**AWS Identity and Access Management (IAM)**

* IAM is not an identity store/authorization system for your applications.

**Principals**

* A ***principal***is an IAM entity that is allowed to interact with AWS resources.
* There are three types of principals: **root users, IAM users, and roles/temporary security tokens**.

**Root User**

* When you first create an AWS account, you begin with only a single sign-in principal that has complete access to all AWS Cloud services and resources in the account.
* This principal is called the ***root user.***
* The root user can be used for both console and programmatic access to AWS resources.
* It has **full privileges** to do anything in the account, including closing the account.
* It is strongly recommended that you do not use the root user for your everyday tasks.

**IAM Users**

* IAM users can be created by principals with IAM administrative privileges at any time through the AWS Management Console, CLI, or SDKs.
* Users are persistent in that there is no expiration period; they are permanent entities that exist until an *IAM administrator* takes an action to delete them.
* Users are an excellent way to enforce the principle of **least privilege**

**Roles/Temporary Security Tokens**

* Roles are used to grant specific privileges to specific actors for a set duration of time.
* These actors can be authenticated by AWS or some trusted external system.
* When one of these actors assumes a role, AWS provides the actor with a temporary security token from the ***AWS Security Token Service (STS)***that the actor can use to access AWS Cloud services.
* Requesting a temporary security token requires specifying how long the token will exist before it expires.
* The range of a temporary security token **lifetime is 15 minutes to 36 hours.**
* Roles and temporary security tokens enable a number of use cases:
  + ***Amazon EC2 Roles***—Granting permissions to applications running on an Amazon EC2 instance.
  + ***Cross-Account Access***—Granting permissions to users from other AWS accounts, whether you control those accounts or not.
  + ***Federation***—Granting permissions to users authenticated by a trusted external system.

**Amazon EC2 Roles**

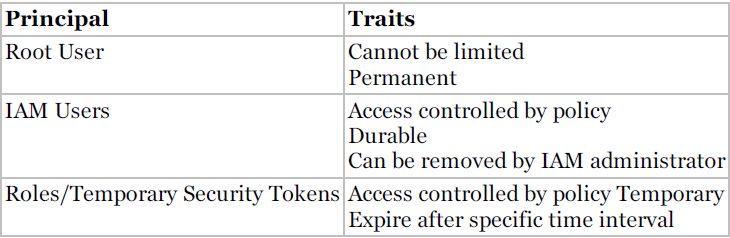
* Using IAM roles for Amazon EC2 removes the need to store AWS credentials in a configuration file.

**Cross-Account Access**

* Another common use case for IAM roles is to grant access to AWS resources to IAM users in other AWS accounts.
* You can set up an IAM role with the permissions you want to grant to users in the other account, then users in the other account can assume that role to access your resources.
* **This is highly recommended as a best practice, as opposed to distributing access keys outside your organization.**

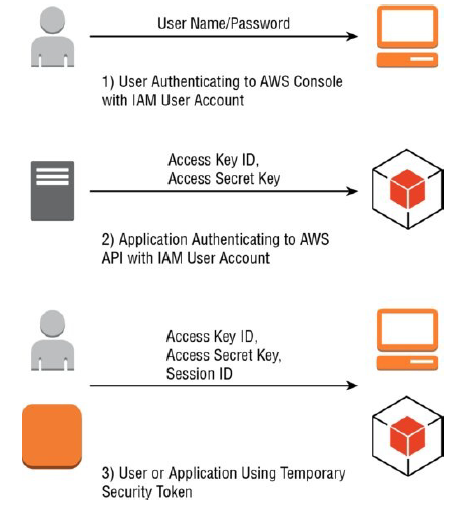
**Federation**

* *IAM Identity Providers* provide the ability to federate these outside identities with IAM and assign privileges to those users authenticated outside of IAM



**Authentication**

* ***User Name/Password***—When a principal represents a human interacting with the console, the human will provide a user name/password pair to verify their identity.
* IAM allows you to create a password policy enforcing password complexity and expiration.
* ***Access Key***—An access key is a combination of an **access key ID** (20 characters) and **an access secret key** (40 characters).
* When a program is manipulating the AWS infrastructure via the API, it will use these values to sign the underlying REST calls to the services.
* ***Access Key/Session TokeToken***—When a process operates under an assumed role, the temporary security token provides an access key for authentication.
* In addition to the access key (remember that it consists of two parts), the token also includes a ***session token***.
* Calls to AWS must include both the two-part **access key** and the **session token** to authenticate.
* It is important to note that when an IAM user is created, it has neither an access key nor a password, and the IAM administrator can set up either or both.
* This adds an extra layer of security in that console users cannot use their credentials to run a program that accesses your AWS infrastructure.



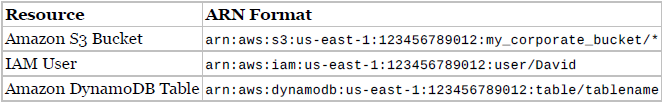
**Authorization**

* The process of specifying exactly what actions a principal can and cannot perform is called ***authorization***.
* Authorization is handled in IAM by defining specific privileges in ***policies***and associating those policies with principals.

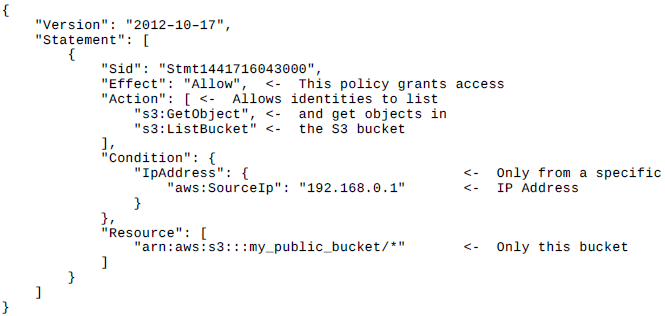
**Policies**

* A *policy* is a JSON document that fully defines a set of permissions to access and manipulate AWS resources.
* Policy documents contain one or more permissions, with each permission defining:
  + ***Effect***—A single word: Allow or Deny.
  + ***Service***—For what service does this permission apply? Most AWS Cloud services support granting access through IAM, including IAM itself.
  + ***Resource***—The resource value specifies the specific AWS infrastructure for which this permission applies. This is specified as an ***Amazon Resource Name (ARN)*.** The format for an ARN varies slightly between services, but the basic format is:

"arn:aws:service:region:account-id:[resourcetype:]resource"

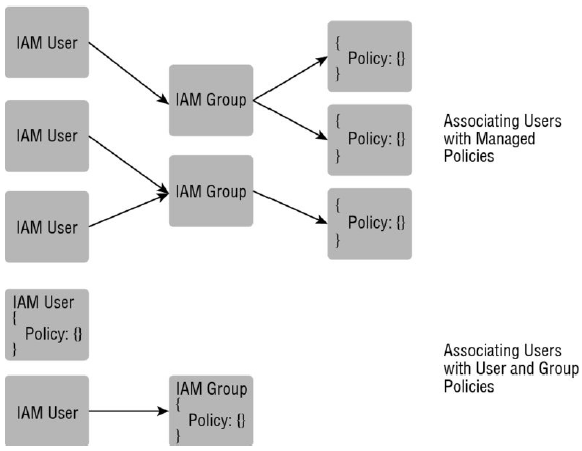


* + ***Action***—the action value specifies the subset of actions within a service that the permission allows or denies.
  + For instance, permission may grant access to any read based action for Amazon S3. A set of actions can be specified with an numerated list or by using wildcards (Read\*).
  + ***Condition***—the condition value optionally defines one or more additional restrictions that limit the actions allowed by the permission.
  + For instance, the permission might contain a condition that limits the ability to access a resource to calls that come from a specific IP address range.
  + Another condition could restrict the permission only to apply during a specific time interval.
  + There are many types of permissions that allow a rich variety of functionality that varies between services. See the IAM documentation for lists of supported conditions for each service.



**Associating Policies with Principals**

* ***User Policy***—These policies exist only in the context of the user to which they are attached. In the console, a user policy is entered into the user interface on the IAM user page.
* ***Managed Policies***—these policies are created in the Policies tab on the IAM page (or through the CLI, and so forth) and exist independently of any individual user. In this way, the same policy can be associated with many users or groups of users.
* **Using predefined managed policies ensures that when new permissions are added for new features, your users will still have the correct access.**
* **Groups** simplify managing permissions for large numbers of users.
* After a policy is assigned to a group, any user who is a member of that group assumes those permissions.
* There are two ways a policy can be associated with an IAM group:
  + ***Group Policy***—These policies exist only in the context of the group to which they are attached. In the AWS Management Console, a group policy is entered into the user interface on the IAM Group page.
  + ***Managed Policies***—In the same way that managed policies (discussed in the “Authorization” section) can be associated with IAM users, they can also be associated with IAM groups.



* A good first step is to use the root user to create a new IAM group called “IAM Administrators” and assign the managed policy, “IAMFullAccess.”
* Then create a new IAM user called “Administrator,” assign a password, and add it to the IAM Administrators group. At this point, you can log off as the root user and perform all further administration with the IAM user account.

**Other Key Features**

**Multi-Factor Authentication (MFA)**

* With MFA, authentication also requires entering a One-Time Password (OTP) from a small device.
* The MFA device can be either a small hardware device you carry with you or a virtual device via an app on your smart phone.
* MFA requires you to verify your identity with both something you *know* and something you *have.*
* An application using an IAM user configured with MFA must query the application user to provide the current code, which the application will then pass to the API.
* It is strongly recommended that AWS customers add MFA protection to their root user.

**Rotating Keys**

* To this end, it is a security best practice to *rotate access keys* associated with your IAM users.
* IAM facilitates this process by allowing two active access keys at a time.
* Access keys should be rotated on a regular schedule.