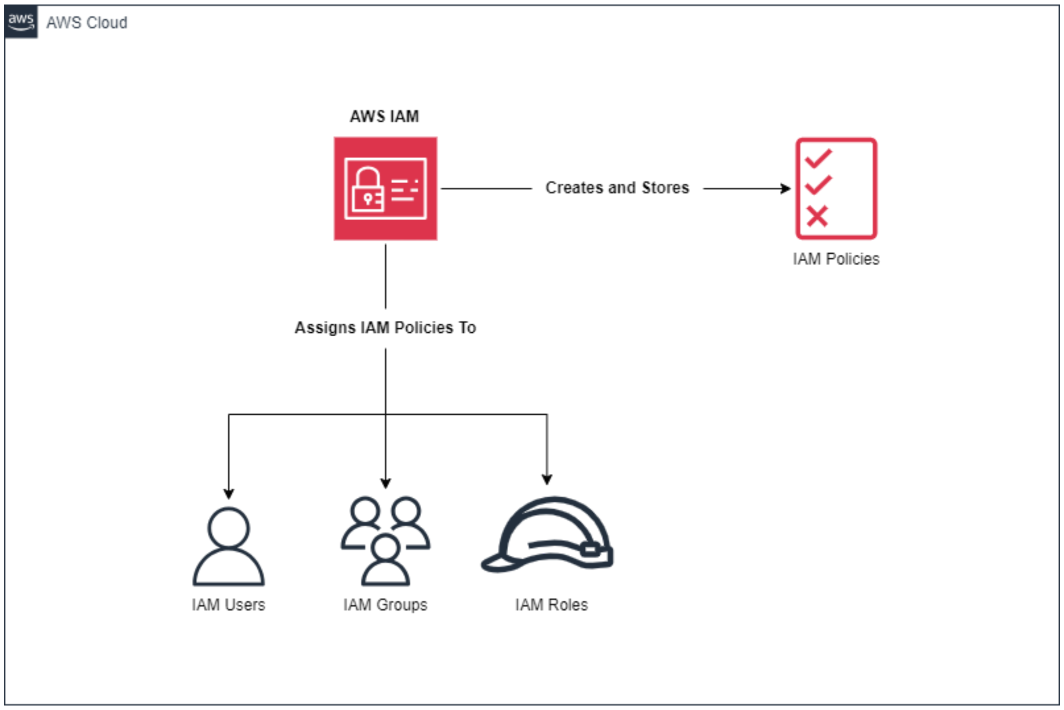
IAM and IAM Roles

The principle of least privilege has already been discussed in the background section, but how exactly do we ensure that the people and teams using our AWS environment are made to follow it? How do we micro-manage permissions for individual users, groups and all the different agents that may be utilizing our AWS resources? The answer is **AWS Identity and Access Management**, or IAM for short.

AWS Identity and Access Management (IAM) is a key service that enhances security and access control within the Amazon Web Services ecosystem. It allows cloud administrators to manage users and groups, setting and assigning fine-grained permissions to them. IAM helps us ensure that only authorized users can perform specific actions, maintaining robust security and compliance.



The versatility of IAM, its wide-ranging use cases and multitude of features and sub-features make it a hard service to summarize properly. But most of what IAM does can be encapsulated as being responsible for **A)** Creating and storing IAM Policies and **B)** Assigning them to different identity types (IAM Users, Groups and Roles). Both factors have been illustrated succinctly in the diagram above.

Now, from that above statement we can craft the following questions, answering which will help us get a more complete understanding of IAM:

- What are IAM Policies?
- What is an identity type, and how do IAM Users, Groups and Roles differ from one another?

IAM Policies

An IAM policy is a document which decides the permissions (i.e. what it can and cannot do) of any AWS resource or IAM identity associated with it. Discussing all the different types of policies is beyond the scope of this book, limiting the discussion to identity-based policies as shown in the diagram.

Identity based policies are documents written in JSON that can assign and control the permissions associated with any IAM User, Group or Role it is associated with. An example of an identity-based IAM Policy that enabled any resource/service it is attached to the ability to access an S3 bucket named `iamBucket`:

```
{
   "Version": "2012-10-17",
   "Statement": {
       "Sid": "ListObjectsInBucket",
       "Effect": "Allow",
       "Action": "s3:ListBucket",
       "Resource": "arn:aws:s3:::iamBucket"
   }
}
```

Note however that it is not necessary for a cloud administrator to write JSON documents every time they wish to assign permissions related to a service. For most widely used tasks, AWS provides us with **Managed Policies**, which are JSON documents created and administered by AWS on our behalf. We can simply pick them up and attach them to the desired resource, bypassing the hurdle of having to write one ourselves.

IAM Identities

We have mentioned IAM identities a great deal, but what are they exactly? In simple words, an IAM Identity is any agent or collection of agents that can interact with resources and services in AWS. Said agent could be a human or it could be a computer, with different identity types being used accordingly.

IAM identities are yes, used to define permission boundaries (what an agent can or cannot do), but it is also used for easy identification, authentication and verification of the agent it relates to, helping us ensure that the agent is "legitimate". The three different identity types are very briefly explained below:

- **IAM Users**: An IAM User represents a single person, a single human being within our AWS environment. When we first create an AWS account for example, a single identity that has complete access to all AWS services and resources in the account is created. This identity is called the AWS account ***Root User***.

- **IAM Groups:** An IAM Group represents a collection of IAM Users, allowing us to easily organize groups of IAM Users together.

- **IAM Roles:** An IAM Role is a broader identity type, and can be assumed by any entity that needs it, be it human or computer. It is most widely used to bestow permissions upon specific applications and/or AWS Services. For example, if we wished for an EC2 instance to be able to access DynamoDB, we would assign an IAM Role to it and write a policy that grants access to DynamoDB before attaching it to said IAM Role.

Note for Beginners
People toying around with AWS through the Management Console may be regularly utilizing the Root User Email to login to the console, however, this is not accepted as a good practice due to various security vulnerabilities. Instead, it is considered better to create an IAM User, assign an AWS Managed Policy called `AdministratorAccess` to it, and login through the credentials of said IAM User credentials.