**IAM in AWS**

IAM stands for identity and access management, In AWS it is security services used to manage/controls authentication and authorization (permissions) for the users & groups over the AWS resources.

Or

**IAM (Identity and Access Management) in** **AWS (Amazon Web Services)** is a service that helps you **securely control access** to AWS resources. It allows you to **manage users, groups, roles, policies and permissions** to control who is authenticated and authorized to use resources.

IAM is a **free service** that enables **secure access management** to your AWS services and resources. With IAM, you can:

* Create and manage AWS users and groups.
* Grant or deny permissions to AWS services and resources.
* Use roles for temporary access and delegation.

IAM is primarily used for:

1. **Authentication**: Identifying who is trying to access (users, applications, or services).
2. **Authorization**: Determining what actions they are allowed to perform (like EC2: Start Instances).
3. **Fine-grained access control**: Assigning minimal privileges needed to perform tasks.
4. **Delegation of access**: Granting temporary access via roles (e.g., for applications or third parties).

**Key Features of IAM**

| **Feature** | **Description** |
| --- | --- |
| **Granular Permissions** | Use JSON policies to specify detailed permissions. |
| **Users and Groups** | Organize and manage access control for individuals and teams. |
| **Roles** | Allow services and users to assume temporary access. |
| **Temporary Credentials** | Enable secure short-term access using tokens. |
| **Multi-Factor Authentication (MFA)** | Add an extra layer of security for user logins. |
| **Access Analyzer** | Helps identify resources shared with external entities. |
| **Service-linked Roles** | Automatically created roles for specific AWS services. |
| **Audit and Compliance** | Integrates with CloudTrail for activity logging. |

**Difference between the Policies and Roles:**

In AWS Identity and Access Management (IAM), both **policies** and **roles** are crucial for managing permissions, but they serve distinct purposes and operate in different ways.

Think of it with an analogy:

* **Policies are like the "rulebook" or "list of permissions."** They define what actions are allowed or denied on which resources.
* **Roles are like a "temporary identity" or a "hat" you can wear.** When you wear that hat (assume the role), you gain all the permissions listed in the rulebooks (policies) attached to that hat.

**IAM Roles:**

* It is an IAM identity in AWS that you can assume to gain temporary access to AWS resources. It has policies attached to define what it can do.
* Roles are used to delegate (grant or assign) access to users, groups, applications, or services by using temporary credentials (passwords or Tokens) instead of persistent (permanent) ones.
* An IAM identity that you can create in your account with specific permissions. Unlike an IAM user, a role does not have standard long-term credentials (password or access keys) associated with it.

**Use case:**

**AWS service accessing other AWS service:** An EC2 instance needing to write to an S3 bucket, so you attach a role to the EC2 instance. (Allowing an EC2 instance to access an S3 bucket).

**Cross-account access**: Allowing users or applications (group) in one AWS account to access resources in another AWS account.

**Note: STS (Secure Token Service):**

In AWS, Security Token Service. It is a service that enables you to request temporary security credentials for AWS Identity and Access Management (IAM) roles or users.

**IAM policies:**

A policy is a JSON document that defines permissions, specifying what actions are allowed or denied on which AWS resources for a user, group, or role.

* **Purpose:** To define what actions an entity can perform on *which* AWS resources, and under whatconditions.
* **Content:** A policy includes elements like **Effect** (Allow or Deny), **Action** (e.g., s3:GetObject, ec2:RunInstances**), Resource** (e.g., arn:aws:s3:::my-bucket/\*), and **optional Condition** (e.g., allow only from a specific IP address).
* **Attachment:** Policies are attached to IAM identities (users, groups, and roles) or directly to AWS resources (resource-based policies, like S3 bucket policies).

Types of IAM policies:

* 1. AWS Managed policies
  2. Custom managed policies.
  3. Inline policies.
  4. Resource based policies.

1. **AWS Managed Policies:**

Predefined policies created and managed by AWS for common use cases (e.g., AmazonS3ReadOnlyAccess).

1. **Custom managed policies:**

Custom policies you create and manage to fit your specific needs.

1. **Inline policies:**

Policies embedded directly within a single user, group, or role. They are deleted when the entity is deleted.

1. **Resource based policies:**

Attached directly to resources (e.g., S3 buckets, SQS queues, KMS keys). They define who can access that specific resource*.*

JSON file start with flower (“{“) brackets and end with flower (“}”) brackets as shown below, and to define multiple polices/values within it we use square brackets **“[ ]”,** and then we use “**,**” for separation. The content which is present in JSON file are in **key-values** pair format.

**Example:**

Suppose you want to give a user **read-only access to a specific S3 bucket** named my-company-data.

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"s3:GetObject",

"s3:ListBucket"

],

"Resource": [

"arn:aws:s3:::my-company-data",

"arn:aws:s3:::my-company-data/\*"

]

}

]

}

Fig: Policy JSON file.

**Explanation**:

* **Effect:** Allow: Grants permission.
* **Action:** Specifies what actions are allowed (get/list objects).
* **Resource:** Limits access to the **my-company-data** bucket and its contents.

Now let’s perform a task practically using IAM.

**Process:** Create Two EC2 instance (instance-01 & instance-02). And create two users (user-01 & user-02). Now write a policies (EC2policy-01) in such a way that user-01 is granted all permission over the instance-01, but not on the instance-02.

Similarly write a policy (EC2policy-02) in such a way that user-02 is granted all permissions over the instance-02 but not on the instance-01.

Then attach EC2policy-01 to the user-01 & EC2policy-02 to the user-02.

Step1: Create two EC2 instances.

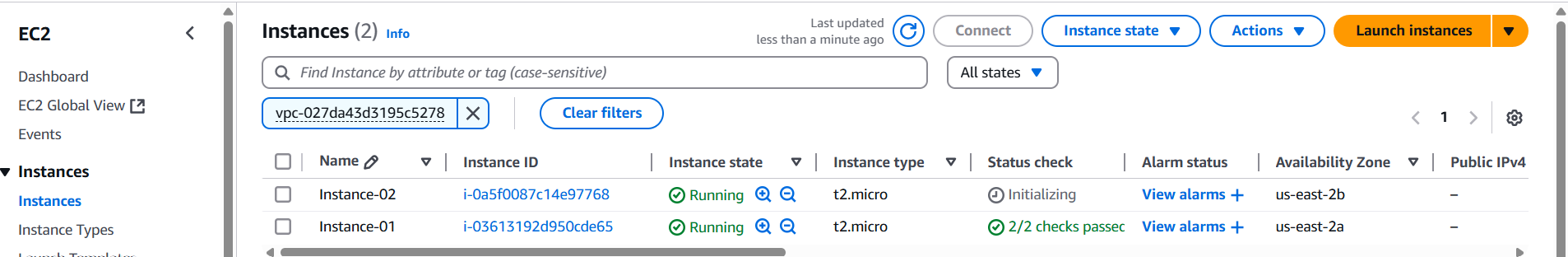


Fig: Two instances (Instance-01 & Instance-02) are created successfully.

**Note:** Add the tags for both the instances.

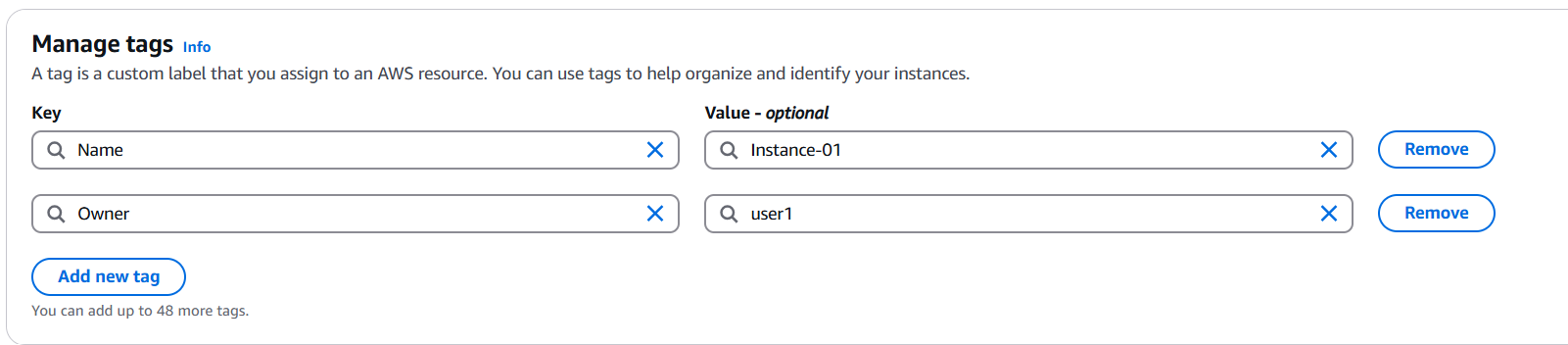


Fig: Instance-01 tag.

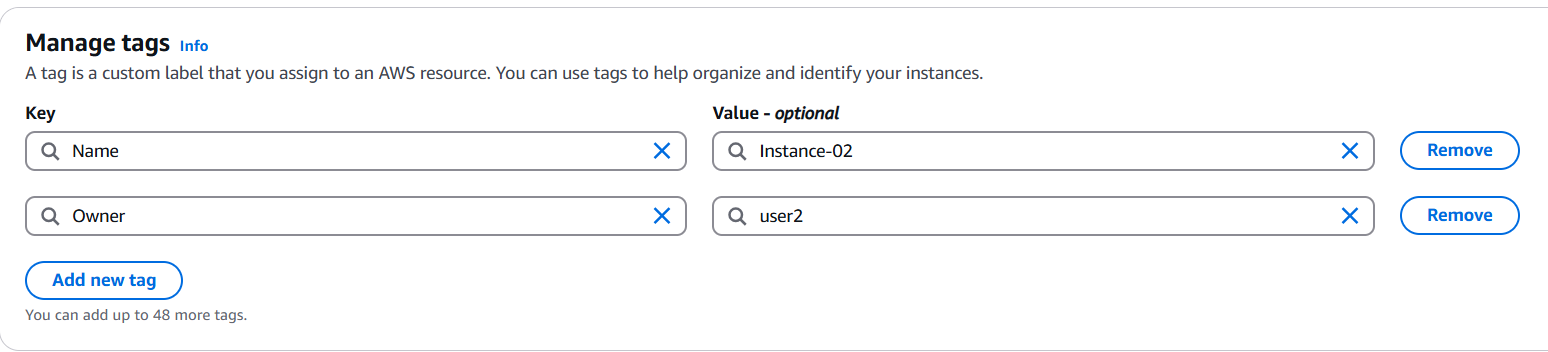


Fig: Instnace-02 tag.

Step2: Create two users (user-01 & user-02).

Go To 🡪 IAM 🡪users🡪 Create user.

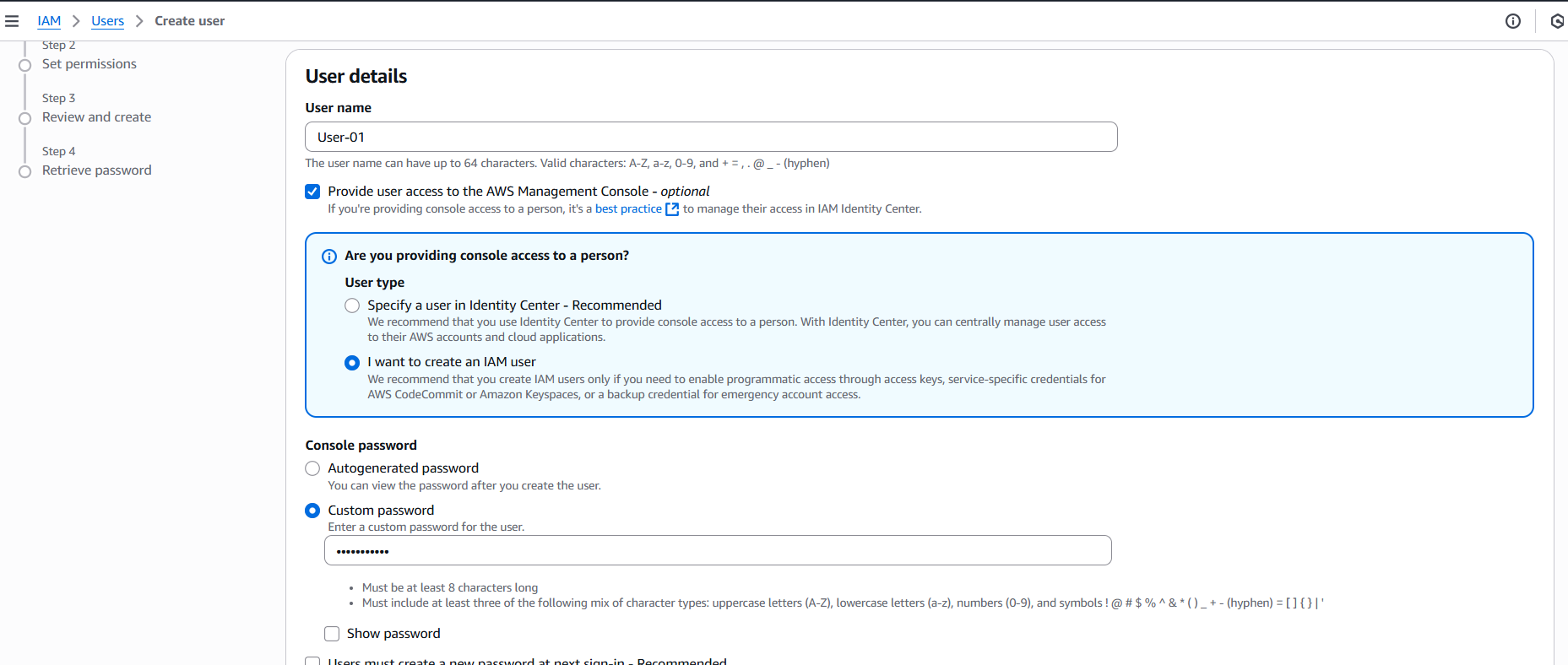


Fig: Create a user and defining a password.

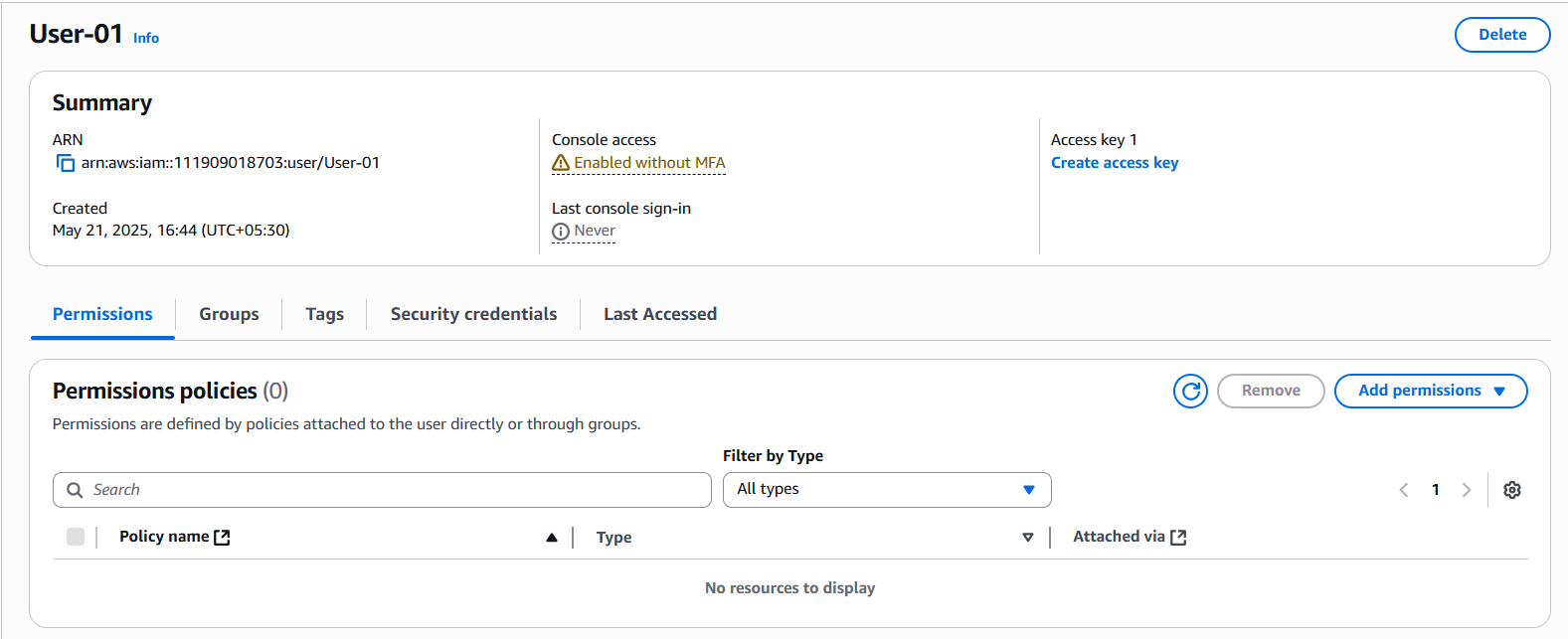


Fig: User (user1) is created successfully without attaching any policies.

Similarly create another user (user2) using IAM.

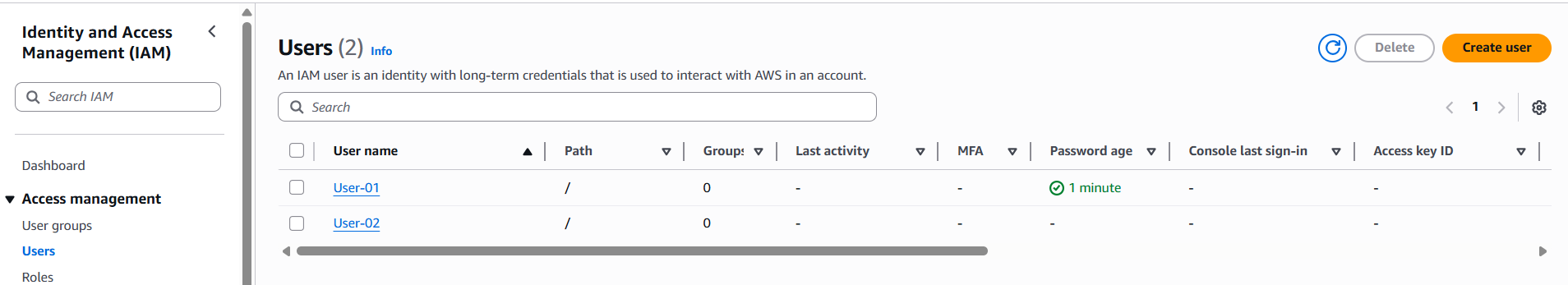


Fig: Two users (user1 & user2) are created successfully.

**Step3**: Write the policies for both the user.

Such that user1 access instance-01but not the Instances-02.

And user2 access instance-02 but not the instances-01.

Go To 🡪 IAM 🡪policies🡪 Create policy.

{

    "Version": "2012-10-17",

    "Statement": [

        {

            "Effect": "Allow",

            "Action": "ec2:Describe\*",

            "Resource": "\*"

        },

        {

            "Effect": "Allow",

            "Action": [

                "ec2:StartInstances",

                "ec2:StopInstances",

                "ec2:RebootInstances"

            ],

            "Resource": [

                "arn:aws:ec2:\*:111909018703:instance/\*"

            ],

            "Condition": {

                "StringEquals": {

                    "ec2:ResourceTag/Owner": "user1"

                }

            }

        }

    ]

}

Fig: IAM policy for User-01.

AWS account ID: 111909018703

Instance-01 tag: "user1"

This IAM policy grants permissions to **start, stop, and reboot** EC2 instances. However, these actions are only allowed on instances that have a specific tag with the **“user1”.**

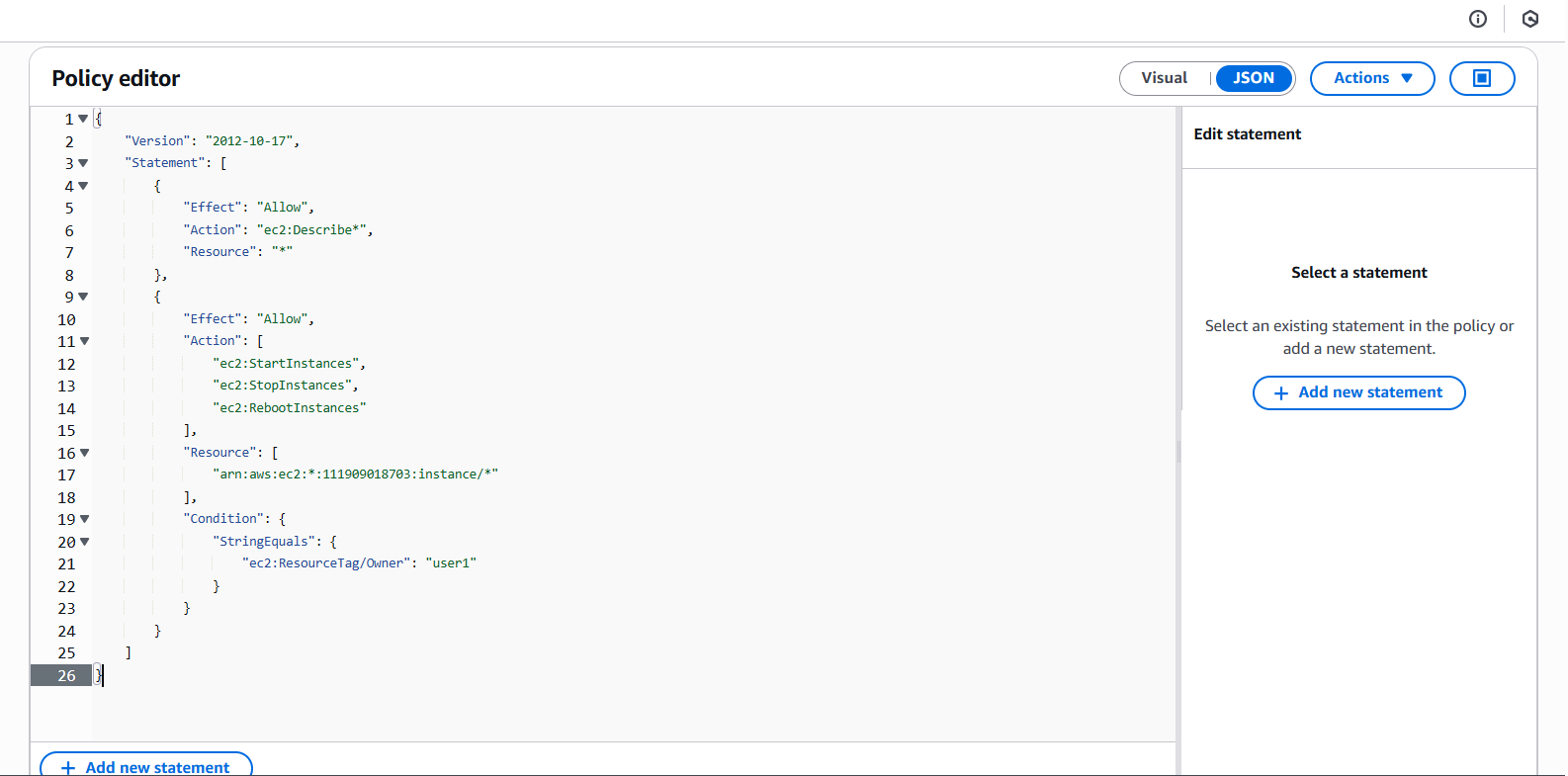


Fig: JSON code is upload.

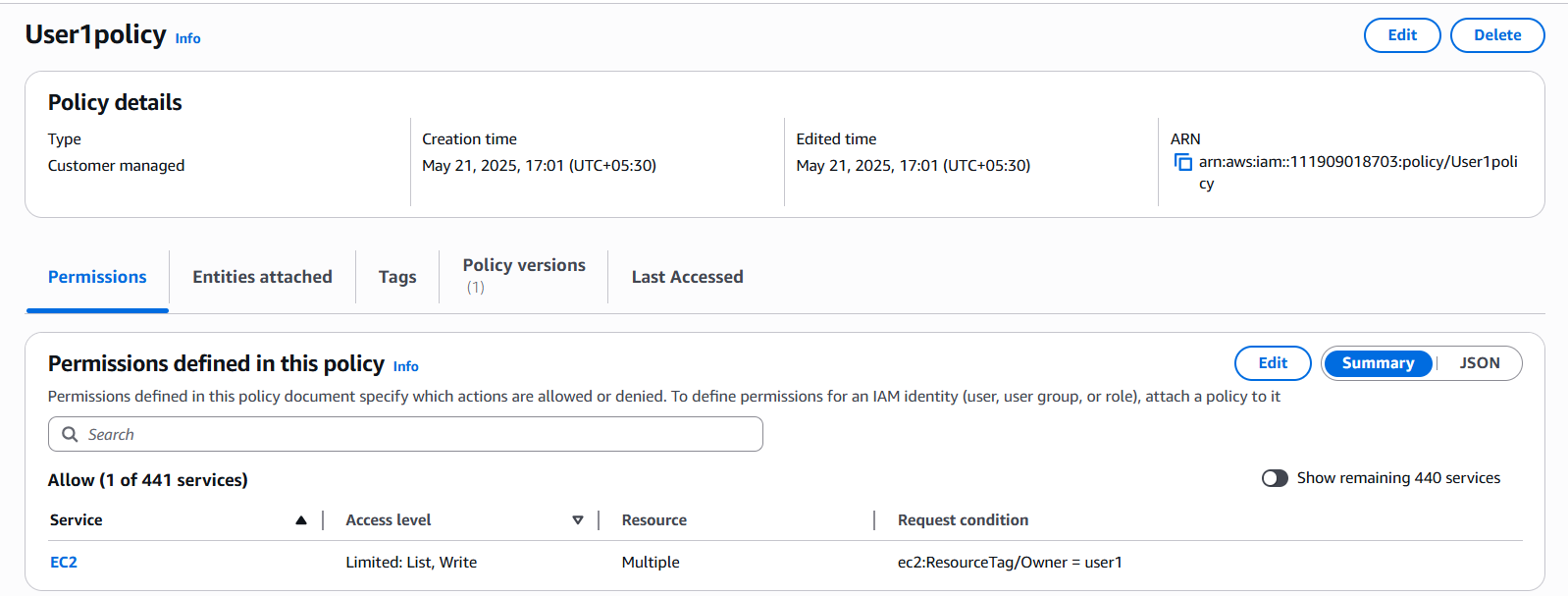


Fig: User1policy is created successfully.

Similarly create another policy (User2policy).

{

    "Version": "2012-10-17",

    "Statement": [

        {

            "Effect": "Allow",

            "Action": "ec2:Describe\*",

            "Resource": "\*"

        },

        {

            "Effect": "Allow",

            "Action": [

                "ec2:StartInstances",

                "ec2:StopInstances",

                "ec2:RebootInstances"

            ],

            "Resource": [

                "arn:aws:ec2:\*:111909018703:instance/\*"

            ],

            "Condition": {

                "StringEquals": {

                    "ec2:ResourceTag/Owner": "user2"

                }

            }

        }

    ]

}

Fig: Policy for User-02.

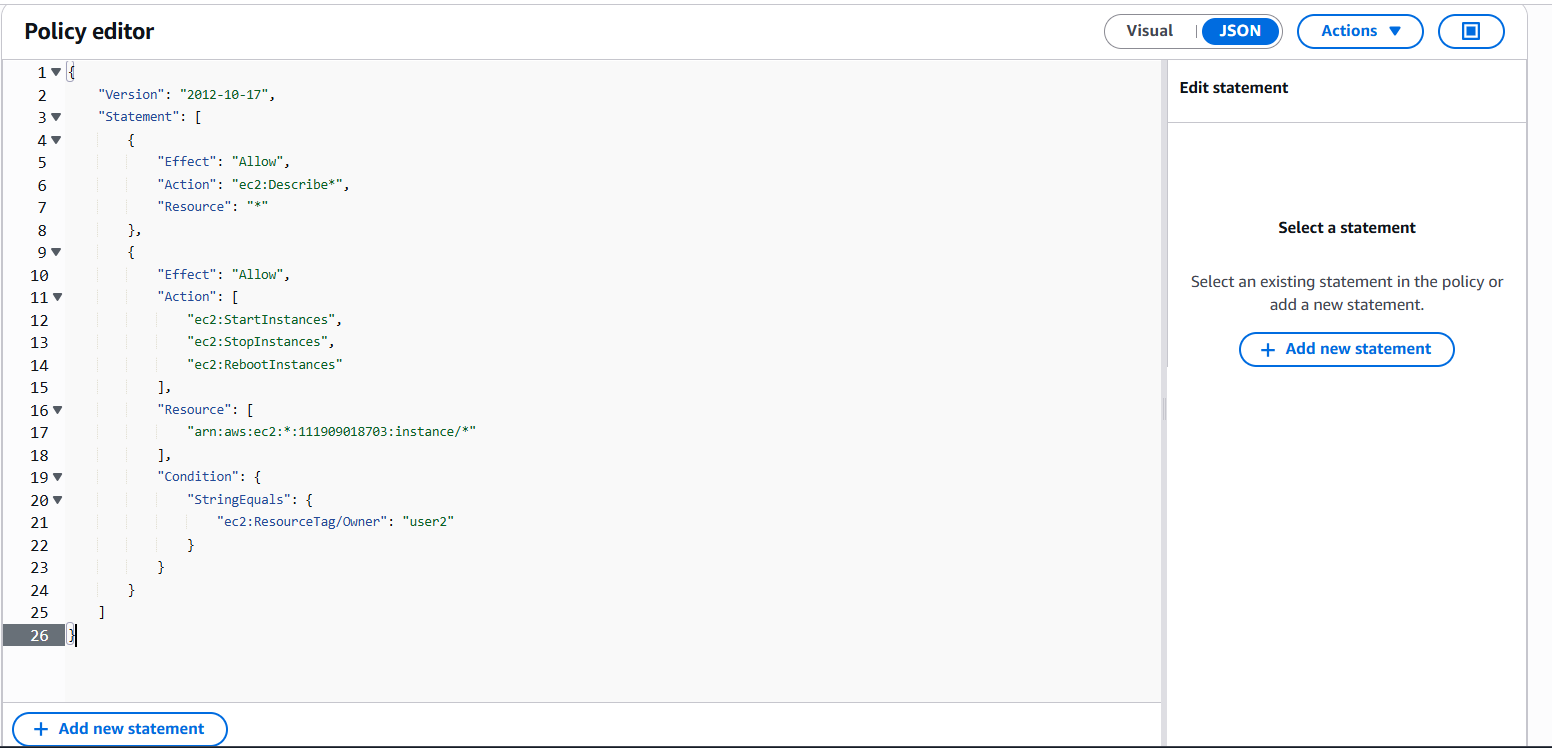


Fig: Policy is upload.

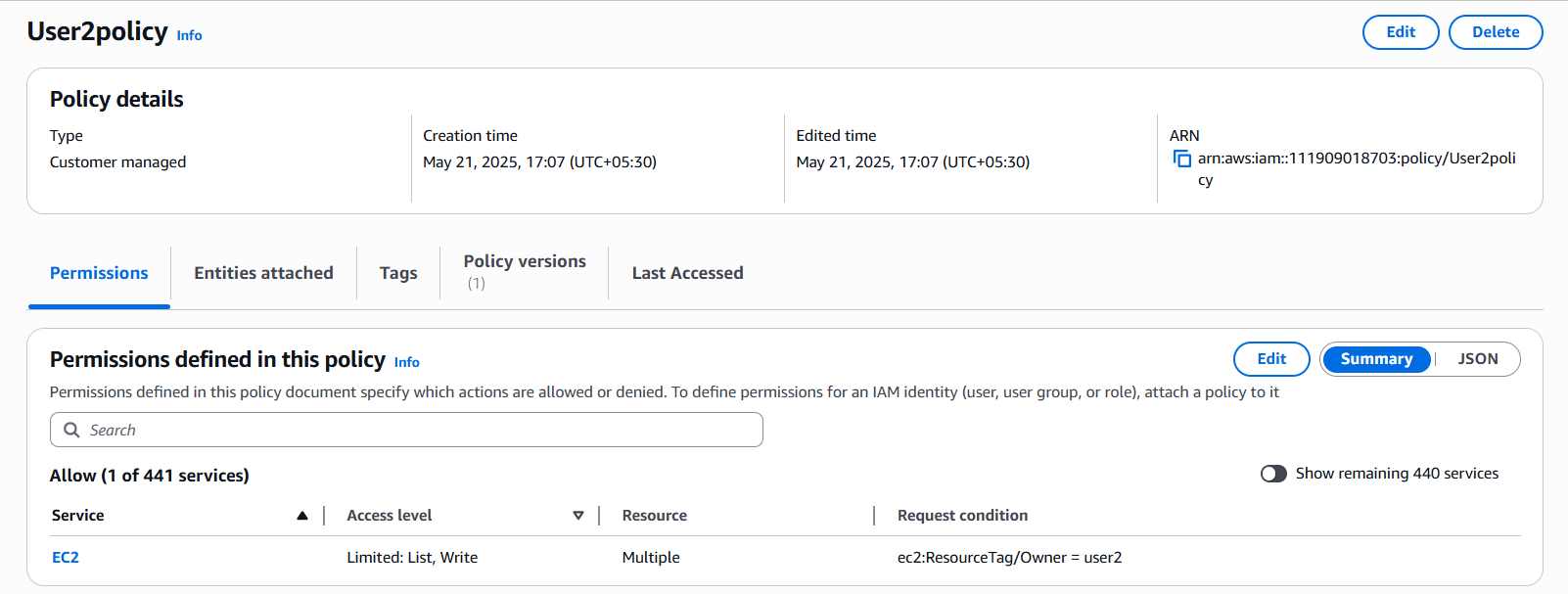


Fig: Policy (User2policy) created successfully.

Step4: Now attach the policies for respected users.

User1policy ===🡺User-01

User2policy ===🡺User-02.

Go To 🡪User-01🡪Add permission 🡪Attach polices directly🡪Search for our custom policy🡪And add it.

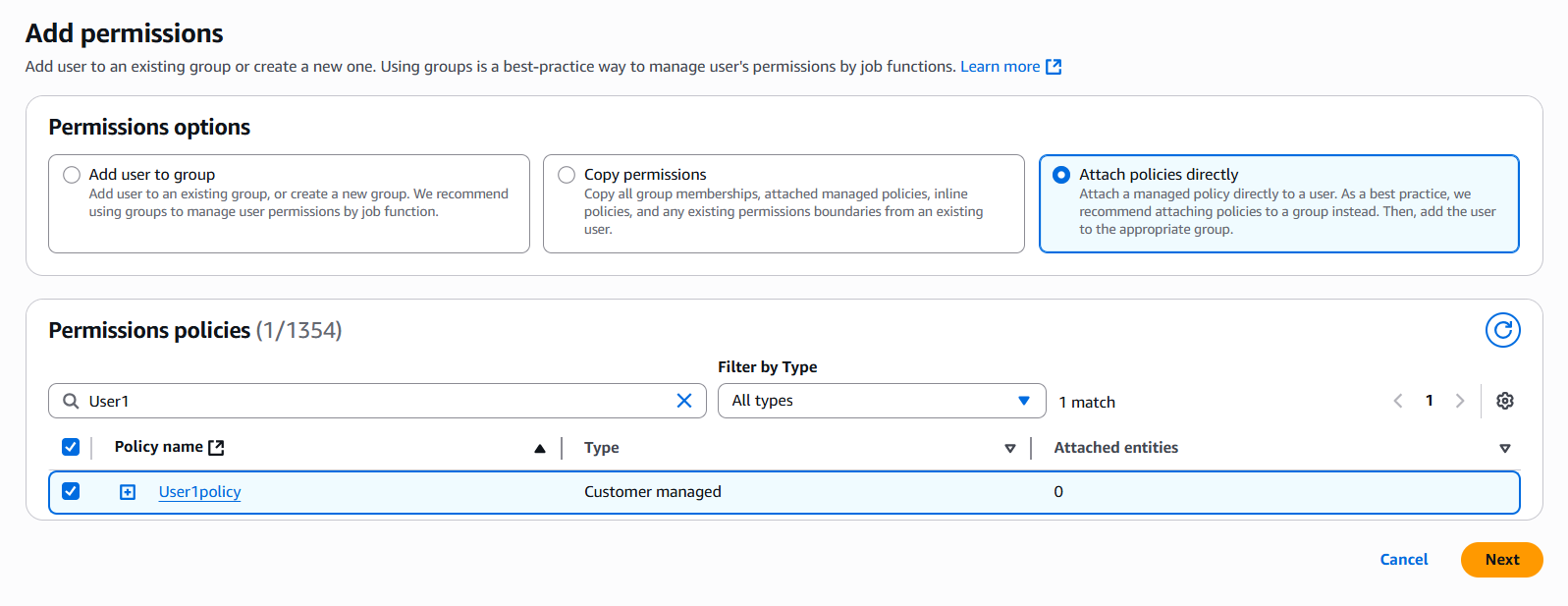


Fig: Adding policy (User1policy) to the user (User-01).

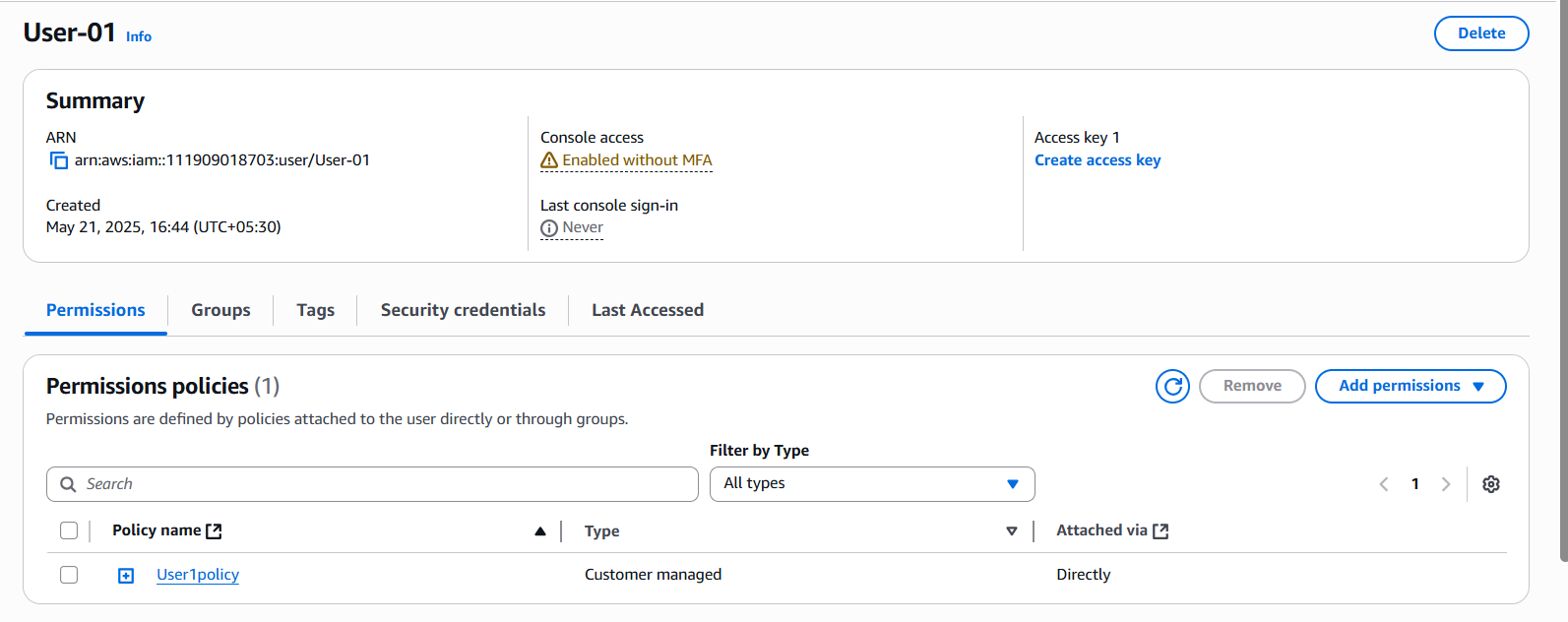
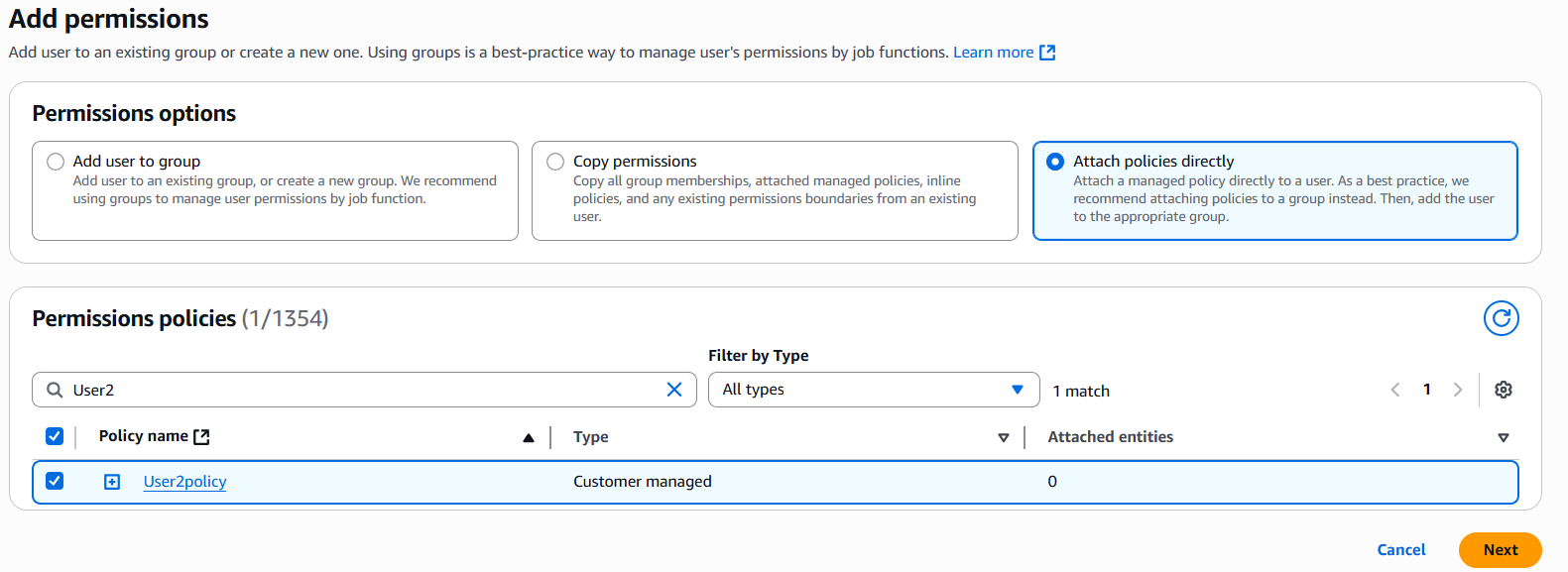
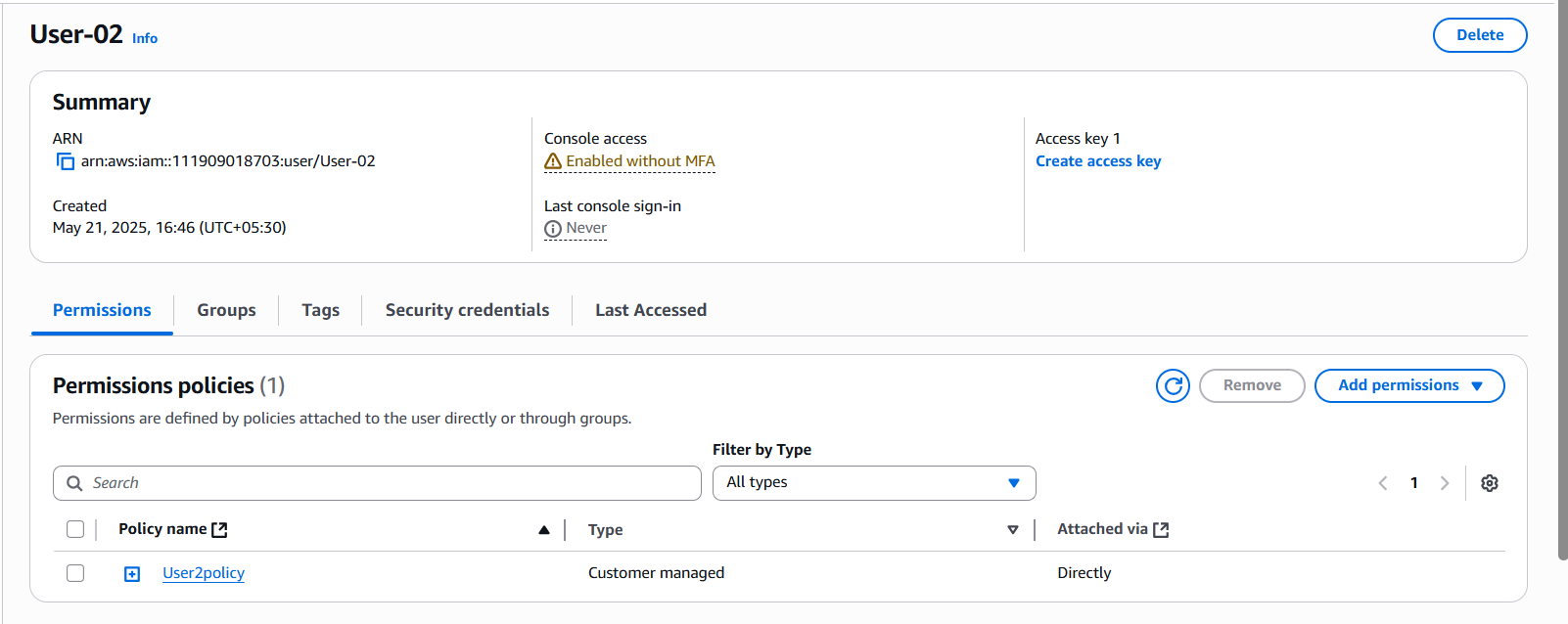


Fig: User1policy attached successfully with User-01.

Now similarly attach the policy (User2policy) to the User-02.





Step5: Now Login to the AWS portal/console using any of IAM user credentials.

I am login to the AWS portal using User-01 credentials.

Account ID: 111909018703

IAM username: User-01

Password: Harish@1000

Now try to stop or start or reboot the inatance-02. While stopping the instance-02 we will get an error message, that failed to stop the instance as shown in below figure, it because the user-01 have no permission to stop or start or reboot the instance-02.

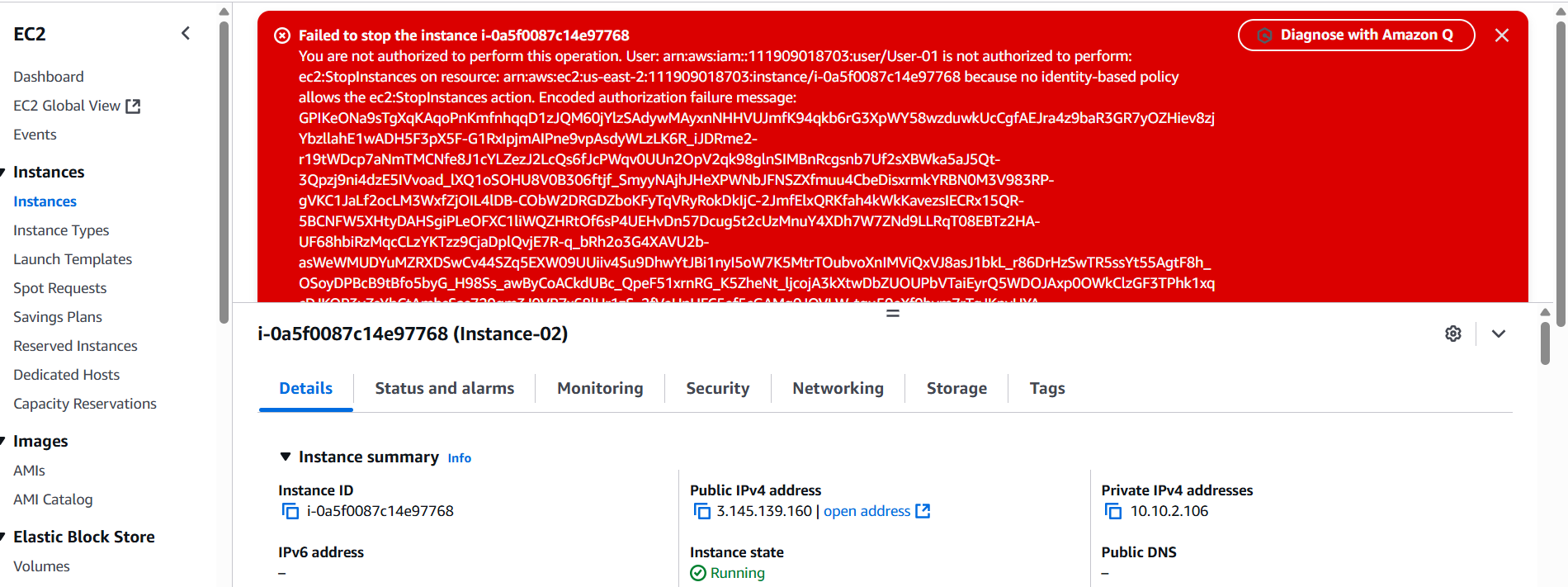
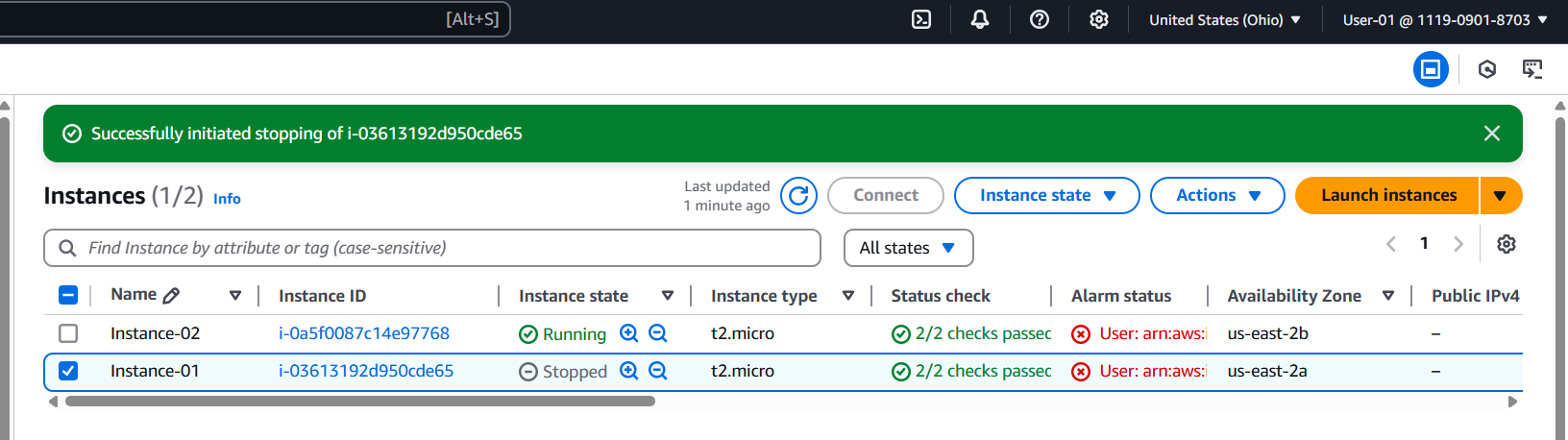


Fig: Unable to stop the instance-02.

But the User-01 can stop or start or reboot the instance-01, as he has permissions over it, as show below figure.



Similarly by login to the AWS portal using User-02 credential we will get same error message while stopping or starting or rebooting the instance-01.

Some of the IAM policies related to the S3 bucket:

1. Full access to the specific S3 bucket.

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": "s3:\*",

"Resource": [

"arn:aws:s3:::example-bucket",

"arn:aws:s3:::example-bucket/\*"

]

}

]

}

1. Read-only permission to a specific S3 bucket.

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"s3:GetObject",

"s3:ListBucket"

],

"Resource": [

"arn:aws:s3:::example-bucket",

"arn:aws:s3:::example-bucket/\*"

]

}

]

}

1. Allow upload (Write-only) to a specific S3 bucket.

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": "s3:PutObject",

"Resource": "arn:aws:s3:::example-bucket/\*"

}

]

}

**Permission Boundary:**

* In AWS Identity and Access Management (IAM), a **permissions boundary** is an advanced feature that allows you to set the **maximum permissions** that an IAM entity (a user or a role) can have.
* That means **Permissions Boundaries** are **policies** (similar in structure to regular IAM policies) that act as a **boundary or ceiling** for what a user or role can do, regardless of what permissions are directly assigned to them.
* The most common and powerful use case for permissions boundaries is to **delegate (Assign or grant) administrative tasks** safely.

**Features:**

### 1. ****Restrict Maximum Permissions****

* A permissions boundary sets the **upper limit** of what an IAM user or role can do.
* Even if their identity-based policies grant broader access, they **cannot exceed** the boundary's permissions.

### 2. ****Policy-Based Control****

* It uses a **managed policy** (either AWS-managed or customer-managed) to define the boundary.
* The syntax and structure are the same as regular IAM policies (JSON format).

### 3. ****Both Must Allow****

* For an action to be allowed:
  + The identity policy **must allow it**, and
  + The permissions boundary **must also allow it**.
* If either one **denies** the action, it’s **denied**.

### 4. ****Applicable to IAM Users and Roles****

* Permissions boundaries can be attached to:
  + **IAM users**
  + **IAM roles**
* They are **not applicable to groups** or **resources**.

### 5. ****Useful for Delegated Administration****

* Helpful in **multi-team** or **multi-account** environments.
* Example: Allow an admin to create IAM roles for developers, but restrict them from granting overly broad permissions by applying a boundary.

### 6. ****Not a Standalone Permission Grant****

* A permissions boundary **doesn’t grant permissions by itself**.
* It only **limits** what the identity’s policies can allow.

**Comparison between IAM policies & permission boundary:**

* **JSON Document:** Both IAM policies and permissions boundaries are defined using the same JSON policy language, specifying Effect (Allow/Deny), and Action, Resource, and Condition elements.
* **Managed Policies:** Both can be AWS-managed or customer-managed policies.
* **Attachment:** Both are attached to IAM entities (users or roles).

### Example Scenario1:

* You give a developer full EC2 access in their IAM policy.
* You attach a permissions boundary that only allows read-only EC2 actions.
* Result: The developer will only be able to perform read-only EC2 actions, **even though their IAM policy allows more**, because the boundary limits it.

**Example senario2:**

* There is Senior DevOps engineer and Junior DevOps engineer.
* Senior DevOps engineer is wants to allow/grant all the permissions expect IAM and S3 service.
* Now write a permission boundary to this scenario.

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": "\*",

"Resource": "\*"

},

{

"Effect": "Deny",

"Action": "iam:\*",

"Resource": "\*"

},

{

"Effect": "Deny",

"Action": "s3:\*",

"Resource": "\*"

}

]

}

Fig: Permission boundary.