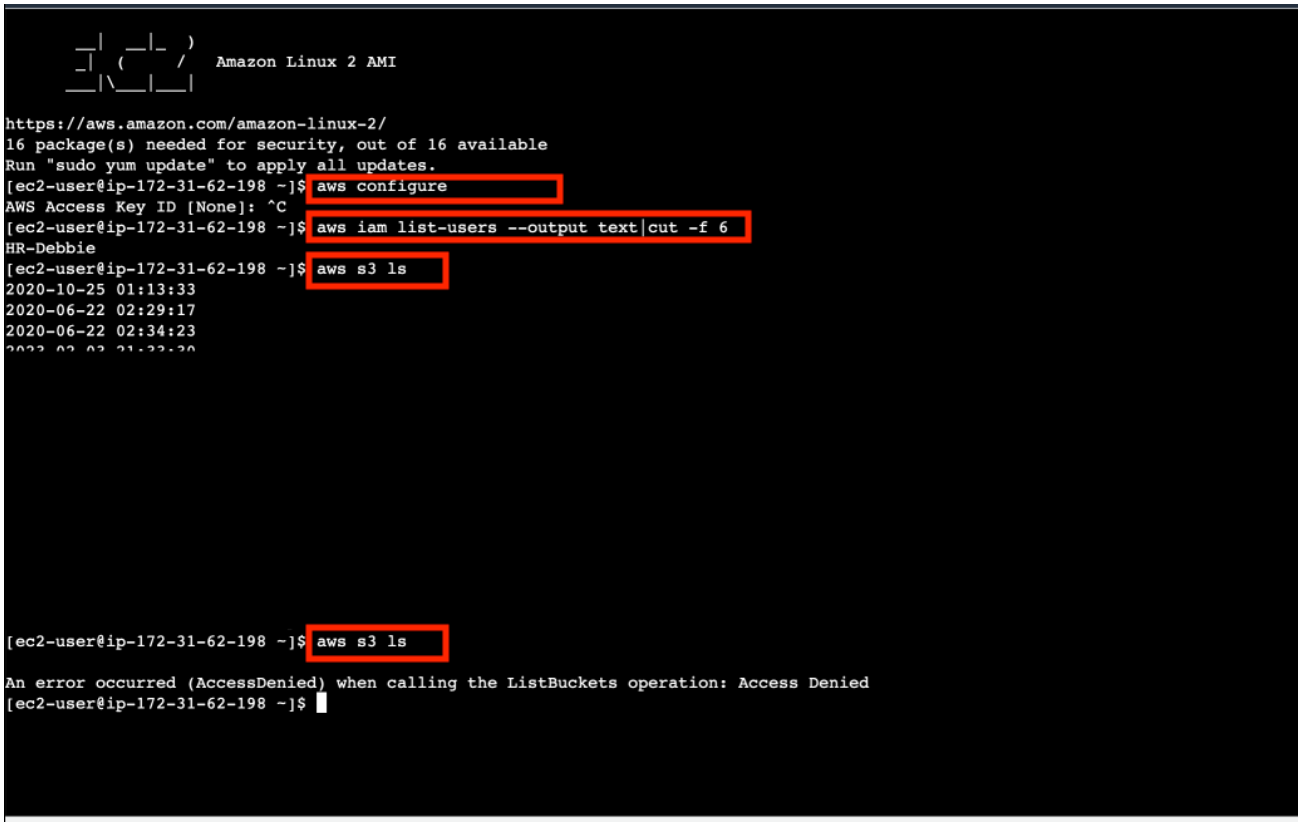
6. Browse to EC2



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 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.



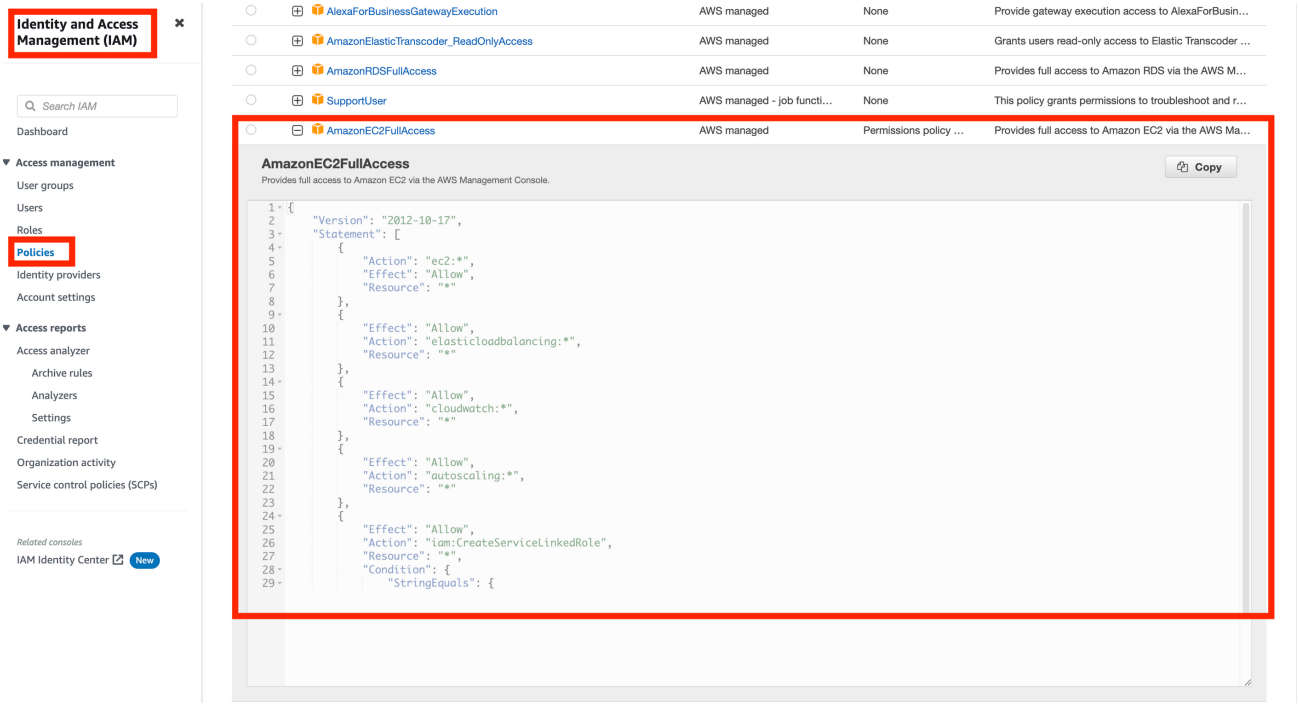
AWS IAM Policies

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.

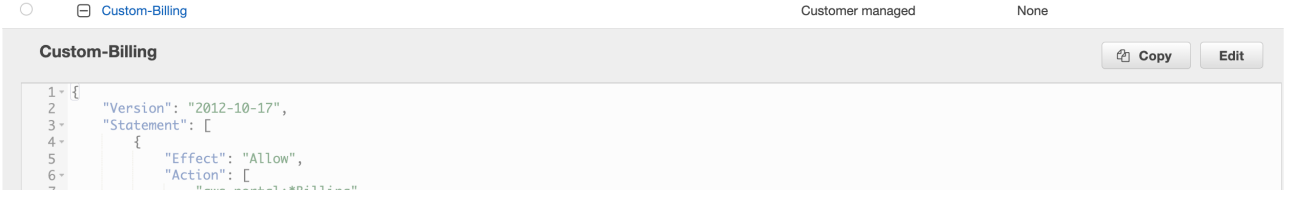
2. **Customer Managed Policies** - these policies are created and maintained by customer.
   a. You define what you would like to allow and disallow.
   b. I will often copy an AWS Managed Policy and modify it for my own needs.

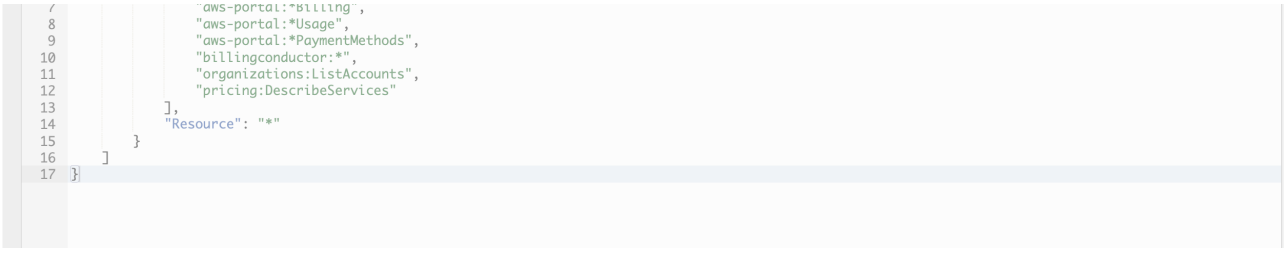
1. In order to access the policies, go in to **IAM** and select **Policies** from the left hand menu as highlighted below. I've highlighted the **AmazonEC2FullAccess** policy - many additional policies are available here for use. The majority are **AWS Managed Policies**, unless you have created a lot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



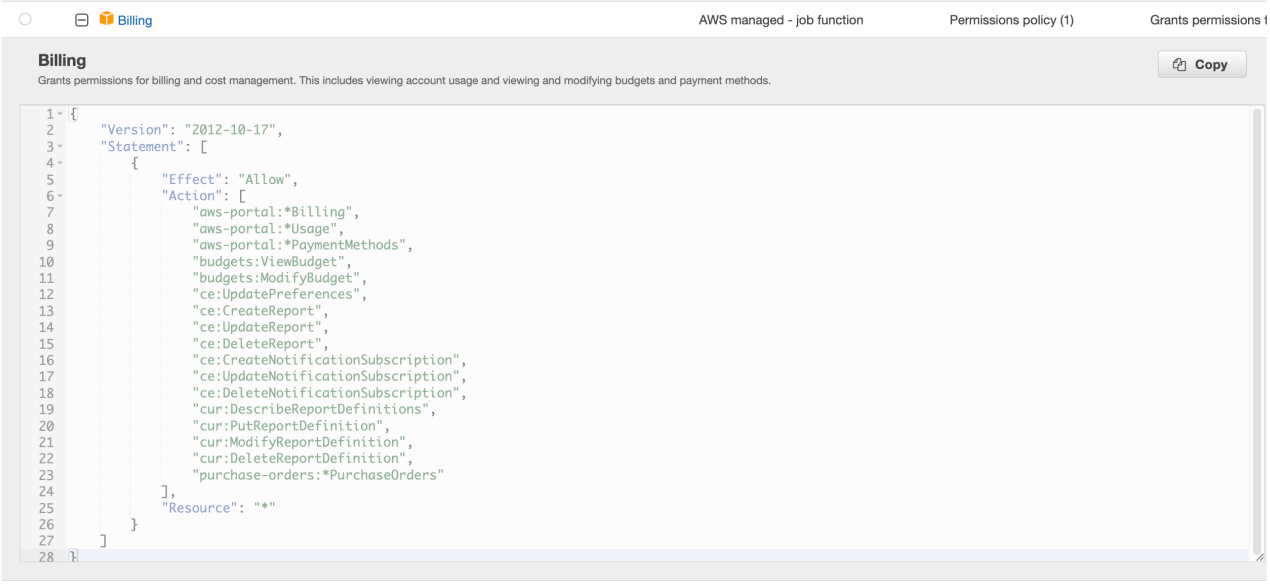
2. Below I share 2 different policies. The first is a customer managed policy for billing and the 2nd example is an **AWS managed policy** which has some different actions defined from my custom policy. Let's go over some of the components in this JSON policy:

- **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)





```
            "aws-portal:*Billing",
            "aws-portal:*Usage",
            "aws-portal:*PaymentMethods",
            "billingconductor:*",
            "organizations:ListAccounts",
            "pricing:DescribeServices"
        ],
        "Resource": "*"
    }
]
}
```



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

Billing

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

Copy

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "aws-portal:*Billing",
                "aws-portal:*Usage",
                "aws-portal:*PaymentMethods",
                "budgets:ViewBudget",
                "budgets:ModifyBudget",
                "ce:UpdatePreferences",
                "ce:CreateReport",
                "ce:UpdateReport",
                "ce:DeleteReport",
                "ce:CreateNotificationSubscription",
                "ce:UpdateNotificationSubscription",
                "ce:DeleteNotificationSubscription",
                "cur:DescribeReportDefinitions",
                "cur:PutReportDefinition",
                "cur:ModifyReportDefinition",
                "cur:DeleteReportDefinition",
                "purchase-orders:*PurchaseOrders"
            ],
            "Resource": "*"
        }
    ]
}
```

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
```

```
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

AWS IAM Policy Evaluation:
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

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