# Chapter 6
# AWS Identity and Access Management (IAM)

## THE AWS CERTIFIED SOLUTIONS ARCHITECT ASSOCIATE EXAM OBJECTIVES COVERED IN THIS CHAPTER MAY INCLUDE, BUT ARE NOT LIMITED TO, THE FOLLOWING:

**Domain 2.0: Implementation/Deployment**

✓ **2.1 Identify the appropriate techniques and methods using Amazon EC2, Amazon S3, Elastic Beanstalk, CloudFormation, Amazon Virtual Private Cloud (VPC), and AWS Identity and Access Management (IAM) to code and implement a cloud solution.**

**Content may include the following:**

- Configure IAM policies and best practices

**Domain 3.0: Data Security**

✓ **3.1 Recognize and implement secure practices for optimum cloud deployment and maintenance.**

**Content may include the following:**

- AWS Identity and Access Management (IAM)

## Introduction

In this chapter, you will learn how *AWS Identity and Access Management (IAM)* secures interactions with the AWS resources in your account, including:

- Which principals interact with AWS through the AWS Management Console, Command Line Interface (CLI), and Software Development Kits (SDKs)
- How each principal is authenticated
- How IAM policies are written to specify the access privileges of principals
- How IAM policies are associated with principals
- How to secure your infrastructure further through Multi-Factor Authentication (MFA) and key rotation
- How IAM roles can be used to delegate permissions and federate users

- How to resolve multiple, possibly conflicting IAM permissions

IAM is a powerful service that allows you to control how people and programs are allowed to manipulate your AWS infrastructure. IAM uses traditional identity concepts such as users, groups, and access control policies to control who can use your AWS account, what services and resources they can use, and how they can use them. The control provided by IAM is granular enough to limit a single user to the ability to perform a single action on a specific resource from a specific IP address during a specific time window. Applications can be granted access to AWS resources whether they are running on-premises or in the cloud. This flexibility creates a very powerful system that will give you all the power you need to ensure that your AWS account users have the ability to meet your business needs while addressing all of the security concerns of your organization.

This chapter will cover the different principals that can interact with AWS and how they are authenticated. It will then discuss how to write policies that define permitted access to services, actions, and resources and associate these policies with authenticated principals. Finally, it will cover additional features of IAM that will help you secure your infrastructure, including MFA, rotating keys, federation, resolving multiple permissions, and using IAM roles.

As important as it is to know what IAM is exactly, it is equally important to understand what it is not:

- First, IAM is not an identity store/authorization system for your applications. The permissions that you assign are permissions to manipulate AWS infrastructure, not permissions within your application. If you are migrating an existing on-premises application that already has its own user repository and authentication/authorization mechanism, then that should continue to work when you deploy on AWS and is probably the right choice. If your application identities are based on Active Directory, your on-premises Active Directory can be extended into the cloud to continue to fill that need. A great solution for using Active Directory in the cloud is AWS Directory Service, which is an Active Directory-compatible directory service that can work on its own or integrate with your on-premises Active Directory. Finally, if you are working with a mobile app, consider *Amazon Cognito* for identity management for mobile applications.
- Second, IAM is not operating system identity management. Remember that under the shared responsibility model, you are in control of your operating system console and configuration. Whatever mechanism you currently use to control access to your server infrastructure will continue to work on Amazon Elastic Compute Cloud (Amazon EC2) instances, whether that is managing individual machine login accounts or a directory service such as Active Directory or Lightweight Directory Access Protocol (LDAP). You can run an Active Directory or LDAP server on Amazon EC2, or you can extend your on-premises system into the cloud. AWS Directory Service will also work well to provide Active Directory functionality in the cloud as a service, whether standalone or integrated with your existing Active Directory.

Table 6.1 summarizes the role that different authentication systems can play in your AWS environment.

**TABLE 6.1** Authentication Technologies

| **Use Case** | **Technology Solutions** |
|---|---|
| Operating System Access | Active Directory LDAP Machine-specific accounts |
| Application Access | Active Directory<br>Application User Repositories<br>Amazon Cognito |
| AWS Resources | IAM |

IAM is controlled like most other AWS Cloud services:

- Through the *AWS Management Console*—Like other services, the AWS Management Console is the easiest way to start learning about and manipulating a service.
- With the *CLI*—As you learn the system, you can start scripting repeated tasks using the CLI.
- Via the *AWS SDKs*—Eventually you may start writing your own tools and complex processes by manipulating IAM directly through the REST API via one of several SDKs.

All of these methods work to control IAM just as they work with other services. In addition, the AWS Partner Network (APN) includes a rich ecosystem of tools to manage and extend IAM.

## Principals

The first IAM concept to understand is principals. A *principal* is an IAM entity that is allowed to interact with AWS resources. A principal can be permanent or temporary, and it can represent a human or an application. 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*. As long as you have an open account with AWS, the root user for that relationship will persist. The root user can be used for both console and programmatic access to AWS resources.

The root user is similar in concept to the UNIX root or Windows Administrator account—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, even the administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user and then securely locking away the root user credentials.

### IAM Users

Users are persistent identities set up through the IAM service to represent individual people or applications. You may create separate IAM users for each member of your operations team so they can interact with the console and use the CLI. You might also create dev, test, and production users for applications that need to access AWS Cloud services (although you will see later in this chapter that IAM roles may be a better solution for that use case).

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; that is, the concept of allowing a person or process interacting with your AWS resources to perform exactly the tasks they need but nothing else. Users can be associated with very granular policies that define these permissions. Policies will be covered in a later section.

### Roles/Temporary Security Tokens

Roles and temporary security tokens are very important for advanced IAM usage, but many AWS users find them confusing. 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

Granting permissions to an application is always tricky, as it usually requires configuring the application with some sort of credential upon installation. This leads to issues around securely storing the credential prior to use, how to access it safely during installation, and how to secure it in the configuration. Suppose that an application running on an Amazon EC2 instance needs to access an Amazon Simple Storage Service (Amazon S3) bucket. A policy granting permission to read and write that bucket can be created and assigned to an IAM user, and the application can use the access key for that IAM user to access the Amazon S3 bucket. The problem with this approach is that the access key for the user must be accessible to the application, probably by storing it in some sort of configuration file. The process for obtaining the access key and storing it encrypted in the configuration is usually complicated and a hindrance to agile development. Additionally, the access key is at risk when being passed around. Finally, when the time comes to rotate the access key, the rotation involves performing that whole process again.

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

An alternative is to create an IAM role that grants the required access to the Amazon S3 bucket. When the Amazon EC2 instance is launched, the role is assigned to the instance. When the application running on the instance uses the Application Programming Interface (API) to access the Amazon S3 bucket, it assumes the role assigned to the instance and obtains a temporary token that it sends to the API. The process of obtaining the temporary token and passing it to the API is handled automatically by most of the AWS SDKs, allowing the application to make a call to access the Amazon S3 bucket without worrying about authentication. In addition to being easy for the developer, this removes any need to store an access key in a configuration file. Also, because the API access uses a temporary token, there is no fixed access key that must be rotated.

## Cross-Account Access

Another common use case for IAM roles is to grant access to AWS resources to IAM users in other AWS accounts. These accounts may be other AWS accounts controlled by your company or outside agents like customers or suppliers. 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

Many organizations already have an identity repository outside of AWS and would rather leverage that repository than create a new and largely duplicate repository of IAM users. Similarly, web-based applications may want to leverage web-based identities such as Facebook, Google, or Login with Amazon. *IAM Identity Providers* provide the ability to federate these outside identities with IAM and assign privileges to those users authenticated outside of IAM.

IAM can integrate with two different types of outside *Identity Providers (IdP)*. For federating web identities such as Facebook, Google, or Login with Amazon, IAM supports integration via OpenID Connect (OIDC). This allows IAM to grant privileges to users authenticated with some of the major web-based IdPs. For federating internal identities, such as Active Directory or LDAP, IAM supports integration via Security Assertion Markup Language 2.0 (SAML). A SAML-compliant IdP such as Active Directory Federation Services (ADFS) is used to federate the internal directory to IAM. (Instructions for configuring many compatible products can be found on the AWS website.) In each case, federation works by returning a temporary token associated with a role to the IdP for the authenticated identity to use for calls to the AWS API. The actual role returned is determined via information received from the IdP, either attributes of the user in the on-premises identity store or the user name and authenticating service of the web identity store.

The three types of principals and their general traits are listed in Table 6.2.

**TABLE 6.2** Traits of AWS Principals

| Principal | Traits |
|---|---|
| Root User | Cannot be limited<br>Permanent |
| IAM Users | Access controlled by policy<br>Durable<br>Can be removed by IAM administrator |
| Roles/Temporary Security Tokens | Access controlled by policy Temporary<br>Expire after specific time interval |

## Authentication

There are three ways that IAM *authenticates* a principal:

- ***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. The AWS SDKs and tools handle all the intricacies of signing the REST calls, so using an access key will almost always be a matter of providing the values to the SDK or tool.
- ***Access Key/Session Token***—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.

Figure 6.1 shows a summary of the different authentication methods.

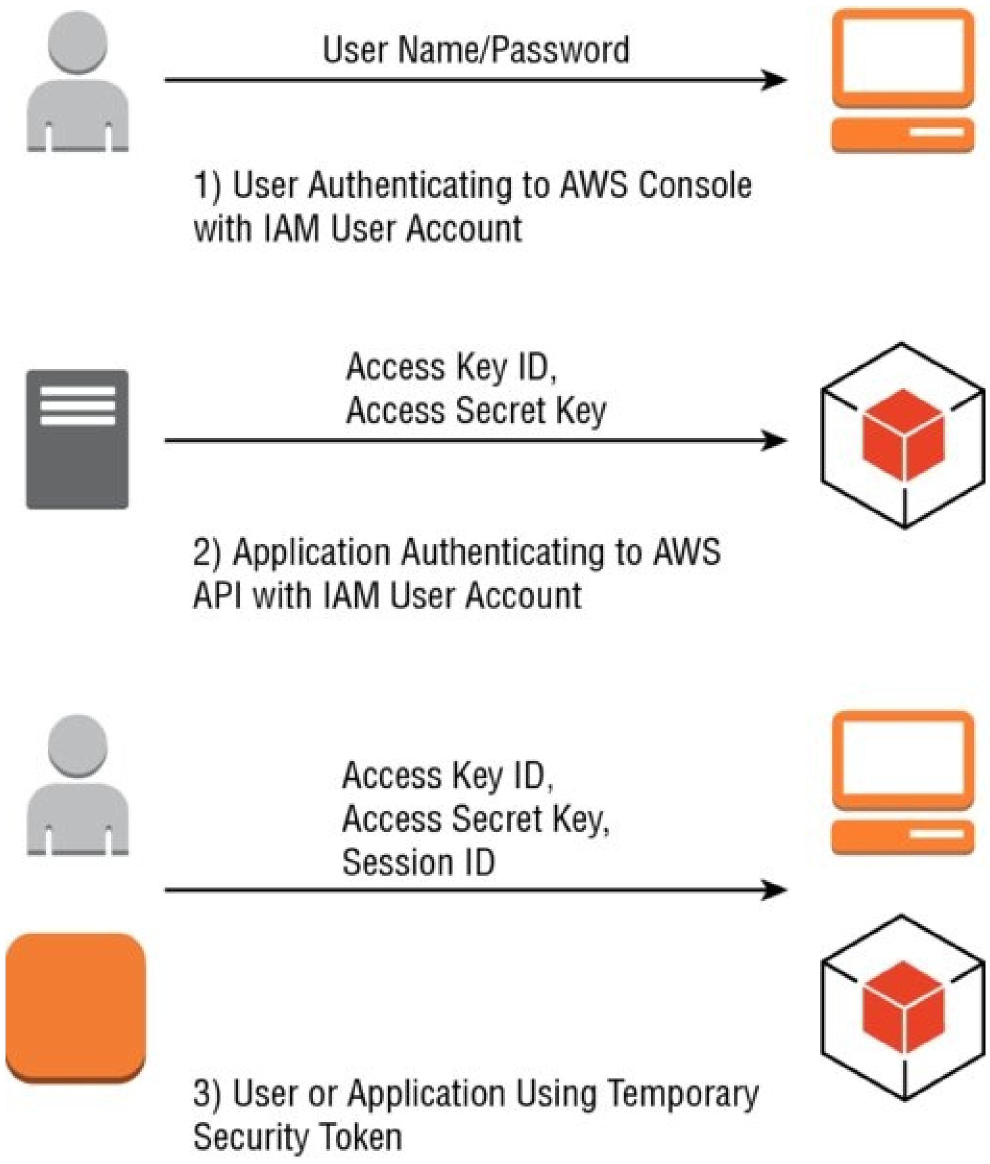
User Name/Password
1) User Authenticating to AWS Console with IAM User Account
Access Key ID, Access Secret Key
2) Application Authenticating to AWS API with IAM User Account
Access Key ID, Access Secret Key, Session ID
3) User or Application Using Temporary Security Token

**FIGURE 6.1** Different identities authenticating with AWS

## Authorization

After IAM has authenticated a principal, it must then manage the access of that principal to protect your AWS infrastructure. 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

Understanding how access management works under IAM begins with understanding 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"
  ```

For some services, wildcard values are allowed; for instance, an Amazon S3 ARN could have a resource of `foldername\*` to indicate all objects in the specified folder. Table 6.3 displays some sample ARNs.

**TABLE 6.3** Sample ARNs

| **Resource** | **ARN Format** |
|---|---|
| Amazon S3 Bucket | `arn:aws:s3:us-east-1:123456789012:my_corporate_bucket/*` |
| IAM User | `arn:aws:iam:us-east-1:123456789012:user/David` |
| Amazon DynamoDB Table | `arn:aws:dynamodb:us-east-1:123456789012:table/tablename` |

- ***Action***—The action value specifies the subset of actions within a service that the permission allows or denies. For instance, a permission may grant access to any read-based action for Amazon S3. A set of actions can be specified with an enumerated 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.

A sample policy is shown in the following listing. This policy allows a principal to list the objects in a specific bucket and to retrieve those objects, but only if the call comes from a specific IP address.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1441716043000",
            "Effect": "Allow",  <-  This policy grants access
            "Action": [ <-  Allows identities to list
                "s3:GetObject", <-  and get objects in
                "s3:ListBucket" <-  the S3 bucket
            ],
            "Condition": {
                "IpAddress": {                          <-  Only from a specific
                    "aws:SourceIp": "192.168.0.1"       <-  IP Address
                }
            },
            "Resource": [
                "arn:aws:s3:::my_public_bucket/*"       <-  Only this bucket
            ]
        }
    ]
}
```

## Associating Policies with Principals

There are several ways to associate a policy with an IAM user; this section will only cover the most common.

A policy can be associated directly with an IAM user in one of two ways:

- ***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. There are a large number of predefined managed policies that you can review on the Policies tab of the IAM page in the AWS Management Console. In addition, you can write your own policies specific to your use cases.

TIP Using predefined managed policies ensures that when new permissions are added for new features, your users will still have the correct access.

The other common method for associating policies with users is with the IAM groups feature. 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. This makes it simpler to assign policies to an entire team in your organization. For instance, if you create an “Operations” group with every IAM user for your operations team assigned to that group, then it is a simple matter to associate the needed permissions to the group, and all of the team's IAM users will assume those permissions. New IAM users can then be assigned directly to the group.

This is a much simpler management process than having to review what policies a new IAM user for the operations team should receive and manually adding those policies to the user. 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.

Figure 6.2 shows the different ways that polices can be associated with an IAM User.

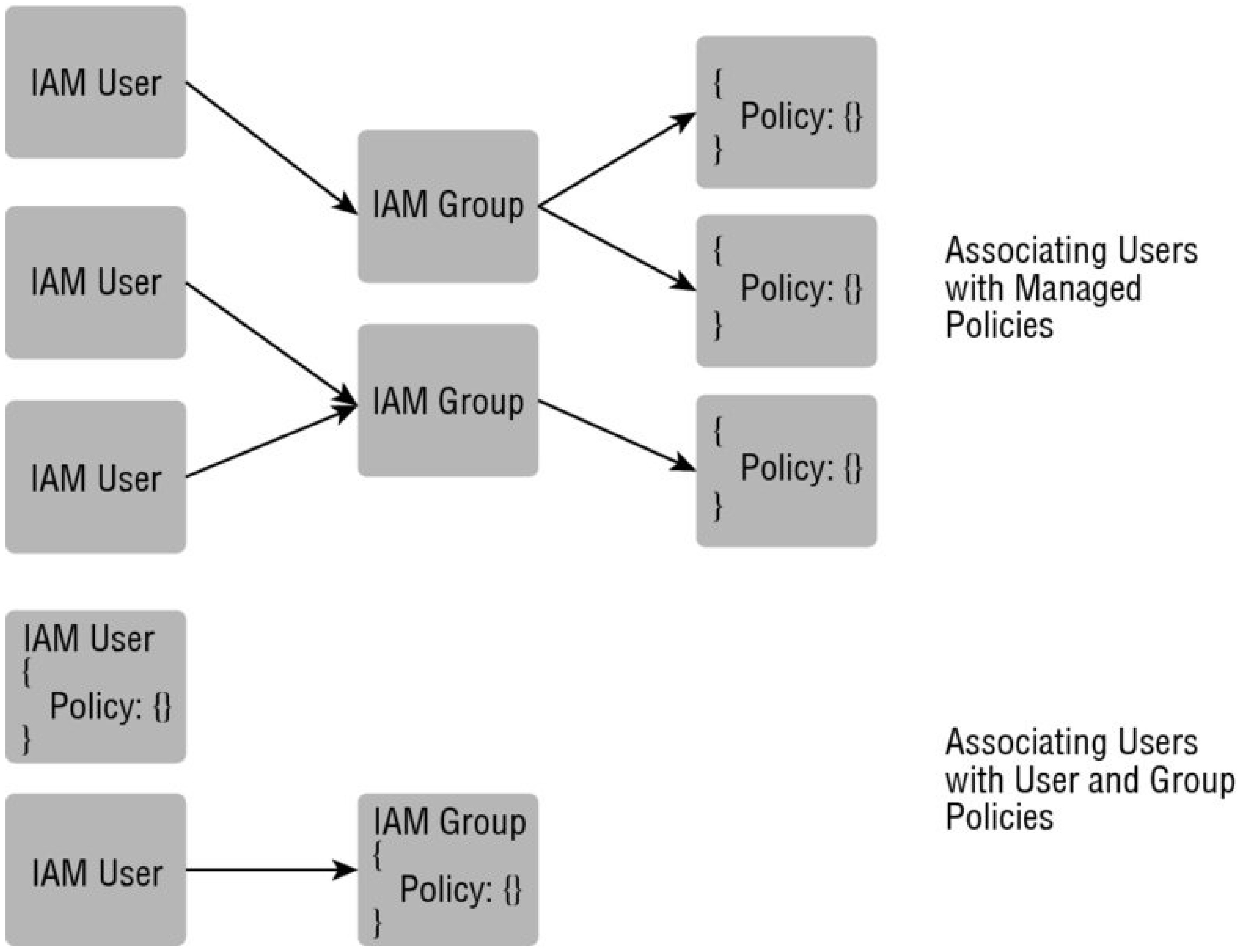

**FIGURE 6.2** Associating IAM users with policies

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.

The final way an actor can be associated with a policy is by assuming a role. In this case, the actor can be:

- An authenticated IAM user (person or process). In this case, the IAM user must have the rights to assume the role.
- A person or process authenticated by a trusted service outside of AWS, such as an on-premises LDAP directory or a web authentication service. In this situation, an AWS Cloud service will assume the role on the actor’s behalf and return a token to the actor.

After an actor has assumed a role, it is provided with a temporary security token associated with the policies of that role. The token contains all the information required to authenticate API calls. This information includes a standard access key plus an additional session token required for authenticating calls under an assumed role.

## Other Key Features

Beyond the critical concepts of principals, authentication, and authorization, there are several other features of the IAM service that are important to understand to realize the full benefits of IAM.

### Multi-Factor Authentication (MFA)

*Multi-Factor Authentication (MFA)* can add an extra layer of security to your infrastructure by adding a second method of authentication beyond just a password or access key. 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 (for example, the AWS Virtual MFA app).

> MFA requires you to verify your identity with both something you *know* and something you *have*.

MFA can be assigned to any IAM user account, whether the account represents a person or application. When a person using an IAM user configured with MFA attempts to access the AWS Management Console, after providing their password they will be prompted to enter the current code displayed on their MFA device before being granted access. 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

The security risk of any credential increases with the age of the credential. 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. The process to rotate keys can be conducted via the console, CLI, or SDKs:

1. Create a new access key for the user.
2. Reconfigure all applications to use the new access key.
3. Disable the original access key (disabling instead of deleting at this stage is critical, as it allows rollback to the original key if there are issues with the rotation).
4. Verify the operation of all applications.
5. Delete the original access key.

> Access keys should be rotated on a regular schedule.

### Resolving Multiple Permissions

Occasionally, multiple permissions will be applicable when determining whether a principal has the privilege to perform some action. These permissions may come from multiple policies associated with a principal or resource policies attached to the AWS resource in question. It is important to know how conflicts between these permissions are resolved:

1. Initially the request is denied by default.
2. All the appropriate policies are evaluated; if there is an explicit “deny” found in any policy, the request is denied and evaluation stops.
3. If no explicit “deny” is found and an explicit “allow” is found in any policy, the request is allowed.
4. If there are no explicit “allow” or “deny” permissions found, then the default “deny” is maintained and the request is denied.

The only exception to this rule is if an `AssumeRole` call includes a role and a policy, the policy cannot expand the privileges of the role (for example, the policy cannot override any permission that is denied by default in the role).

## Summary

IAM is a powerful service that gives you the ability to control which people and applications can access your AWS account at a very granular level. Because the root user in an AWS account cannot be limited, you should set up IAM users and temporary security tokens for your people and processes to interact with AWS.

Policies define what actions can and cannot be taken. Policies are associated with IAM users either directly or through group membership. A temporary security token is associated with a policy by assuming an IAM role. You can write your own policies or use one of the managed policies provided by AWS.

Common use cases for IAM roles include federating identities from external IdPs, assigning privileges to an Amazon EC2 instance where they can be assumed by applications running on the instance, and cross-account access.

IAM user accounts can be further secured by rotating keys, implementing MFA, and adding conditions to policies. MFA ensures that authentication is based on something you have in addition to something you know, and conditions can add further restrictions such as limiting client IP address ranges or setting a particular time interval.

## Exam Essentials

**Know the different principals in IAM.** The three principals that can authenticate and interact with AWS resources are the root user, IAM users, and roles. The root user is associated with the actual AWS account and cannot be restricted in any way. IAM users are persistent identities that can be controlled through IAM. Roles allow people or processes the ability to operate temporarily with a different identity. People or processes assume a role by being granted a temporary security token that will expire after a specified period of time.

**Know how principals are authenticated in IAM.** When you log in to the AWS Management Console as an IAM user or root user, you use a user name/password combination. A program that accesses the API with an IAM user or root user uses a two-part access key. A temporary security token authenticates with an access key plus an additional session token unique to that temporary security token.

**Know the parts of a policy.** A policy is a JSON document that defines one or more permissions to interact with AWS resources. Each permission includes the effect, service, action, and resource. It may also include one or more conditions. AWS makes many predefined policies available as managed policies.

**Know how a policy is associated with a principal.** An authenticated principal is associated with zero to many policies. For an IAM user, these policies may be attached directly to the user account or attached to an IAM group of which the user account is a member. A temporary security token is associated with policies by assuming an IAM role.

**Understand MFA.** MFA increases the security of an AWS account by augmenting the password (something you know) with a rotating OTP from a small device (something you have), ensuring that anyone authenticating the account has both knowledge of the password and possession of the device. AWS supports both Gemalto hardware MFA devices and a number of virtual MFA apps.

**Understand key rotation.** To protect your AWS infrastructure, access keys should be rotated regularly. AWS allows two access keys to be valid simultaneously to make the rotation process straightforward: Generate a new access key, configure your application to use the new access key, test, disable the original access key, test, delete the original access key, and test again.

**Understand IAM roles and federation.** IAM roles are prepackaged sets of permissions that have no credentials. Principals can assume a role and then use the associated permissions. When a temporary security token is created, it assumes a role that defines the permissions assigned to the token. When an Amazon EC2 instance is associated with an IAM role, SDK calls acquire a temporary security token based on the role associated with the instance and use that token to access AWS resources.

Roles are the basis for federating external IdPs with AWS. You configure an IAM IdP to interact with the external IdP, the authenticated identity from the IdP is mapped to a role, and a temporary security token is returned that has assumed that role. AWS supports both SAML and OIDC IdPs.

**Know how to resolve conflicting permissions**. Resolving multiple permissions is relatively straightforward. If an action on a resource has not been explicitly allowed by a policy, it is denied. If two policies contradict each other; that is, if one policy allows an action on a resource and another policy denies that action, the action is denied. While this sounds improbable, it may occur due to scope differences in a policy. One policy may expose an entire fleet of Amazon EC2 instances, and a second policy may explicitly lock down one particular instance.

## Exercises

For assistance in completing the following exercises, refer to the IAM User Guide at http://docs.aws.amazon.com/IAM/latest/UserGuide/.

### EXERCISE 6.1

#### Create an IAM Group

In this exercise, you will create a group for all IAM administrator users and assign the proper permissions to the new group. This will allow you to avoid assigning policies directly to a user later in these exercises.

1. Log in as the root user.
2. Create an IAM group called `Administrators`.
3. Attach the managed policy, `IAMFullAccess`, to the `Administrators` group.

### EXERCISE 6.2

#### Create a Customized Sign-In Link and Password Policy

In this exercise, you will set up your account with some basic IAM safeguards. The password policy is a recommended security practice, and the sign-in link makes it easier for your users to log in to the AWS Management Console.

1. Customize a sign-in link, and write down the new link name in full.
2. Create a password policy for your account.

### EXERCISE 6.3

#### Create an IAM User

In this exercise, you will create an IAM user who can perform all administrative IAM functions. Then you will log in as that user so that you no longer need to use the root user login. Using the root user login only when explicitly required is a recommended security practice (along with adding MFA to your root user).

1. While logged in as the root user, create a new IAM user called `Administrator`.
2. Add your new user to the `Administrators` group.
3. On the Details page for the administrator user, create a password.
4. Log out as the root user.
5. Use the customized sign-in link to sign in as `Administrator`.

## EXERCISE 6.4

### Create and Use an IAM Role

In this exercise, you will create an IAM role, associate it with a new instance, and verify that applications running on the instance assume the permissions of the role. IAM roles allow you to avoid storing access keys on your Amazon EC2 instances.

1. While signed in as administrator, create an Amazon EC2-type role named `S3Client`.
2. Attach the managed policy, `AmazonS3ReadOnlyAccess`, to `S3Client`.
3. Launch an Amazon Linux EC2 instance with the new role attached (Amazon Linux AMIs come with CLI installed).
4. SSH into the new instance, and use the CLI to list the contents of an Amazon S3 bucket.

## EXERCISE 6.5

### Rotate Keys

In this exercise, you will go through the process of rotating access keys, a recommended security practice.

1. Select the administrator, and create a two-part access key.
2. Download the access key.
3. Download and install the CLI to your desktop.
4. Configure the CLI to use the access key with the AWS Configure command.
5. Use the CLI to list the contents of an Amazon S3 bucket.
6. Return to the console, and create a new access key for the administrator account.
7. Download the access key, and reconfigure the CLI to use the new access key.
8. In the console, make the original access key inactive.
9. Confirm that you are using the new access key by once again listing the contents of the Amazon S3 bucket.
10. Delete the original access key.

## EXERCISE 6.6

### Set Up MFA

In this exercise, you will add MFA to your IAM administrator. You will use a virtual MFA application for your phone. MFA is a security recommendation on powerful accounts such as IAM administrators.

1. Download the AWS Virtual MFA app to your phone.
2. Select the administrator user, and manage the MFA device.
3. Go through the steps to activate a Virtual MFA device.
4. Log off as administrator.
5. Log in as administrator, and enter the MFA value to complete the authentication process.

## EXERCISE 6.7

### Resolve Conflicting Permissions

In this exercise, you will add a policy to your IAM administrator user with a conflicting permission. You will then attempt actions that verify how IAM resolves conflicting permissions.

1. Use the policy generator to create a new policy.
2. Create the policy with Effect: Deny; AWS Service: Amazon S3; Actions: *; and ARN: *.
3. Attach the new policy to the `Administrators` group.
4. Use the CLI to attempt to list the contents of an Amazon S3 bucket. The policy that allows access and the policy that denies access should resolve to deny access.

## Review Questions

1. Which of the following methods will allow an application using an AWS SDK to be authenticated as a principal to access AWS Cloud services? (Choose 2 answers)
   A. Create an IAM user and store the user name and password for the user in the application's configuration.
   B. Create an IAM user and store both parts of the access key for the user in the application's configuration.
   C. Run the application on an Amazon EC2 instance with an assigned IAM role.
   D. Make all the API calls over an SSL connection.
2. Which of the following are found in an IAM policy? (Choose 2 answers)
   A. Service Name
   B. Region
   C. Action
   D. Password
3. Your AWS account administrator left your company today. The administrator had access to the root user and a personal IAM administrator account. With these accounts, he generated other IAM accounts and keys. Which of the following should you do today to protect your AWS infrastructure? (Choose 4 answers)
   A. Change the password and add MFA to the root user.
   B. Put an IP restriction on the root user.
   C. Rotate keys and change passwords for IAM accounts.
   D. Delete all IAM accounts.
   E. Delete the administrator's personal IAM account.
   F. Relaunch all Amazon EC2 instances with new roles.
4. Which of the following actions can be authorized by IAM? (Choose 2 answers)
   A. Installing ASP.NET on a Windows Server
   B. Launching an Amazon Linux EC2 instance
   C. Querying an Oracle database
   D. Adding a message to an Amazon Simple Queue Service (Amazon SQS) queue
5. Which of the following are IAM security features? (Choose 2 answers)
   A. Password policies
   B. Amazon DynamoDB global secondary indexes
   C. MFA

D. Consolidated Billing

6. Which of the following are benefits of using Amazon EC2 roles? (Choose 2 answers)

   A. No policies are required.

   B. Credentials do not need to be stored on the Amazon EC2 instance.

   C. Key rotation is not necessary.

   D. Integration with Active Directory is automatic.

7. Which of the following are based on temporary security tokens? (Choose 2 answers)

   A. Amazon EC2 roles

   B. MFA

   C. Root user

   D. Federation

8. Your security team is very concerned about the vulnerability of the IAM administrator user accounts (the accounts used to configure all IAM features and accounts). What steps can be taken to lock down these accounts? (Choose 3 answers)

   A. Add multi-factor authentication (MFA) to the accounts.

   B. Limit logins to a particular U.S. state.

   C. Implement a password policy on the AWS account.

   D. Apply a source IP address condition to the policy that only grants permissions when the user is on the corporate network.

   E. Add a CAPTCHA test to the accounts.

9. You want to grant the individuals on your network team the ability to fully manipulate Amazon EC2 instances. Which of the following accomplish this goal? (Choose 2 answers)

   A. Create a new policy allowing EC2:* actions, and name the policy NetworkTeam.

   B. Assign the managed policy, EC2FullAccess, to a group named NetworkTeam, and assign all the team members' IAM user accounts to that group.

   C. Create a new policy that grants EC2:* actions on all resources, and assign that policy to each individual's IAM user account on the network team.

   D. Create a NetworkTeam IAM group, and have each team member log in to the AWS Management Console using the user name/password for the group.

10. What is the format of an IAM policy?

    A. XML

    B. Key/value pairs

    C. JSON

    D. Tab-delimited text