AWS Identity and Access Management

Using IAM

API Version 2010-05-08

AWS Identity and Access Management: Using IAM

AWS services or capabilities described in AWS Documentation may vary by region/location. Click Getting Started with Amazon AWS to see specific differences applicable to the China (Beijing) Region.

Table of Contents

What Is IAM?

This section provides an introduction to IAM.

AWS Identity and Access Management is a web service that enables Amazon Web Services (AWS) customers to manage users and user permissions in AWS. The service is targeted at organizations with multiple users or systems that use AWS products such as Amazon EC2, Amazon RDS, and the AWS Management Console. With IAM, you can centrally manage users, security credentials such as access keys, and permissions that control which AWS resources users can access.

Without IAM, organizations with multiple users and systems must either create multiple AWS accounts, each with its own billing and subscriptions to AWS products, or employees must all share the security credentials of a single AWS account. Also, without IAM, you have no control over the tasks a particular user or system can do and what AWS resources they might use.

IAM addresses this issue by enabling organizations to create multiple *users* (each user is a person, system, or application) who can use AWS products, each with individual security credentials, all controlled by and billed to a single AWS account. With IAM, each user is allowed to do only what they *need* to do as part of the user's job.

Topics

Video Introduction to IAM

In the following video you'll learn the basics of using IAM to manage access to specific resources in your organization's AWS account. This video uses the AWS Management Console to show you how to create groups of users, set permissions for each group, generate a password, and use a sign-in URL to sign in to the console as an IAM user. Introduction to AWS Identity and Access Management (IAM)

Pricing of IAM

AWS Identity and Access Management is a feature of your AWS account offered at no additional charge. You will be charged only for use of other AWS services by your IAM users. For information about the pricing of other AWS services, see the Amazon Web Services pricing page.

Features of IAM

IAM includes the following features:

- **Central control of users and security credentials—**You can control creation, rotation, and revocation of each user's AWS security credentials (such as access keys)
- **Central control of user access—**You can control what data in the AWS system users can access and how they access it
- **Shared AWS resources—**Users can share data for collaborative projects
- **Permissions based on organizational groups—**You can restrict users' AWS access based on their job duties (for example, admin, developer, etc.) or departments. When users move inside the organization, you can easily update their AWS access to reflect the change in their role
- **Central control of AWS resources—**Your organization maintains central control of the AWS data the users create, with no breaks in continuity or lost data as users move around within or leave the organization
- **Control over resource creation—**You can help make sure that users create AWS data only in sanctioned places
- **Networking controls—**You can help make sure that users can access AWS resources only from within the organization's corporate network, using SSL
- **Single AWS bill—**Your organization's AWS account gets a single AWS bill for all your users' AWS activity

Supported AWS Products

IAM is integrated with many AWS products. For information about which services are integrated with IAM, see AWS Services That Support IAM (p. 265).

Note
The APIs for services do not change when they add support for IAM. Products that integrate with IAM have no new API actions related to access control.

Migration to IAM

If your organization already uses AWS, migrating to IAM can be easy or potentially more challenging, depending on how your organization currently allocates its AWS resources. Here are the three scenarios.

1. **Your organization has just a single AWS account.** In this case, you can easily migrate to using IAM, because all the organization's AWS resources are already together under a single AWS account.

2. **Your organization has multiple AWS accounts, with each AWS account belonging to a division in the organization.** If these divisions don't need to share resources or users, then migrating is easy. Each division can keep its own AWS account and use IAM separately from the other divisions. You could also use Consolidated Billing, which would allow your organization to get a single bill across the AWS accounts (see IAM and Consolidated Billing (p. 4)).
3. **Your organization has multiple AWS accounts that don't represent logical boundaries between divisions.** If you need the AWS accounts to share their resources and have common users, migrating to IAM will be more of a challenge. You will need to move the resources that need to be shared so they're under the ownership of a single AWS account. However, there's no automatic way to transfer the AWS resources from one AWS account to another. You need to create those resources again under the single AWS account.

No Change to Basic AWS Account Functions

There's no change to how an AWS account functions in terms of its login/password, security credentials, payment method, AWS account activity page, usage report, and so on. At this time, the AWS account activity page does not show a breakdown by user.

Security Credentials

Any person or application that interacts with AWS requires *security credentials*. AWS uses these credentials to identify who is making the call and whether to allow the requested access.

When you sign up for AWS, you sign up with an email address and password. Using these credentials, you can get full access to all resources in your AWS account. Because you can't control access on account credentials, AWS recommends that you use IAM credentials for day-to-day interaction with AWS. We recommend that you lock away the credentials that you used for setting up the account. As soon as you've created your account, set up an administrators group for your organization, create IAM users (including one for yourself), add them to the administrators group, and then give them privileges to administer your AWS resources. For more information, see IAM Best Practices (p. 26).

Note
To help control who has access to the AWS account's credentials, AWS recommends that you use multi-factor authentication (MFA) with your AWS account's email address and password. For detailed information about AWS MFA, see the AWS Multi-Factor Authentication FAQs.

With IAM, you can control who can access which resources. For example, you can create individual users and give them each their own user name, password, and access keys. After you create your users, you can assign them different permissions to control which resources they can access. For more information, see IAM Users and Groups (p. 39).

You can also use MFA with IAM. For information, see Using Multi-Factor Authentication (MFA) Devices with AWS (p. 69).

For more information about account vs. IAM credentials, see Root Account Credentials vs. IAM User Credentials in the *AWS General Reference*.

How Do I Get Credentials?

AWS account credentials
To manage your AWS account credentials, such as your password, access keys (access key ID and secret access key), or MFA device, sign in with your account's email address and password at https://console.amazonaws.cn//iam/home?#security_credential.

IAM credentials

By default, a user has no security credentials. You create security credentials for your users as needed.

If your users require access to the AWS Management Console, you must create passwords for them. For more information, see Credentials (Passwords, Access Keys, and MFA devices) (p. 56). Users sign in to the console using a special URL for your account. For more information, see How IAM Users Sign In to Your AWS Account (p. 46).

If your users require programmatic access to AWS, you must create access keys (access key ID and secret access key) for them. For more information, see Creating an IAM User in Your AWS Account (p. 43).

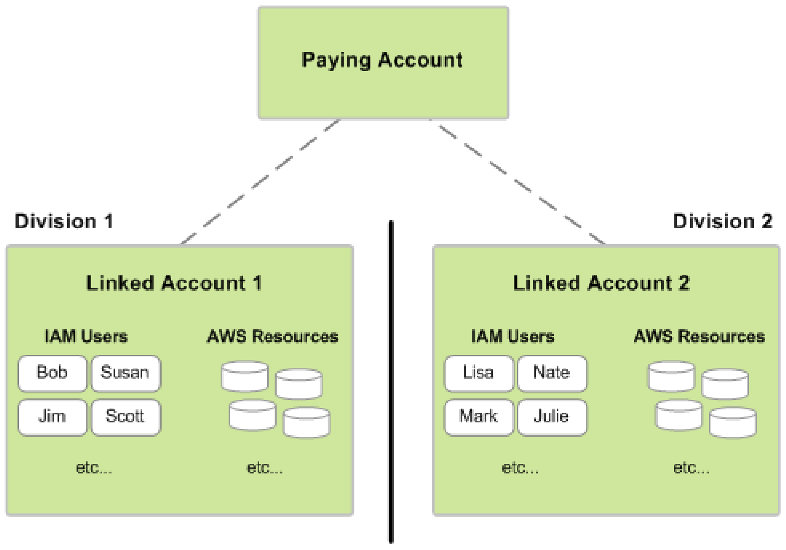
You can also grant your users permission to create and manage their own credentials, or you can have an administrators group in your organization handle this. To grant users permissions to manage their credentials, see Permissions for Administering IAM Users, Groups, and Credentials (p. 95).

Important
For security purposes, we recommend that you rotate your users' credentials on a regular basis. A user can have multiple access keys at a given time for this purpose. For more information, see Rotating Credentials (p. 92).

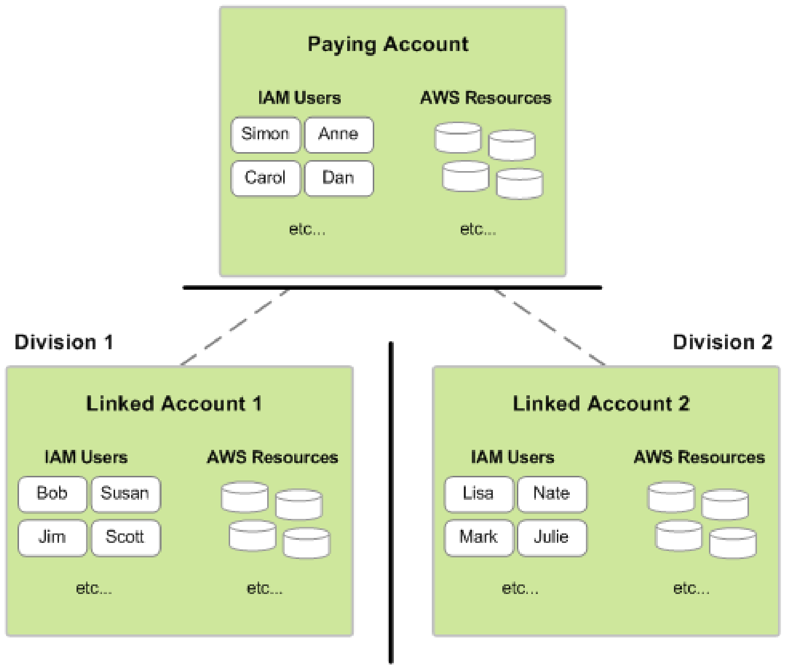
IAM and Consolidated Billing

AWS offers a billing feature called *Consolidated Billing*. This lets you receive a single bill for multiple AWS accounts. (For more information, see Consolidated Billing in *AWS Billing and Cost Management User Guide*.) In contrast, IAM lets you get a single bill across all the *users* in a single AWS account.

Your organization can use Consolidated Billing and IAM together. You might do this if your organization has multiple large divisions, and you want to isolate the users and AWS resources in each division from the other divisions. You could have a separate AWS account for each division, and use IAM in each division to create users and control their access to the division's AWS resources. You could then use Consolidated Billing to get a single bill across all the AWS accounts. The following diagram illustrates the concept.



With Consolidated Billing, one AWS account becomes the *paying account*, and pays for its own charges plus the charges of any *linked AWS accounts*. Each linked AWS account doesn't need to maintain a payment method with AWS, only the paying account does. Each month, AWS charges the paying account only. The paying account still functions like a normal AWS account; it could have its own users and AWS resources. Just as with the other AWS accounts, the users and resources in the paying account are isolated from the users and resources in the divisions' AWS accounts. The following diagram shows the paying account with its own users and AWS resources.



IAM Concepts

Topics

To help you understand IAM, we recommend you think about it in the following parts:

- The basic *entities* that comprise your AWS account (such as users and groups)
- The *permissions* you grant to the entities in your AWS account

Concepts Related to AWS Account Entities

This section describes the basic entities in your AWS account and how they fit together. They're presented in a logical order, with the first items you need to know at the start of the list.

AWS Account

Tip
If you're already an AWS customer, you're probably already familiar with AWS accounts and their characteristics. With IAM, an AWS account remains essentially the same, except the account can now have *users* under its umbrella.

An *AWS account* is the first entity you create when initiating a relationship with AWS. The AWS account centrally owns all the resources created under its umbrella and pays for all AWS activity for those resources.

Any permissions created for users or groups within the AWS account don't apply to the AWS account itself. The AWS account has permission to do anything and everything with all the AWS account resources. In this way, the AWS account is similar in concept to the UNIX *root* or *superuser*.

The AWS account has its own set of security credentials that can be used to interact with AWS programmatically. The AWS account also has an email address login and password, giving it access to the AWS Management Console and to the secure pages on the AWS website. The secure pages display the AWS account's security credentials and payment method, among other data.

Note
To help keep that sensitive information secure, we recommend that you use Multi-Factor Authentication (MFA) with your AWS account's email login and password. For detailed information about AWS MFA, see the AWS Multi-Factor Authentication FAQs. For information about using MFA with IAM, see Using Multi-Factor Authentication (MFA) Devices with AWS (p. 69).

Ideally, someone in your organization uses the AWS account credentials to set up an administrator group for the AWS account. The group has admin users responsible for all subsequent management of users and permissions. After setting up the admins group, your organization would no longer use the AWS account credentials, and only users (with their own credentials) would interact with AWS from then on.

Note
If your users need to interact with an AWS product that doesn't integrate with IAM, then they must use the AWS account's credentials. For a list of AWS products that integrate with IAM, see Supported AWS Products (p. 2).

Before using IAM, your organization might have had a single AWS account for all the employees, or the organization might have given each division or each employee a separate AWS account. With IAM, your organization will probably have one or only a few AWS accounts, but many users. The organization won't need to have an individual AWS account per employee. If your organization already has an AWS account, you don't need to get a new or different one for your organization to use IAM. In fact, we recommend you keep the existing one for your organization when you begin to use IAM, because any AWS resources that your organization has already created can't be moved to a different AWS account.

Role

A *role* is an entity that has a set of permissions, and that another entity assumes to make calls to access your AWS resources. The entity who assumes the role uses temporary security credentials to make calls. For more information about roles and how they differ from users or groups, see IAM Roles (Delegation and Federation) (p. 116).

Concepts Related to Permissions

This section summarizes the basic concepts related to permissions. For a more detailed discussion of these concepts, see Permissions and Policies (p. 171).

Resource

A *resource* is an entity in an AWS service that a user can interact with, such as an Amazon S3 bucket or object, an Amazon SQS queue, and so on.

Resources typically have a friendly name (such as `example_bucket`), and then an *Amazon Resource Name (ARN)*, which uses a standardized format and uniquely identifies the resource in AWS. For more information about ARNs, see IAM Identifiers (p. 8).

Permission

A *permission* is the concept of allowing (or disallowing) an entity such as a user, group, or role some type of access to one or more resources. For example, Bob has permission to read and write objects to a particular Amazon S3 bucket named example_bucket.

There are two general types or permissions you might use within AWS: *user-based* and *resource-based*. The easiest way to understand the difference is to think of the question each type of permission answers:

- **User-based—**What does a particular entity have access to?

  For example, Bob might have permission to use the Amazon EC2 `RunInstances` actions with any of the AWS account's resources, permission to access items in a specific Amazon S3 bucket that contains items that start with the string "/Bob/", and permission to use any of the IAM actions with his own security credentials.
- **Resource-based—**Who has access to a particular resource?

  For example, the example_bucket resource might have a permission that states that Bob, Susan, and DevApp1 have read, write, and list permission.

For more information about permissions, see Overview of AWS IAM Permissions (p. 171).

Policy

A *policy* is a document that provides a formal statement of one or more permissions. With IAM, you can attach a policy to one or more entities (an IAM user, group, or role) to assign the permissions stated in the policy to those entities. You can attach multiple policies to an entity, and you can attach a policy to multiple entities. If you want to attach the same policy to multiple IAM users, we recommend you put the users in a group and attach the policy to the group.

For more information about policies, see Overview of IAM Policies (p. 174) and Managing Policies (p. 192).

Other Permissions Systems

IAM integrates with AWS services so you can control an entity's access to those services' resources. Some of the AWS services that IAM integrates with have their own resource-based permissions systems for giving access to specific resources. How IAM works with the different permissions systems is covered elsewhere in this guide.

For more information, see AWS Services That Support IAM (p. 265).

Accessing IAM

You can work with AWS Identity and Access Management using the following:

- AWS Management Console. The console lets you use a browser-based interface to manage IAM and AWS resources. For more information about accessing IAM through the console, see IAM and the AWS Management Console (p. 34). For a tutorial on using the console, see Creating an Administrators Group Using the Console (p. 20).
- AWS Command Line Interface (AWS CLI). The AWS CLI lets you issue commands at your computer's command line to perform IAM and AWS tasks; this can be faster and more convenient than using the console. The CLI is also useful if you want to build scripts that perform IAM tasks. For information about setting up and using the AWS CLI, see the AWS Command Line Interface User Guide.
- AWS SDKs. AWS provides SDKs that consist of libraries and sample code for various programming languages and platforms (Java, Ruby, .NET, iOS, Android, etc.). The SDKs provide a convenient way to create programmatic access to IAM and AWS. For example, the SDKs take care of tasks such as cryptographically signing requests, managing errors, and retrying requests automatically. For information about the AWS SDKs, including how to download and install them, see the Tools for Amazon Web Services page.
- IAM Query API. Finally, you can access IAM and AWS programmatically using the IAM Query API, which lets you issue HTTPS requests directly. (When you use the Query API, you must include code to digitally sign requests using your credentials.) For more information, see the IAM API Reference.

Using any of these access methods, you can manage IAM resources, such as the performing the following tasks:

- Create groups and assign permissions to groups
- Add users
- Create security credentials for your users
- Assign passwords to your users

Note
In order to access IAM, whether through the console or programmatically, you need security credentials. For information, see AWS Security Credentials in the *AWS General Reference*.

About IAM Entities

Topics

IAM Identifiers

Topics

IAM uses a few different identifiers for users, groups, roles, policies, and server certificates. This section describes the identifiers and when you use each.

Friendly Names and Paths

When you create a user, a role, a group, or a policy, or when you upload a server certificate, you give it a friendly name, such as Bob, TestApp1, Developers, ManageCredentialsPermissions, or ProdServerCert.

If you are using the IAM API or AWS Command Line Interface (AWS CLI) to create IAM entities, you can also optionally give the entity a path. You might use the path to identify which division or part of the organization the entity belongs in. For example: /division_abc/subdivision_xyz/product_1234/engineering/. Examples of how you might use paths are shown in the next section (see IAM ARNs (p. 9)).

Just because you give a user and group the same path doesn't automatically put that user in that group. For example, you might create a Developers group and specify its path as /division_abc/subdivision_xyz/product_1234/engineering/. Just because you create a user named Bob and give him that same path doesn't automatically put Bob in the Developers group.

IAM doesn't enforce any boundaries between users or groups based on their paths. Users with different paths can use the same resources (assuming they've been granted permission to). For information about limitations on names, see Limitations on IAM Entities (p. 13).

IAM ARNs

Most resources have a friendly name (for example, a user named `Bob` or a group named `Developers`). However, the access policy language requires you to specify the resource or resources using the following *Amazon Resource Name (ARN)* format.

```
arn:aws:service:region:account:resource
```

Where:

- `service` identifies the AWS product. For IAM resources, this is always `iam`.
- `region` is the region the resource resides in. For IAM resources, this is always left blank.
- `account` is the AWS account ID with no hyphens (for example, 123456789012).
- `resource` is the portion that identifies the specific resource by name.

You can use ARNs in IAM for users (IAM and federated), groups, roles, policies, instance profiles, virtual MFA devices, and server certificates (p. 162). The following table shows the ARN format for each and an example. The region portion of the ARN is blank because IAM resources are global.

The following examples show ARNs for different types of IAM resources.

```
arn:aws:iam::123456789012:root
arn:aws:iam::123456789012:user/Bob
arn:aws:iam::123456789012:user/division_abc/subdivision_xyz/Bob
arn:aws:iam::123456789012:group/Developers
arn:aws:iam::123456789012:group/division_abc/subdivision_xyz/product_A/Developers
arn:aws:iam::123456789012:role/S3Access
arn:aws:iam::123456789012:policy/ManageCredentialsPermissions
arn:aws:iam::123456789012:instance-profile/Webserver
arn:aws:sts::123456789012:federated-user/Bob
arn:aws:sts::123456789012:assumed-role/Accounting-Role/Mary
arn:aws:iam::123456789012:mfa/BobJonesMFA
arn:aws:iam::123456789012:server-certificate/ProdServerCert
arn:aws:iam::123456789012:server-certificate/division_abc/subdivision_xyz/Prod
ServerCert
```

The following example shows a policy you could assign to Bob to allow him to manage his own access keys. Notice that the resource is Bob himself.

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": ["iam:*AccessKey*"],
   "Resource": "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:user/division_abc/sub
division_xyz/Bob"
  }]
}
```

Note
When you use ARNs to identify resources in an IAM policy, you can include *policy variables* that let you include placeholders for run-time information (such as the user's name) as part of the ARN. For more information, see IAM Policy Variables Overview (p. 232)

You can use wildcards in the *`resource`* portion of the ARN to specify multiple users or groups or policies. For example, to specify all users working on product_1234, you would use:

```
arn:aws:iam::123456789012:user/division_abc/subdivision_xyz/product_1234/*
```

Let's say you have users whose names start with the string `app_`. You could refer to them all with the following ARN.

```
arn:aws:iam::123456789012:user/division_abc/subdivision_xyz/product_1234/app_*
```

To specify all users, groups, or policies in your AWS account, use a wildcard after the `user/`, `group/`, or `policy` part of the ARN, respectively.

```
arn:aws:iam::123456789012:user/*
arn:aws:iam::123456789012:group/*
arn:aws:iam::123456789012:policy/*
```

Don't use a wildcard in the `user/`, `group/`, or `policy` part of the ARN. In other words, the following is not allowed:

```
arn:aws:iam::123456789012:u*
```

Example 1: Use of Paths and ARNs for Distributed Administrator Groups

Dave is the main administrator in Example Corp., and he decides to use paths to help delineate the users in the company and set up a separate administrator group for each path-based division. Following is a subset of the full list of paths he plans to use:

- /marketing/
- /sales/
- /legal/

Dave creates an administrator group for the marketing part of the company and calls it Marketing_Admin. He assigns it the /marketing/ path. The group's ARN is

```
arn:aws:iam::123456789012:group/marketing/Marketing_Admin
```

Dave assigns the following policy to the Marketing_Admin group that gives the group permission to use all IAM actions with all groups and users in the /marketing/ path. The policy also gives the Marketing_Admin group permission to perform any Amazon S3 actions on the objects in the portion of the corporate bucket dedicated to the marketing employees in the company.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iam:*",
      "Resource": [
        "arn:aws:iam::123456789012:group/marketing/*",
        "arn:aws:iam::123456789012:user/marketing/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "s3:*",
      "Resource": "arn:aws:s3:::example_bucket/marketing/*"
    },
    {
      "Effect": "Allow",
      "Action": "s3:ListBucket*",
      "Resource": "arn:aws:s3:::example_bucket",
      "Condition": {"StringLike": {"s3:prefix": "marketing/*"}}
    }
  ]
}
```

The policy has a separate statement that is necessary to let the group list the objects only in the portion of the bucket dedicated to the marketing group. For more information about constructing policies to control user and group access to Amazon S3, see Using IAM Policies in the *Amazon Simple Storage Service Developer Guide*.

Dave then creates a user named Jules in the /marketing/ path, and assigns Jules to the Marketing_Admin group. Jules can now create and manage new users and groups in the /marketing/ path, and work with the objects in the marketing part of the bucket.

Dave then sets up similar administrator groups for the other paths (for example, /sales/, etc.).

Example 2: Use of Paths and ARNs for a Project-Based Group

In this example, Jules in the Marketing_Admin group creates a project-based group within the /marketing/ path, and assigns users from different parts of the company to the group. This example illustrates that a user's path isn't related to the groups the user is in.

The marketing group has a new product they'll be launching, so Jules creates a new group in the /marketing/ path called Widget_Launch. Jules then assigns the following policy to the group, which gives the group access to objects in the part of the example_bucket designated to this particular launch.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:*",
      "Resource": "arn:aws:s3:::example_bucket/marketing/newproductlaunch/wid
get/*"
    },
    {
      "Effect": "Allow",
      "Action": "s3:ListBucket*",
      "Resource": "arn:aws:s3:::example_bucket",
     "Condition": {"StringLike": {"s3:prefix": "marketing/newproductlaunch/wid
get/*"}}
    }
  ]
}
```

Jules then assigns the users who are working on this launch to the group. This includes Patricia and Eli from the /marketing/ path. It also includes Chris and Chloe from the /sales/ path, and Aline and Jim from the /legal/ path.

Unique IDs

When IAM creates a user, group, role, policy, instance profile, or server certificate, it assigns to each entity a unique ID that looks like the following example:

`AIDAJQABLZS4A3QDU576Q`

For the most part, you use friendly names and ARNs when you work with IAM entities, so you don't need to know the unique ID for a specific entity. However, the unique ID can sometimes be useful when it isn't practical to use friendly names.

One example pertains to reusing friendly names in your AWS account. Within your account, a friendly name for a user, group, or policy must be unique. For example, you might create an IAM user named David. Your company uses Amazon S3 and has a bucket with folders for each employee; the bucket has a resource-based policy (a bucket policy) that lets users access only their own folders in the bucket. Suppose that the employee named David leaves your company and you delete the corresponding IAM user. But later another employee named David starts and you create a new IAM user named David. If the bucket policy specifies the IAM user named David, the policy could end up granting the new David access to information in the Amazon S3 bucket that was left by the former David.

However, every IAM user has a unique ID, even if you create a new IAM user that reuses a friendly name that you deleted before. In the example, the old IAM user David and the new IAM user David have different unique IDs. If you create resource policies for Amazon S3 buckets that grant access by unique ID and not just by user name, it reduces the chance that you could inadvertently grant access to information that an employee should not have.

Another example where user IDs can be useful is if you maintain your own database (or other store) of IAM user information. The unique ID can provide a unique identifier for each IAM user you create, even if over time you have IAM users that reuse a name, as in the previous example.

Getting the Unique ID

The unique ID for an IAM entity is not available in the IAM console. To get the unique ID, you can use the following AWS CLI commands or IAM API calls.

AWS CLI:

- get-group
- get-role
- get-user
- get-policy
- get-instance-profile
- get-server-certificate

IAM API:

- GetGroup
- GetRole
- GetUser
- GetPolicy
- GetInstanceProfile
- GetServerCertificate

Limitations on IAM Entities

This section lists restrictions on IAM entities, and describes how to get information about entity usage and IAM quotas.

> **Note**
> To get account-level information about entity usage and quotas, use the GetAccountSummary API action or the get-account-summary AWS CLI command.

The following are restrictions on names:

- Policy documents can contain only the following Unicode characters: horizontal tab (x09), linefeed (x0A), carriage return (x0D), and characters in the range x20 to xFF.
- Names of users, groups, roles, policies, instance profiles, and server certificates must be alphanumeric, including the following common characters: plus (+), equal (=), comma (,), period (.), at (@), underscore (_), and hyphen (-).
- Path names must begin and end with a forward slash (/).
- Policy names for inline policies (p. 179) must be unique to the user, group, or role they are embedded in, and can contain any Basic Latin (ASCII) characters, minus the following reserved characters: backward slash (\), forward slash (/), asterisk (*), question mark (?), and white space. These characters are reserved according to RFC 3986.
- User passwords (login profiles) can contain any Basic Latin (ASCII) characters.
- AWS account ID aliases must be unique across AWS products, and must be alphanumeric following DNS naming conventions. An alias must be lowercase, it must not start or end with a hyphen, it cannot contain two consecutive hyphens, and it cannot be a 12 digit number.

For a list of Basic Latin (ASCII) characters, go to the Library of Congress Basic Latin (ASCII) Code Table.

The following are the default maximums for your entities:

- Groups per AWS account: 100
- Users per AWS account: 5000
  If you need to add a large number of users, consider using temporary security credentials. For more information about temporary security credentials, go to Using Temporary Security Credentials.
- Roles per AWS account: 250
- Instance profiles per AWS account: 100
- Roles per instance profiles: 1 (each instance profile can contain only 1 role)
- Number of groups per user: 10 (that is, the user can be in this many groups)
- Access keys per user: 2
- Signing certificates per user: 2
- MFA devices in use per user: 1
- MFA devices in use per AWS account (at the root account level): 1
- Virtual MFA devices (assigned or unassigned) per AWS account: equal to the user quota for the account
- Server certificates per AWS account: 20
- AWS account aliases per AWS account: 1
- Login profiles per user: 1
- SAML providers per account: 100
- Identity providers (IdPs) per SAML provider: 10
- Keys per SAML provider: 10
- Customer managed policies per AWS account: 1000
- Versions per managed policy: 5
- Managed policies attached per IAM user, group, or role: 2

You can request to increase some of these quotas for your AWS account on the IAM Limit Increase Contact Us Form. Currently you can request to increase the limit on users per AWS account, groups per AWS account, roles per AWS account, instance profiles per AWS account, and server certificates per AWS account.

The following are the maximum lengths for entities:

- Path: 512 characters
- User name: 64 characters
- Group name: 128 characters
- Role name: 64 characters
- Instance profile name: 128 characters
- Unique ID (applicable to users, groups, roles, managed policies, and server certificates): 32 characters
- Policy name: 128 characters
- Certificate ID: 128 characters
- Login profile password: 1 to 128 characters
- AWS account ID alias: 3 to 63 characters.
- Role trust policy (the policy that determines who is allowed to assume the role): 2,048 characters
- For inline policies (p. 179): You can add as many inline policies as you want to a user, role, or group, but the total aggregate policy size (the sum size of all inline policies) per entity cannot exceed the following limits:
  - User policy size cannot exceed 2,048 characters

  - Role policy size cannot exceed 10,240 characters
  - Group policy size cannot exceed 5,120 characters

    Note
    IAM does not count whitespace when calculating the size of a policy against these limitations.
- For managed policies (p. 179): You can add up to two managed policies to a user, role, or group. The size of each managed policy cannot exceed 5,120 characters.

  Note
  IAM does not count whitespace when calculating the size of a policy against this limitation.

Getting Set Up

AWS Identity and Access Management (IAM) helps you securely control access to Amazon Web Services (AWS) and your account resources. IAM can also keep your account credentials private. With IAM, you can create multiple IAM users under the umbrella of your AWS account or enable temporary access through identity federation with your corporate directory. In some cases, you can also enable access to resources across AWS accounts.

Without IAM, however, you must either create multiple AWS accounts—each with its own billing and subscriptions to AWS products—or your employees must share the security credentials of a single AWS account. In addition, without IAM, you cannot control the tasks a particular user or system can do and what AWS resources they might use.

This guide provides a conceptual overview of IAM, describes business use cases, and explains AWS permissions and policies.

Topics

Using IAM to Give Users Access to Your AWS Resources

Here are the ways you can use IAM to control access to your AWS resources.

Type of access	Why would I use it?	Where can I get more information?
Access for users under your AWS account	You want to add users under the umbrella of your AWS account, and you want to use IAM to create users and manage their permissions.	To learn how to use the AWS Management Console to create users and to manage their permissions under your AWS account, see Getting Started (p. 19). To learn about using the IAM API or AWS Command Line Interface to create users under your AWS account, see Creating an Administrators Group Using the AWS Command Line Interface (AWS CLI) (p. 21). For more information about working with IAM users, see IAM Users and Groups (p. 39).
Non-AWS user access via identity federation between your authorization system and AWS	You have non-AWS users in your identity and authorization system, and they need access to your AWS resources.	To learn how to use security tokens to give your users access to your AWS account resources through federation with your corporate directory, go to Using Temporary Security Credentials. For information about the AWS Security Token Service API, go to the AWS Security Token Service API Reference.
Cross-account access between AWS accounts	You want to share access to certain AWS resources with users under other AWS accounts.	To learn how to use IAM to grant permissions to other AWS accounts, see Roles - Terms and Concepts (p. 116).

Do I Need to Sign Up for IAM?

If you don't already have an AWS account, you need to create one to use IAM. You don't need to specifically sign up to use IAM. There is no charge to use IAM.

Note
IAM works only with AWS products that are integrated with IAM. For a list of services that support IAM, see AWS Services That Support IAM (p. 265).

To sign up for AWS

1. Open http://www.amazonaws.cn/, and then click **Sign Up**.
2. Follow the on-screen instructions.

   Part of the sign-up procedure involves receiving a phone call and entering a PIN using the phone keypad.

Additional Resources

Here are some resources to help you get things done with IAM.

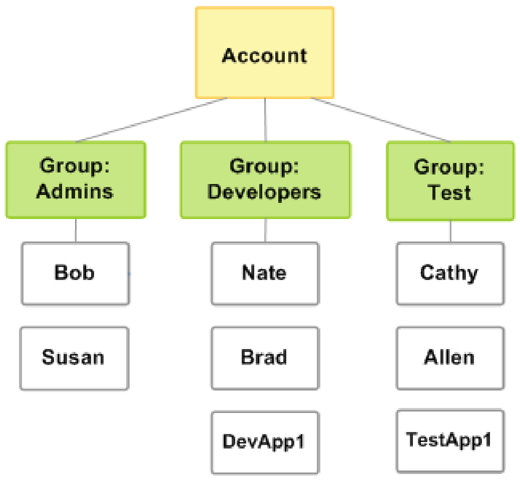
- Manage your AWS account credentials: AWS Security Credentials in the *AWS General Reference*
- Get started with IAM: What Is IAM? (p. 1)
- Learn more about how IAM works: What Is IAM? (p. 1)

- Set up a command-line interface (CLI) to use with IAM: AWS Command Line Interface User Guide
- Download an AWS SDK for convenient programmatic access to IAM: Tools for Amazon Web Services
- Get the release notes: Release Notes
- Get the FAQ: AWS Identity and Access Management FAQ
- Get technical support: AWS Support Center
- Get premium technical support: AWS Premium Support Center
- Find definitions of AWS terms: Amazon Web Services Glossary
- Get community support: IAM Discussion Forums
- Contact AWS: Contact Us

Getting Started

This topic shows you how to give access to your AWS resources by creating users under your AWS account. First, you'll learn concepts you should understand before you create groups and users, and then you'll walk through how to perform the necessary tasks using the AWS Management Console. The first task is to set up an administrators group for your AWS account. Having an administrators group for your AWS account isn't required, but we strongly recommend it.

The following figure shows a simple example of an AWS account with three groups. A group is a collection of users who have similar responsibilities. In this example, one group is for administrators (it's called *Admins*). There's also a *Developers* group and a *Test* group. Each group has multiple users. Each user can be in more than one group, although the figure doesn't illustrate that. You can't put groups inside other groups. You use policies to grant permissions to groups.



In the procedure that follows, you will perform the following tasks:

- Create an Administrators group and give the group permission to access all of your AWS account's resources.
- Create a user for yourself and add that user to the Administrators group.
- Create a password for your user so you can sign in to the AWS Management Console.

You will grant the Administrators group permission to access all your available AWS account resources. Available resources are any AWS products you use, or that you are signed up for. Users in the Administrators group can also access your AWS account information, *except* for your AWS account's security credentials.

Topics

Creating an Administrators Group Using the Console

This procedure describes how to create an IAM group named Administrators, grant the group full permissions for all AWS services, create an IAM user for yourself, and add the user to the Administrators group.

To create the Administrators group

1. Sign in to the AWS Management Console and open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, click **Groups**, then click **Create New Group**.
3. In the **Group Name** box, type `Administrators` and then click **Next Step**.
4. In the list of policies, select the check box next to the **AdministratorAccess** policy. You can use the **Filter** menu and the **Search** box to filter the list of policies.
5. Click **Next Step**, then click **Create Group**.

Your new group is listed under **Group Name**.

To create an IAM user for yourself, add the user to the Administrators group, and create a password for the user

1. In the navigation pane, click **Users** and then click **Create New Users**.
2. In box **1**, enter a user name. Clear the check box next to **Generate an access key for each user**, then click **Create**.
3. In the list of users, click the name (not the check box) of the user you just created. You can use the **Search** box to search for the user name.
4. In the **Groups** section, click **Add User to Groups**.
5. Select the check box next to the **Administrators** group, then click **Add to Groups**.
6. Scroll down to the **Security Credentials** section. Under **Sign-In Credentials**, click **Manage Password**.
7. Select **Assign a custom password**, then enter a password in the **Password** and **Confirm Password** boxes. When you are finished, click **Apply**.

You have created an IAM group named Administrators, granted full permissions for all AWS services to the group, created an IAM user for yourself, and added the user to the group. You can use the same process to create more groups and users, and to give your users access to your AWS account resources. To learn about using policies to restrict users' permissions to specific AWS resources, go to Permissions and Policies (p. 171) and Example Policies for Administering AWS Resources (p. 203).

Creating an Administrators Group Using the AWS Command Line Interface (AWS CLI)

The first thing we recommend you do after creating an AWS account is to set up an administrators group. It's not required, but is highly recommended. Going forward, the users in the administrators group should set up the groups, users, and so on, for the AWS account, and all future key-based interaction should be through the AWS account's users and their own keys.

Tip
If you read through Getting Started (p. 19), you used the AWS Management Console to set up an administrators group for your AWS account. We've repeated the information here if you're interested in using a different interface than the one presented in *Getting Started*.

Process for Setting Up an Administrators Group

1. Create a group and give it a name (for example, *Admins*). For more information, see Creating a Group (p. 21).
2. Attach a policy that gives the group administrative permissions—access to all AWS actions and resources. For more information, see Attaching a Policy to the Group (p. 22).
3. Add at least one user to the group. For more information, see Creating an IAM User in Your AWS Account (p. 43).

Creating a Group

This section shows how to create a group in the IAM system. For information about the limitations on the group's name and the maximum number of groups you can have, see Limitations on IAM Entities (p. 13).

To create an administrators group

1. Enter the `aws iam create-group` command with the name you've chosen for the group. Optionally, you can include a path as part of the group name. For more information about paths, see Friendly Names and Paths (p. 9).

   In this example, you create a group named *Admins*.

   ```
   aws iam create-group --group-name Admins
   ```

   Sample response:

   ```
   {
       "Group": {
           "Path": "/",
           "CreateDate": "2014-06-05T20:29:53.622Z",
           "GroupId":"ABCDEFGHABCDEFGHABCDE",
           "Arn": "arn:aws:iam::123456789012:group/Admins",
           "GroupName": "Admins"
       }
   }
   ```

2. Enter the `aws iam list-groups` command to list the groups in your AWS account and confirm the group was created.

```
aws iam list-groups
```

Sample response:

```
{
    "Groups": [
        {
            "Path": "/",
            "CreateDate": "2014-06-05T20:29:53.622Z",
            "GroupId":"ABCDEFGHABCDEFGHABCDE",
            "Arn": "arn:aws:iam::123456789012:group/Admins",
            "GroupName": "Admins"
        }
    ]
}
```

The response includes the Amazon Resource Name (ARN) for your new group. The ARN is a standard format that AWS uses to identify resources. The 12-digit number in the ARN is your AWS account ID. The friendly name you assigned to the group (Admins) appears at the end of the group's ARN.

Attaching a Policy to the Group

This section shows how to attach a policy that lets any user in the group perform any action on any resource in the AWS account. You do this by attaching the AWS managed policy (p. 179) called AdministratorAccess to the Admins group. For more information about policies, see Permissions and Policies (p. 171).

To add a policy giving full administrator permissions

1. Enter the `aws iam attach-group-policy` command to attach the policy called AdministratorAccess to your Admins group. The command uses the ARN of the AWS managed policy called AdministratorAccess.

   ```
   aws iam attach-group-policy --group-name Admins --policy-arn
   arn:aws:iam::aws:policy/AdministratorAccess
   ```

   If the command is successful, there's no response.
2. Enter the `aws iam list-attached-group-policies` command to confirm the policy is attached to the Admins group.

   ```
   aws iam list-attached-group-policies --group-name Admins
   ```

   The response lists the names of the policies attached to the Admins group. A response like the following tells you that the policy named AdministratorAccess has been attached to the Admins group:

   ```
   {
       "AttachedPolicies": [
           {
               "PolicyName": "AdministratorAccess",
               "PolicyArn": "arn:aws:iam::aws:policy/AdministratorAccess"
           }
       ],
   ```

```
    "IsTruncated": false
}
```

You can confirm the contents of a particular policy with the `aws iam get-policy` command.

Important
After you have the administrators group set up, you must add at least one user to it. For more information about adding users to a group, see Creating an IAM User in Your AWS Account (p. 43).

How Users Sign In to Your Account

After you have created IAM users and created passwords for them, users can sign in to the AWS Management Console for your AWS account by using a special URL, which has this format:

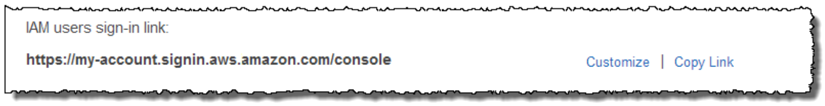
```
https://AWS-account-ID.signin.www.amazonaws.cn/console
```

For example, the URL might look like this:

```
https://123456789012.signin.www.amazonaws.cn/console
```

By default, the sign-in URL for your account includes your account ID. You can create a unique sign-in URL for your account so that the URL includes a name instead of an account ID. For more information, see Your AWS Account ID and Its Alias (p. 36).

You can also find the sign-in URL for an account on the IAM console dashboard.


IAM users sign-in link:
https://my-account.signin.aws.amazon.com/console

Tip
To create a bookmark for your account's unique sign-in page in your web browser, you should manually enter your account's sign-in URL in the bookmark entry. Don't use your web browser's "bookmark this page" feature.

IAM users in your account have access only to the AWS resources that you have granted them permissions to, using a policy that is attached to the user or to an IAM group that the user belongs to. To work in the console, users must have permissions to perform the actions that the console performs, such as listing and creating AWS resources. For more information, see Permissions and Policies (p. 171) and Example Policies for Administering AWS Resources (p. 203).

Note
If your organization has an existing identity system, you might want to create a single sign-on (SSO) option that gives users access to the AWS Management Console for your account without requiring them to have an IAM user identity and without requiring them to sign in separately to your organization's site and to AWS. For more information, see Giving Federated Users Direct Access to the AWS Management Console in the *Using Temporary Security Credentials* guide.

If users need programmatic access to work with your account, you can create an access key (an access key ID and a secret access key) for each user, as described in Creating, Modifying, and Viewing User Access Keys (AWS Management Console) (p. 68).

Where to Go Next

Topics

The Getting Started chapter shows you how to use AWS Identity and Access Management to create groups and users within your AWS account, but IAM also enables you to do many tasks not covered there. This section provides links to additional resources that will help you deepen your understanding of permissions and how to use IAM with other AWS products to control users' access.

Learn More About IAM

To learn more about IAM, read the following sections:

- Business Use Cases (p. 30) describes how you might use IAM with other AWS products.
- IAM Users and Groups (p. 39) has details about managing users and groups in your AWS account.
- AWS Services That Support IAM (p. 265) links to information about how IAM works with other AWS products (for example, Amazon EC2 or Amazon S3), and what you can do to control your users' access to specific AWS products and resources.
- *Friendly Names and Paths* in IAM Identifiers (p. 8) explains how to use *paths* to logically separate groups and users based on organizational structure.

Learn About Policies and Permissions

To learn more about IAM policies and permissions, refer to the following resources:

- Permissions and Policies (p. 171) describes permissions, policies, and the access policy language you use to write policies.
- Example Policies for Administering AWS Resources (p. 203) provides examples of policies that control access for your IAM resources.
- The AWS Policy Generator is a tool you can use to help you create custom policies.

Other Ways to Access IAM

This chapter has shown you how to create users and groups using the AWS Management Console. You can continue using IAM through the console, or you can try one of the other interfaces.

Use the AWS Command Line Interface

If you want to issue commands with the AWS Command Line Interface, go to the AWS Command Line Interface User Guide. The guide describes how to set up and use the AWS CLI, describes the commands, and lists available commands by function.

Use an Existing Library

AWS offers SDKs that let you access IAM programmatically to create groups, users, and so on. There are AWS SDKs for Java, .NET, PHP, Android, iOS, Python, Ruby and other languages and platforms. For more information, go to Tools for Amazon Services.

Code Directly to the Web Service API

If you want to write code directly to the IAM Query API, go to IAM API Reference. The guide describes how to create and authenticate API requests. Making Query Requests (p. 278) describes the basics about using the Query API.

IAM Resources

Use the following resources to learn more about how to work with this service.

- **What Is IAM? (p. 1)** – Describes how to use the service and all its features through the AWS Management Console, the CLI, or API.
- **AWS Command Line Interface User Guide** – Gives complete descriptions of the commands in the CLI.
- **IAM API Reference** – Gives complete descriptions of the API actions, parameters, and data types; and a list of errors that the service returns.

- **AWS Developer Tools** – Links to developer tools and resources that provide documentation, code samples, release notes, and other information to help you build innovative applications with AWS.
- **AWS Support Center** – The hub for creating and managing your AWS Support cases. Also includes links to other helpful resources, such as forums, technical FAQs, service health status, and AWS Trusted Advisor.
- **AWS Support** – The primary web page for information about AWS Support, a one-on-one, fast-response support channel to help you build and run applications in the cloud.
- **Contact Us** – A central contact point for inquiries concerning AWS billing, account, events, abuse, and other issues.
- **AWS Site Terms** – Detailed information about our copyright and trademark; your account, license, and site access; and other topics.

IAM Best Practices and Use Cases

This section provides practical guidance for working with IAM. Here you can learn what the recommended best practices are for when and how to use IAM. You can also see how IAM is used in real-world scenarios to work with other AWS services.

Topics

IAM Best Practices

Topics

This article presents a list of suggestions for using the AWS Identity and Access Management (IAM) service to help secure your IAM resources.

Lock away your AWS account access keys

Any request you make for an AWS resource requires credentials so that AWS can make sure you have permission to access the resource. You can use access keys (an access key ID and secret access key) to make programmatic requests to AWS. However, we recommend that you do not use your AWS account access keys. The access key ID and secret access key for your AWS account give you access to all your resources, including your billing information. (For more information about credentials for working with AWS, see Types of Security Credentials in the *Amazon Web Services General Reference*.)

Therefore, protect your AWS account credentials like you would your credit card numbers or any other secret in these ways:

- If you don't already have a set of access keys for your account, don't create them unless you absolutely need them. Instead, use your account password to sign in to the AWS Management Console and create an IAM user for yourself that has administrative privileges—see the next section.
- If you do have a set of AWS access keys for your account, delete them. If you want to keep them, make sure that you rotate (change) the access keys credentials regularly. To delete or rotate your AWS account access keys, go to https://console.amazonaws.cn//iam/home?#security_credential and sign in with your account's email address and password. You can manage your access keys in the **Access Keys** section.
- Never share your AWS account password or access keys with anyone. The remaining sections of this document discuss various ways to avoid having to share your account credentials with other users and to avoid having to embed them in an application.
- Use a strong password to help protect account-level access to the AWS Management Console. For information about managing your AWS account password, see Changing Your AWS Account Password (p. 58).
- Enable AWS multi-factor authentication (MFA) on your AWS account. For more information, see Using Multi-Factor Authentication (MFA) Devices with AWS (p. 69).

Create individual IAM users

Don't use your AWS root account credentials to access AWS, and don't give your credentials to anyone else. Instead, create individual users (using the IAM console, the APIs, or the command line interface) for anyone who needs access to your AWS account. Create an IAM user for yourself as well, give that IAM user administrative privileges, and use that IAM user for all your work.

By creating individual IAM users for people accessing your account, you can give each IAM user a unique set of security credentials. You can also grant different permissions to each IAM user. If necessary, you can change or revoke an IAM user's permissions any time. (If you give out your AWS root credentials, it can be difficult to revoke them.)

Note
Before you set permissions for individual IAM users, though, see the next point about groups.

Use groups to assign permissions to IAM users

Instead of defining permissions for individual IAM users, it's usually more convenient to create groups that relate to job functions (`Admins`, `Developers`, `Accounting`, etc.), define the relevant permissions for each group, and then assign IAM users to those groups. All the users in an IAM group share the same permissions. That way, you can make changes for everyone in a group in just one place. As people move around in your company, you can simply change what IAM group their IAM user belongs to.

For more information, see the following:

- What Is IAM? (p. 1)
- Creating an Administrators Group Using the AWS Command Line Interface (AWS CLI) (p. 21)
- Adding and Removing Users in an IAM Group (p. 51)

Grant least privilege

When you create IAM policies, follow the standard security advice of granting *least privilege*—that is, granting only the permissions required to perform a task. Determine what users need to do and then craft policies for them that let the users perform *only* those tasks. Similarly, create policies for individual resources (such as Amazon S3 buckets) that identify precisely who is allowed to access the resource, and allow only the minimal permissions for those users. For example, perhaps developers should be allowed to write to an Amazon S3 bucket, but testers only need to read from it.

It's more secure to start with a minimum set of permissions and grant additional permissions as necessary, rather than starting with permissions that are too lenient and then trying to tighten them later.

Defining the right set of permissions requires some research to determine what is required for the specific task, what actions a particular service supports, and what permissions are required in order to perform those actions.

A good way to start is to use the built-in AWS managed policies in the console. These policies include predefined permissions for common use cases (administrator, power user, etc.) in individual services.

You can also create custom policies that set permissions precisely. When you do, you need to be familiar with how Allow and Deny work in policies, and how policies are evaluated when more than one policy is in force during a request for a resource. (In cases where the right combination of permissions is complex, it can be tempting to loosen up the permissions—even to grant "all access" privileges—to make sure users have access to resources. But we do not recommend this approach; as noted, standard security advice is to grant least privilege.) You'll also need to test thoroughly to make sure that users can do their work and that policies are working as you intend.

For more information, see the following:

- Permissions and Policies (p. 171)
- IAM Policy Evaluation Logic (p. 243)
- Policy topics for individual services, which provide examples of how to write policies for service-specific resources. Examples:
  - Controlling Access to Amazon DynamoDB Resources in the *Amazon DynamoDB Developer Guide*
  - Using IAM Policies in the *Amazon Simple Storage Service Developer Guide*
  - Access Control List (ACL) Overview in the *Amazon Simple Storage Service Developer Guide*

Configure a strong password policy for your users

If you choose to allow users to change their own passwords, require that they create strong passwords, and that they rotate their passwords periodically. On the **Account Settings** page of the IAM console, you can create a password policy for your account. You can use the password policy to define password requirements, such as minimum length, whether it requires a non-alphabetic character, how frequently it must be rotated, and so on.

For more information, see Setting an Account Password Policy for IAM Users (p. 59).

Enable MFA for privileged users

For extra security, enable multi-factor authentication (MFA) for privileged IAM users (users who are allowed access to sensitive resources). With MFA, users have a device that generates a unique authentication code (a one-time password, or OTP) and users must provide both their normal credentials (like their user name and password) and the OTP. The MFA device can either be a special piece of hardware, or it can be a virtual device, such as an app that runs on a smartphone.

For more information, see Using Multi-Factor Authentication (MFA) Devices with AWS (p. 69).

Use roles for applications that run on Amazon EC2 instances

Applications that run on an Amazon EC2 instance need credentials in order to access other AWS services. To provide credentials to the application in a secure way, use IAM *roles*. A role is an entity that has its own set of permissions, but that isn't a user or group. Roles also don't have their own permanent set of credentials the way IAM users do. Instead, a role is *assumed* by other entities. Credentials are then either associated with the assuming identity, or IAM dynamically provides temporary credentials (in the case of Amazon EC2).

When you launch an Amazon EC2 instance, you can specify a role for the instance as a launch parameter. Applications that run on the EC2 instance can use the role's credentials when they access AWS resources. The role's permissions determine what the application is allowed to do.

For more information, see Using IAM Roles to Delegate Permissions to Applications that Run on Amazon EC2 (p. 140).

Delegate by using roles instead of by sharing credentials

You might need to allow users from another AWS account to access resources in your AWS account. If so, don’t share security credentials, such as access keys, between accounts. Instead, use IAM roles. You can define a role that specifies what permissions the IAM users in the other account are allowed, and from which AWS accounts IAM users are allowed to assume the role.

For more information, see Roles - Terms and Concepts (p. 116).

Rotate credentials regularly

Change your own passwords and access keys regularly, and make sure that all IAM users in your account do as well. That way, if a password is compromised without your knowledge, you limit how long the password can be used to access your resources. You can apply a password policy to your account to require all your IAM users to rotate their passwords, and you can choose how often they must do so.

For more information about setting a password policy in your account, see Setting an Account Password Policy for IAM Users (p. 59).

For more information about rotating credentials, see Rotating Credentials (p. 92).

Remove unnecessary credentials

Remove IAM user credentials (that is, passwords and access keys) that are not needed. For example, an IAM user that is used for an application does not need a password (passwords are necessary only to sign in to AWS websites). Similarly, if a user does not and will never use access keys, there's no reason for the user to have them.

You can generate and download a credential report that lists all IAM users in your account and the status of their various credentials, including passwords, access keys, MFA devices, and signing certificates. For passwords, the credential report shows how recently a password has been used. Passwords that have not been used recently might be good candidates for removal.

For more information about IAM credential reports, see Getting Credential Reports for Your AWS Account (p. 93).

Use policy conditions for extra security

To the extent that it's practical, define the conditions under which the policy allows access to a resource. For example, you can write conditions to specify a range of allowable IP addresses for a request, or you can specify that a request is allowed only within a specified date range or time range. You can also set conditions that require the use of SSL or MFA. For example, you can require that a user has authenticated with an MFA device in order to be allowed to terminate an Amazon EC2 instance. The more explicitly you can define when resources are available (and otherwise unavailable), the safer your resources will be.

For more information, see Condition (p. 219).

Keep a history of activity in your AWS account

Take advantage of the logging features available in AWS services. By examining log files of activity in your AWS account, you can see what actions users have taken in your account and what resources have been used, along with details like the time and date of actions, what the source IP was for an action, which actions failed due to inadequate permissions, and so on.

Logging is supported in multiple ways. AWS CloudTrail provides logging across AWS services. Amazon S3 and Amazon CloudFront have their own logging solutions. Log files about activity in your AWS account are written to Amazon S3 buckets.

For more information, see the following topics in the AWS documentation:

- AWS CloudTrail User Guide
- Server Access Logging in the *Amazon Simple Storage Service Developer Guide*.
- Access Logs in the *Amazon CloudFront Developer Guide*.

Video presentation about IAM best practices

The following video includes a conference presentation that covers these best practices and shows additional details about how to work with the features discussed here.

AWS re:Invent 2014 | (SEC305) IAM Best Practices

Business Use Cases

Topics

This section describes a simple business use case for IAM to help you understand basic ways you might implement the service to control the AWS access your users have. The use case is described at a high level, without the mechanics of how you'd use the IAM API to achieve the results you want.

The use case centers on a fictional company called Example Corp. After setting up an AWS account for Example Corp., we show two typical examples of how the company might use IAM—first, with Amazon Elastic Compute Cloud (Amazon EC2), and then with Amazon Simple Storage Service (Amazon S3).

For more information about using IAM with other services from AWS, including how to implement individual APIs, see AWS Services That Support IAM (p. 265).

Initial Setup of Example Corp.

Joe is the founder of Example Corp. Upon starting the company, he creates his own AWS account and he uses AWS products by himself. Then he hires employees to work as developers, admins, testers, managers, and system administrators.

Joe uses the IAM API with the AWS account's security credentials to create a user for himself called Joe, and a group called *Admins*. He gives the Admins group the permissions it needs to administer users, groups, and permissions for the AWS account, and he gives the Admins group permissions to perform all actions on all the AWS account's resources (for example, root privileges).

At this point, Joe can stop using the AWS account's credentials to interact with AWS, and instead he begins using only his user credentials.

Joe also creates a group called `AllUsers` so he can easily apply any account-wide permissions to all users in the AWS account. He adds himself to the group. He then creates a group called `Developers`, a group called `Testers`, a group called `Managers`, and a group called `SysAdmins`. He creates users for each of his employees, and puts the users in their respective groups. He also adds them all to the AllUsers group.

For information about how to set up an Admins group, see Creating an Administrators Group Using the AWS Command Line Interface (AWS CLI) (p. 21). For information about creating users, see Creating an IAM User in Your AWS Account (p. 43).

Use Case for IAM with Amazon EC2

This use case illustrates how Example Corp. uses IAM with Amazon EC2. To understand this part of the use case, you need to have a basic understanding of Amazon EC2. For more information about Amazon EC2, go to the Amazon EC2 User Guide for Linux Instances.

Amazon EC2 Permissions for the Groups

To provide "perimeter" control, Joe attaches a policy to the `AllUsers` group that denies any AWS request from a user if the originating IP address is outside the Example Corp.'s corporate network.

At Example Corp., different groups require different permissions:

- **System Administrators**—Need permission to create and manage AMIs, instances, snapshots, volumes, security groups, and so on. Joe attaches a policy to the `SysAdmins` group that gives members of the group permission to use all the Amazon EC2 actions.
- **Developers**—Need the ability to work with instances only. Joe therefore attaches a policy to the `Developers` group that allows developers to call `DescribeInstances`, `RunInstances`, `StopInstances`, `StartInstances`, and `TerminateInstances`.

  Note
  Amazon EC2 uses SSH keys, Windows passwords, and security groups to control who has access to specific Amazon EC2 instances. There's no method in the IAM system to allow or deny access to a specific instance.

- **Managers**—Should not be able to perform any EC2 actions except listing the Amazon EC2 resources currently available. Therefore, Joe attaches a policy to the `Managers` group that only lets them call Amazon EC2 "Describe" APIs.

For examples of what these policies might look like, see Example Policies for Administering AWS Resources (p. 203) and Using AWS Identity and Access Management in the *Amazon EC2 User Guide for Linux Instances*.

User's Role Change

At some point, one of the developers, Don, changes roles and becomes a manager. Joe moves Don from the `Developers` group to the `Managers` group. Now that he's in the `Managers` group, Don's ability to interact with Amazon EC2 instances is limited. He can't launch or start instances. He also can't stop or terminate existing instances, even if he was the user who launched or started the instance. He can list only the instances that Example Corp. users have launched.

Use Case for IAM with Amazon S3

This use case illustrates how Example Corp. uses IAM with Amazon S3. Joe has created an Amazon S3 bucket for the company called `example_bucket`.

Creation of Other Users and Groups

As employees, Don and Mark each need to be able to create their own data in the company's bucket, as well as read and write shared data that all developers will work on. To enable this, Joe logically arranges the data in `example_bucket` using an Amazon S3 key prefix scheme as shown in the following figure.

```
/example_bucket
    /home
        /don
        /mark
    /share
        /developers
        /managers
```

Joe divides the master `/example_bucket` into a set of home directories for each employee, and a shared area for groups of developers and managers.

Now Joe creates a set of policies to assign permissions to the users and groups:

- **Home directory access for Don:** Joe attaches a policy to Don that lets him read, write, and list any objects with the Amazon S3 key prefix `/example_bucket/home/don/`
- **Home directory access for Mark:** Joe attaches a policy to Mark that lets him read, write, and list any objects with the Amazon S3 key prefix `/example_bucket/home/mark/`
- **Shared directory access for the Developers group:** Joe attaches a policy to the group that lets developers read, write, and list any objects in `/example_bucket/share/developers/`
- **Shared directory access for the Managers group:** Joe attaches a policy to the group that lets managers read, write, and list objects in `/example_bucket/share/managers/`

Note
Amazon S3 doesn't automatically give a user who creates a bucket or object permission to perform other actions on that bucket or object. Therefore, in your IAM policies, you must explicitly give users permission to use the Amazon S3 resources they create.

The preceding set of policies clearly defines the actions and resources available in IAM bucket policies or bucket Access Control Lists (ACLs) when anyone in the company attempts to work on data in the corporate space. For examples of what these policies might look like, see Access Control in the *Amazon Simple Storage Service Developer Guide*. For information on how policies are evaluated at run time, see IAM Policy Evaluation Logic (p. 243).

User's Role Change

At some point, one of the developers, Don, changes roles and becomes a manager. We'll assume he no longer needs access to the documents in the `share/developers` directory. Joe, as an admin, moves Don to the `Managers` group and out of the `Developers` group. With just that simple reassignment, Don automatically gets all permissions granted to the `Managers` group, but can no longer access data in the `share/developers` directory.

Integration with a Third-Party Business

Organizations often work with partner companies, consultants, and contractors. Example Corp. has a partner called the Widget Company, and a Widget Company employee named Nate needs to put data into a bucket for Example Corp.'s use. Joe creates a group called WidgetCo and a user named `Nate` and adds Nate to the WidgetCo group. Joe also creates a special bucket called `example_partner_bucket` for Nate to use.

Joe updates existing policies or adds new ones to accommodate the partner Widget Company. For example, Joe can create a new policy that denies members of the WidgetCo group the ability to use any actions other than write. This policy would be necessary only if there's a broad policy that gives all users access to a wide set of Amazon S3 actions.

IAM and the AWS Management Console

The AWS Management Console provides a web-based way to administer AWS services. You can sign in to the console and create, list, and perform other tasks with AWS services for your account, such as starting and stopping Amazon EC2 instances and Amazon RDS databases, creating Amazon DynamoDB tables, creating IAM users, and so on.

If you're the account owner, you can sign in to the console directly. If you've created IAM users in your account, assigned passwords to those users, and given the users permissions, they can sign in to the console using a URL that's specific to your account.

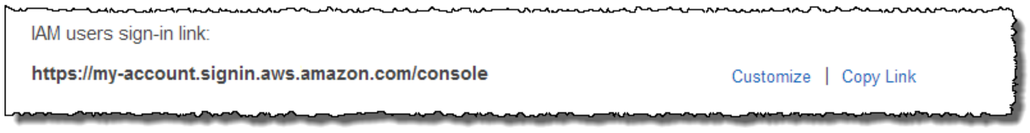
This section provides information about the IAM-enabled AWS Management Console sign-in page and explains how to create a unique sign-in URL for your account. For information about creating user passwords, see Credentials (Passwords, Access Keys, and MFA devices) (p. 56).

Note
If your organization has an existing identity system, you might want to create a single sign-on (SSO) option that gives users access to the AWS Management Console for your account without requiring them to have an IAM user identity and without requiring them to sign in separately to your organization's site and to AWS. For more information, see Giving Federated Users Direct Access to the AWS Management Console in the *Using Temporary Security Credentials* guide.

Topics

The AWS Management Console Sign-in Page

Users who want to use the AWS Management Console must sign in to your AWS account through the unique sign-in page that's specific to your account. You provide your users with the URL they need to access the sign-in page. You can find the URL for your account on the dashboard of the IAM console.


IAM users sign-in link:

https://my-account.signin.aws.amazon.com/console

Important
In addition to providing users with a URL to your unique sign-in page, before users can sign in to your page, you must provide each user with a password and, if appropriate, an MFA device. For detailed information about passwords and MFA devices, see Credentials (Passwords, Access Keys, and MFA devices) (p. 56) and Using Multi-Factor Authentication (MFA) Devices with AWS (p. 69).

The unique sign-in page URL is created automatically when you begin using IAM. It has the following format.

```
https://My_AWS_Account_ID.signin.www.amazonaws.cn/console/
```

Note
To locate your AWS account ID, go to the AWS AWS Security Credentials page. Your account ID is in the **Account Identifiers** section.

You can customize the unique sign-in URL for your account if you want the URL to contain your company name (or other friendly identifier) instead of your AWS account ID. For more information about customizing the unique sign-in URL, see Your AWS Account ID and Its Alias (p. 36).

Tip
To create a bookmark for your account's unique sign-in page in your web browser, you should manually enter your account's sign-in URL in the bookmark entry. Don't use your web browser's "bookmark this page" feature.

Using AWS Account Credentials to Sign In to the AWS Management Console

When users sign in to your AWS account, they sign in via a unique sign-in page. For their convenience, this sign-in page uses a cookie to remember user status so that the next time a user goes to the AWS Management Console, the AWS Management Console uses the unique sign-in page by default.

If you want to sign in to the AWS Management Console using your AWS account credentials instead of IAM user credentials, go to the unique sign-in page and then click **Sign in using root account credentials**. The Amazon Web Services sign-in page appears, from which you can sign in using your AWS account credentials.

Controlling User Access to the AWS Management Console

Users who sign in to your AWS account through the sign-in page can access your AWS resources through the AWS Management Console to the extent that you grant them permission. The following list shows the ways you can grant users access to your AWS account resources through the AWS Management Console. It also shows how users can access other AWS account features through the AWS website.

Note
There is no charge to use IAM.

The AWS Management Console
: You create a password for each user who needs access to the AWS Management Console. Users access the console via your IAM-enabled AWS account sign-in page. For information about accessing the sign-in page, see The AWS Management Console Sign-in Page (p. 34). For information about creating passwords, see Credentials (Passwords, Access Keys, and MFA devices) (p. 56).

Your AWS resources, such as Amazon EC2 instances, Amazon S3 buckets, and so on
: Even if your users have passwords, they still need permission to access your AWS resources. When you create a user, that user has no permissions by default. To give your users the permissions they need, you attach policies to them. If you have many users who will be performing the same tasks with the same resources, you can assign those users to a group, then assign the permissions to that group. For information about creating users and groups, see IAM Users and Groups (p. 39). For information about using policies to set permissions, see Permissions and Policies (p. 171).

AWS Discussion Forums
: Anyone can read the posts on the AWS Discussion Forums. Users who want to post questions or comments to the AWS Discussion Forum can do so using their user name. The first time a user posts to the AWS Discussion Forum, the user is prompted to enter a nickname and email address for use only by that user in the AWS Discussion Forums.

Your AWS account billing and usage information
: You can grant users access your AWS account billing and usage information. For more information, see Controlling Access to Your Billing Information in the *AWS Billing and Cost Management User Guide*.

Your AWS account profile information
: Users cannot access your AWS account profile information.

Your AWS account security credentials
: Users cannot access your AWS account security credentials.

Note
IAM policies control access regardless of the interface. For example, you could provide a user with a password to access the AWS Management Console, and the policies for that user (or any groups the user belongs to) would control what the user can do in the AWS Management Console. Or, you could provide the user with AWS access keys for making API calls to AWS, and the policies would control which actions the user could call through a library or client that uses those access keys for authentication.

Your AWS Account ID and Its Alias

Finding Your AWS Account ID

To find your AWS account ID number in the AWS Management Console, click on **Support** in the navigation bar in the upper-right, and then click **Support Center**. Your currently signed in account ID appears below the **Support** menu.

About Account Aliases

If you want the URL for your sign-in page to contain your company name (or other friendly identifier) instead of your AWS account ID, you can create an alias for your AWS account ID. This section provides information about AWS account aliases and lists the API actions you use to create an alias.

Your sign-in page URL has the following format, by default.

```
https://your_AWS_Account_ID.signin.www.amazonaws.cn/console/
```

If you create an AWS account alias for your AWS account ID, your sign-in page URL will look like the following example.

```
https://youralias.signin.www.amazonaws.cn/console/
```

Note
The original URL containing your AWS account ID remains active after you create your AWS account alias.

Tip
To create a bookmark for your account's unique sign-in page in your web browser, you should manually enter your account's sign-in URL in the bookmark entry. Don't use your web browser's "bookmark this page" feature.

Creating, Deleting, and Listing an AWS Account Alias

You can use the AWS Management Console, the IAM API, or the command line interface to create or delete your AWS account alias.

Important
Your AWS account cannot have more than one alias. If you create a new alias for your AWS account, the new alias overwrites the old alias, and the URL containing the old alias will no longer work.

AWS Management Console

To create or remove an account alias

1. Sign in to the AWS Management Console and open the IAM console at https://console.amazonaws.cn/iam/.
2. On the navigation pane, select **Dashboard**.
3. Find the IAM users sign-in link.
4. To create the alias, click **Customize**, enter the name you want to use for your alias, then click **Yes, Create**.
5. To remove the alias, click **Customize**, and then click **Yes, Delete**. The sign-in URL reverts to using your AWS account ID.

API or CLI

The following table lists the API actions or command line interface (CLI) commands to use to create, delete, or list an AWS account ID sign-in page alias.

Task	Command to Use
Create an alias for your AWS Management Console sign-in page URL	CLI: aws iam create-account-alias API: `CreateAccountAlias`
Delete an AWS account ID alias	CLI: aws iam delete-account-alias API: `DeleteAccountAlias`

Task	Command to Use
List your AWS account ID alias	CLI: aws iam list-account-aliases API: `ListAccountAliases`

Note
The alias must be unique across all Amazon Web Services products. It must contain only digits, lowercase letters, and hyphens. For more information on limitations on AWS account entities, see Limitations on IAM Entities (p. 13).

MFA Devices and Your IAM-Enabled Sign-in Page

If a user must use an MFA device to sign in to your IAM-enabled sign-in page, the user is prompted to enter the MFA device authentication code after entering a user name and password. In most cases, the user will be able to use the AWS Management Console after entering the required information.

However, it's possible for an MFA device to get out of synchronization. If after several unsuccessful tries a user cannot sign in to the AWS Management Console, the user will be prompted to synchronize the MFA device. The user can follow the on-screen prompts to synchronize the MFA device. For information about how you can synchronize a device for a user under your AWS account, see Synchronizing an MFA Device (p. 77).

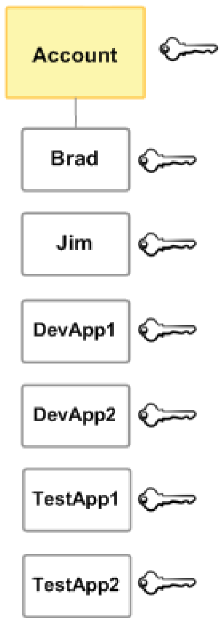
IAM Users and Groups

This section describes *IAM users*, which you create in order to provide AWS identities (authentication) for people and processes in your AWS account. This section also describes *groups*, which are collections of IAM users that you can manage as a unit.

IAM Users

An IAM *user* is an entity that you create in AWS that provides a way to interact with AWS. A primary use for IAM users is to give people you work with identities that they can use to sign in to the AWS Management Console and to make requests to AWS services.

Newly created IAM users have no password and no access key (access key ID and secret access key). If the user needs to administer your AWS resources using the AWS Management Console, you can create a password for the user. If the user needs to interact with AWS programmatically (using the command line interface (CLI), the AWS SDK, or service-specific APIs), you can create an access key for that user. The credentials you create for users are what they use to uniquely identify themselves to AWS.



You can enhance the security of the user's credentials by enabling multi-factor authentication (MFA) (p. 69) for the user. With MFA, users have to provide both the credentials that are part of their user identity (a password or access key) and a temporary numeric code that's generated on a hardware device or by an application on a smartphone or tablet.

New IAM users also have no permissions (p. 171) to do anything —that is, they are not authorized to perform any AWS actions or to access any AWS resources. An advantage of having individual IAM users is that you can assign permissions individually to each user. You might assign administrative permissions to a few users, who then can administer your AWS resources and can even administer other IAM users. In most cases, you want to limit a user's permissions to just the tasks (AWS actions) and resources that the user needs for his or her job. Imagine an IAM user named `Dave`. When you create the user `Dave`, you create a password for that user and you attach permissions to the user that let him start Amazon EC2 instances and read (`GET`) information from an Amazon RDS database.

Each user is associated with one and only one AWS account. Because users are defined within your AWS account, users don't need to have a payment method on file with AWS. Any AWS activity performed by users in your account is billed to your account.

There's a limit to the number of users you can have. For more information, see Limitations on IAM Entities (p. 13).

IAM Users Aren't Necessarily People

An IAM user doesn't necessarily have to represent an actual person. An IAM user is really just an identity with associated permission. You might create an IAM user to represent an application that needs to have credentials in order to make requests to AWS. An application might have its own identity in your account, and its own set of permissions, the same way that processes have their own identities and permissions in an operating system like Windows or Linux.

Scenarios for Creating IAM Users

Because an IAM user is just an identity with specific permissions in your account, you might not need to create IAM users for every occasion on which you need credentials. In many cases, you can take advantage of the AWS Security Token Service to create temporary security credentials and IAM *roles* instead of using the long-term credentials associated with an IAM user.

Consider the following scenarios for when you might (and might not) need to create an IAM user in order to have credentials for accessing AWS.

Do you need to create an IAM user?

- **You created an AWS account and you're the only person who works in your account.**

  Yes. It's possible to work with AWS using the credentials for your root account. However, as a security best practice, we strongly recommend that you create an IAM user for yourself and use the credentials for that user when you work with AWS.
- **Other people in your group need to work in your AWS account, and your group is using no other identity mechanism.**

  Yes. Create IAM users for the individual people who need access to your AWS resources, assign appropriate permissions to each user, and give each user his or her own credentials.
- **You want to use the command-line interface (CLI) to work with AWS.**

  Yes. The CLI needs credentials that it can use to make calls to AWS. Create an IAM user and give that user permissions to run the CLI commands you need. Then configure the CLI on your computer to use these credentials.
- **You're creating an application that runs on an Amazon EC2 instance and that makes requests to AWS.**

  No. Don't create an IAM user and pass the user's credentials to the application or embed the credentials in the application. Instead, use roles for EC2 to give the application temporary security credentials. For details, see Using IAM Roles to Delegate Permissions to Applications that Run on Amazon EC2 (p. 140).
- **You're creating an app that runs on a mobile phone and that makes requests to AWS.**

  No. Don't create an IAM user and distribute the user's access key with the app. Instead, use an identity provider like Login with Amazon, Amazon Cognito, Facebook, or Google to authenticate users, and then use that identity to get temporary security credentials. For more information, see the following:
  - Amazon Cognito Overview in the *AWS SDK for Android Developer Guide*
  - Amazon Cognito Overview in the *AWS SDK for iOS Developer Guide*
  - Creating a Mobile App with Third-Party Sign-In in the *Using Temporary Security Credentials* guide
- **Users in your company are authenticated in your corporate network and want to be able to use AWS without having to sign in again—that is, you want to allow users to federate into AWS.**

  Maybe. It depends on which approach you take to federating user identies into AWS. In that case, instead of creating individual IAM users for each user who needs AWS access, it might be practical to use a proxy server to translate user identities from the network into temporary AWS security credentials. If your company supports SAML 2.0, you can establish trust between your company's identity broker and AWS. For more information, see Using Your Company's Own Authentication System to Grant Access to AWS Resources and Using Your Organization's Authentication System and SAML to Grant Access to AWS Resources in the *Using Temporary Security Credentials* guide.

IAM User Tasks

For more information, see the following topics:

- Creating an IAM User in Your AWS Account (p. 43)
- How IAM Users Sign In to Your AWS Account (p. 46)
- Listing IAM Users (p. 47)

- Changing a User's Name or Path (p. 48)
- Deleting an IAM User from Your AWS Account (p. 48)

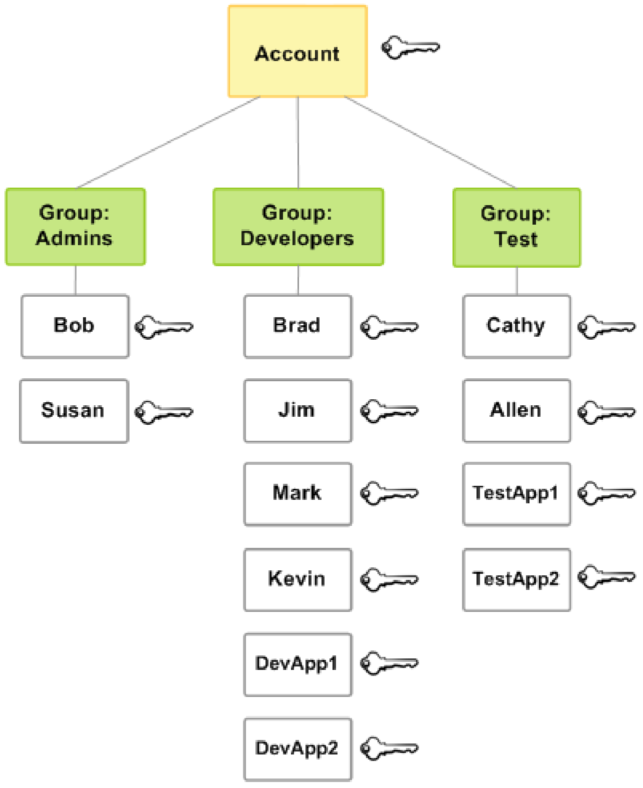
IAM Groups

A *group* is a collection of IAM users. Groups let you specify permissions for a collection of users, which can make it easier to manage the permissions for those users. For example, you could have a group called *Admins* and give that group the types of permissions that administrators typically need. Any user in that group automatically has the permissions that are assigned to the group. If a new user joins your organization and should have administrator privileges, you can assign the appropriate permissions by adding the user to that group. Similarly, if a person changes jobs in your organization, instead of editing that user's permissions, you can remove him or her from the old group and add him or her to the new group.

Following are some important characteristics of groups:

- A group can contain many users, and a user can belong to multiple groups.
- Groups can't be nested; they can contain only users, not other groups.
- There's no default group that automatically includes all users in the AWS account. If you want to have a group like that, you need to create it and assign each new user to it.
- There's a limit to the number of groups you can have, and a limit to how many groups a user can be in. For more information, see Limitations on IAM Entities (p. 13).

The following diagram shows a simple example of a small company. The company owner uses the AWS account credentials to create an `Admins` group that can includes users who can create and manage the users as the company grows. The `Admins` group establishes a `Development` group and a `Test` group. Each of these groups consists of users (humans and applications) that interact with AWS (Jim, Brad, DevApp1, and so on). Each user has an individual set of security credentials. In this example, each user belongs to a single group. However, users can belong to multiple groups.



IAM Group Tasks

For more information, see the following topics:

- Creating IAM Groups (p. 50)
- Adding and Removing Users in an IAM Group (p. 51)
- Listing IAM Groups (p. 52)
- Changing a Group's Name or Path (p. 53)
- Deleting an IAM Group (p. 54)

Creating an IAM User in Your AWS Account

This section describes the process for creating an IAM user in your AWS account. You might create an IAM user when someone joins your organization, or when you have a new application that needs to make API calls to AWS.

Topics

- Overview (p. 44)
- AWS-Assigned User Identifiers (p. 44)
- Creating an IAM User (AWS Management Console) (p. 44)
- Creating an IAM User (CLI or API) (p. 45)

Overview

In outline, the process of creating a user consists of these steps:

1. Create the user using the console or a CLI or API command.
2. (Optional) Add the user to one or more groups.
3. If the user will administer AWS resources using the AWS Management Console, create a password for the user and attach a policy to the user or the group that grants permissions to perform the actions you want to allow.
4. If the user will be making API calls or using the command line interface (CLI), create an access key (an access key ID and a secret access key) for that user.

   Important
   This is your only opportunity to view or download the keys, and you must provide this information to your users before they can begin using an AWS API. If you don't download and save them now, you will need to create new access keys for the users later. Save the user's new access key ID and secret access key in a safe and secure place. **You will not have access to the secret access keys again after this step.**
5. (Optional) Configure a multi-factor authentication (MFA) device for the user, which requires the user to provide a temporary code each time he or she signs into the AWS Management Console.
6. Provide the user with the information needed to sign-in. This includes the credentials and the URL for the web page where the user enters those credentials. For more information, see How IAM Users Sign In to Your AWS Account (p. 46)

   You can give users permissions to manage their own security credentials. (By default, users do not have permissions to manage their own credentials.) For more information, see Permissions for Administering IAM Users, Groups, and Credentials (p. 95).

For information about the permissions that you need in order to create a user, see Permissions for Administering IAM Users, Groups, and Credentials (p. 95).

AWS-Assigned User Identifiers

When you create a user, IAM creates these ways to identify the user:

- A "friendly name" for the user, which is the name that you specified when you created the user, such as `Bob` and `Alice`. These are the names you see in the AWS Management Console.
- An Amazon Resource Name (ARN) for the user. You use the ARN when you need to uniquely identify the user across all of AWS, such as when you specify the user as a principal in an IAM policy for an Amazon S3 bucket. An ARN for an IAM user might look like the following:

  ```
  arn:aws:iam::account-ID-without-hyphens:user/Bob
  ```
- A unique identifier for the user. (This ID is returned only when you use the API or CLI to create the user, not in the console.)

For more information about these identifiers, see IAM Identifiers (p. 8).

Creating an IAM User (AWS Management Console)

To create a user using the AWS Management Console

1. Sign in to the AWS Management Console and open the IAM console at https://console.amazonaws.cn/iam/.

2. In the navigation pane, click **Users** and then click **Create New Users**.
3. Enter the user names for the users you want to create. You can create up to five users at one time.

   Note
   User names can use only alphanumeric characters plus these characters: plus (+), equal (=), comma (,), period (.), at (@), and hyphen (-). For more information about limitations on IAM entities, see Limitations on IAM Entities (p. 13).

4. If you want to generate an access key ID and secret access key for new users, select **Generate an access key for each user**. Users must have keys if they need to work with the AWS CLI or with the AWS SDKs or APIs. Click **Create**.

   Note
   If you have users who will work with the AWS Management Console, you must create passwords for each of them. Creating passwords is described later in this procedure.

5. A page appears that enables you to download the access key IDs for the new user or users. To save the access keys for the new user or users, click **Download Credentials**. This lets you save the access key IDs and secret access keys to a CSV file on your computer.

   Important
   This is your only opportunity to view or download the keys, and you must provide this information to your users before they can begin using an AWS API. If you don't download and save them now, you will need to create new access keys for the users later. Save the user's new access key ID and secret access key in a safe and secure place. **You will not have access to the secret access keys again after this step.**

6. (Optional) Give the user permission to manage his or her own security credentials. For more information, see Allow Users to Manage Their Own Passwords and Access Keys (p. 98).

Creating an IAM User (CLI or API)

1. Create a user.

   CLI command: `aws iam create-user`

   API command: `CreateUser`
2. (Optional) Give the user a password, which is required if the user needs to use the AWS Management Console. Then give them the URL of your account's sign-in page. (p. 46)

   CLI command: `aws iam create-login-profile`

   API command: `CreateLoginProfile`
3. (Optional) Create an access key for the user, which is required if the user needs to programmatically access AWS resources.

   CLI command: `aws iam create-access-key`

   API command: `CreateAccessKey`

   Important
   This is your only opportunity to view or download the keys, and you must provide this information to your users before they can begin using an AWS API. If you don't download and save them now, you will need to create new access keys for the users later. Save the user's new access key ID and secret access key in a safe and secure place. **You will not have access to the secret access keys again after this step.**
4. (Optional) Attach a policy to the user that defines the user's permissions. Note that a best practice is to instead manage user permissions by adding the user to a group and attaching a policy to the group. (See next step.)

CLI command: `aws iam attach-user-policy`

API command: `AttachUserPolicy`

5. (Optional) Add the user to one or more groups.

   CLI command: `aws iam add-user-to-group`

   API command: `AddUserToGroup`

6. (Optional) Give the user permission to manage his or her own security credentials. For more information, see Allow Users to Manage Their Own Passwords and Access Keys (p. 98).

For an example of how to use AWS CLI commands to perform these tasks, see AWS Identity and Access Management from the AWS Command Line Interface in the *AWS Command Line Interface User Guide*.

How IAM Users Sign In to Your AWS Account

After you have created IAM users and created passwords for them, users can sign in to the AWS Management Console for your AWS account by using a special URL, which has this format:

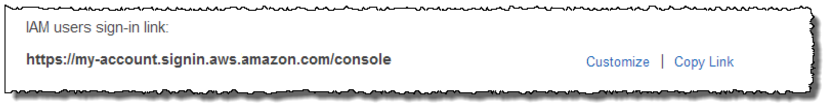
```
https://AWS-account-ID.signin.www.amazonaws.cn/console
```

For example, the URL might look like this:

```
https://123456789012.signin.www.amazonaws.cn/console
```

By default, the sign-in URL for your account includes your account ID. You can create a unique sign-in URL for your account so that the URL includes a name instead of an account ID. For more information, see Your AWS Account ID and Its Alias (p. 36).

You can also find the sign-in URL for an account on the IAM console dashboard.


IAM users sign-in link:

https://my-account.signin.aws.amazon.com/console

Tip
To create a bookmark for your account's unique sign-in page in your web browser, you should manually enter your account's sign-in URL in the bookmark entry. Don't use your web browser's "bookmark this page" feature.

IAM users in your account have access only to the AWS resources that you have granted them permissions to, using a policy that is attached to the user or to an IAM group that the user belongs to. To work in the console, users must have permissions to perform the actions that the console performs, such as listing and creating AWS resources. For more information, see Permissions and Policies (p. 171) and Example Policies for Administering AWS Resources (p. 203).

Note
If your organization has an existing identity system, you might want to create a single sign-on (SSO) option that gives users access to the AWS Management Console for your account without requiring them to have an IAM user identity and without requiring them to sign in separately to your organization's site and to AWS. For more information, see Giving Federated Users Direct Access to the AWS Management Console in the *Using Temporary Security Credentials* guide.

If users need programmatic access to work with your account, you can create an access key (an access key ID and a secret access key) for each user, as described in Creating, Modifying, and Viewing User Access Keys (AWS Management Console) (p. 68).

Listing IAM Users

This section describes how to list the users in your AWS account or in a specific group, and how to list all the groups that a user is in.

For information about the permissions that you need in order to list users, see Permissions for Administering IAM Users, Groups, and Credentials (p. 95).

Topics

Listing Users (AWS Management Console)

To list all users in your AWS account

1. Sign in to the AWS Management Console and open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, click **Users**. The console displays the users in your AWS account.

   Note
   The AWS Management Console can display up to 1000 users. If you have more than 1000 users in your AWS account, use the AWS CLI command `aws iam list-users` to list all the users in your account.
3. To see what groups a user is in, click the user name and then open the **Groups** section.

To list all users in a group

1. Sign in to the AWS Management Console and open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, click **Groups**, click the name of the group, and then open the **Users** section.

Listing Users (CLI and API)

To list IAM users, use the following commands:

- List all the users in the account.

  CLI command: `aws iam list-users`

  API command: `ListUsers`
- List the users in a specific group.

  CLI command: `aws iam get-group`

  API command: `GetGroup`
- List all the groups that a user is in.

CLI command: `aws iam list-groups-for-user`

API command: `ListGroupsForUser`

Changing a User's Name or Path

To change a user's name or path, you must use the IAM CLI or API. There is no option in the console to rename a user.

For information about the permissions that you need in order to rename a user, see Permissions for Administering IAM Users, Groups, and Credentials (p. 95).

To rename IAM users, use the following commands:

- CLI: `aws iam update-user`
- API: `UpdateUser`

When you change a user's name or path, the following happens:

- Any policies attached to the user stay with the user under the new name.
- The user stays in the same groups under the new name.
- The unique ID for the user remains the same. For more information about unique IDs, see Unique IDs (p. 12).
- Any resource or role policies that refer to the user *as the principal* (the user is being granted access) are automatically updated to use the new name or path. For example, any queue-based policies in Amazon SQS or resource-based policies in Amazon S3 are automatically updated to use the new name and path.

IAM does not automatically update policies that refer to the user *as a resource* to use the new name or path; you must manually do that. For example, imagine that user `Bob` has a policy attached to him that lets him manage his security credentials. If an administrator renames `Bob` to `Robert`, the administrator also needs to update that policy to change the resource from this:

```
arn:aws:iam::account-ID-without-hyphens:user/division_abc/subdivision_xyz/Bob
```

to this:

```
arn:aws:iam::account-ID-without-hyphens:user/division_abc/subdivision_xyz/Robert
```

This is true also if an administrator changes the path; the administrator needs to update the policy to reflect the new path for the user.

Deleting an IAM User from Your AWS Account

You might delete an IAM user from your account if someone quits your company. If the user is only temporarily away from your company, you can disable the user's credentials instead of deleting the user entirely from the AWS account. That way, you can prevent the user from accessing the AWS account's resources during the absence but you can re-enable the user later. For more information about disabling credentials, see Managing Access Keys for IAM Users (p. 67).

For information about the permissions that you need in order to delete a user, see Permissions for Administering IAM Users, Groups, and Credentials (p. 95).

Topics

Deleting an IAM User (AWS Management Console)

When you use the AWS Management Console to delete an IAM user, IAM deletes the following information:

- The user
- Any group memberships—that is, the user is removed from any IAM groups that the user was a member of
- Any password associated with the user
- Any access keys belonging to the user
- All inline policies embedded in the user (policies that are applied to a user via group permissions are not affected)

  Note
  Any managed policies attached to the user are detached from the user when the user is deleted. Managed policies are not deleted when you delete a user.
- Any associated MFA device

To use the AWS Management Console to delete a user

1. Sign in to the AWS Management Console and open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, click **Users**, and then select the check box next to the user name.
3. From the **User Actions** list at the top of the page, select **Delete User**.
4. Click **Yes, Delete**.

Deleting an IAM User (AWS CLI)

To delete a user from your account using the AWS Command Line Interface (AWS CLI) CLI, follow these steps:

1. Delete the user's keys and certificates. This helps ensure that the user can't access your AWS account's resources anymore. Note that when you delete a security credential, it's gone forever and can't be retrieved.

   `aws iam delete-access-key` and `aws iam delete-signing-certificate`
2. Delete the user's password, if the user has one.

   `aws iam delete-login-profile`
3. Deactivate the user's MFA device, if the user has one.

   `aws iam deactivate-mfa-device`
4. Detach any policies that are attached to the user.

   `aws iam list-attached-user-policies` (to list the policies attached to the user) and `aws iam detach-user-policy` (to detach the policies)

5. Get a list of any groups the user was in, and remove the user from those groups.

   `aws iam list-groups-for-user` and `aws iam remove-user-from-group`
6. Delete the user.

   `aws iam delete-user`

Creating IAM Groups

To set up a group, you need to create the group and then give it permissions based on the type of work you expect the users in the group to do.

For information about the permissions that you need in order to create a group, see Permissions for Administering IAM Users, Groups, and Credentials (p. 95).

Topics

Creating an IAM Group (AWS Management Console)

For an example of how to set up a group, go to Creating an Administrators Group Using the Console (p. 20).

Attaching a Policy to an IAM Group (AWS Management Console)

The following procedure describes how to attach a managed policy to a group. To attach an AWS managed policy—that is, a pre-written policy provided by AWS—to a group, follow the steps in the following procedure now. To attach a customer managed policy—that is, a policy with custom permissions that you create—you must first create the policy. For information about creating customer managed policies, see Creating Customer Managed Policies (p. 194).

For more information about permissions and policies, see Permissions and Policies (p. 171).

To attach a policy to a group

1. Sign in to the AWS Management Console and open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, select **Policies**.
3. In the list of policies, select the check box next to the name of the policy to attach. You can use the **Filter** menu and the **Search** box to filter the list of policies.
4. Click **Policy Actions**, then click **Attach**.
5. Click **All Types** in the **Filter** menu, then click **Groups**.
6. Select the check box next to the name of the group to attach the policy to, then click **Attach Policy**.

Creating an IAM Group (AWS CLI or API)

To create IAM groups and attach policies to them, use the following commands:

- Create a group

  CLI: `aws iam create-group`

  API: `CreateGroup`
- Attach a policy to the group

  CLI: `aws iam attach-group-policy`

  API: `AttachGroupPolicy`

Adding and Removing Users in an IAM Group

You can add IAM users to a group to make it easier to manage users. Any policies that are attached to the group automatically apply to the users in the group. (If the group has a policy attached and the user also has a policy attached, IAM evaluates both policies when a request is made using the user's credentials.) If a user in your organization changes responsibilities, you can move the IAM user to a different group that has different permissions.

For information about the permissions that you need in order to add and remove users in a group, see Permissions for Administering IAM Users, Groups, and Credentials (p. 95).

Topics

Adding and Removing Users in a Group (AWS Management Console)

To add users to a group or remove users from a group

1. Sign in to the AWS Management Console and open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, click **Groups**, and then click the name of the group.
3. Open the **Users** section.
4. To add users to the group, click **Add Users to Group**. To remove users from the group, click **Remove Users from Group**.
5. Select the users you want to add to the group or remove from the group, and then click **Add Users** or **Remove Users**.

Adding and Removing Users in a Group (CLI and API)

To add or delete an IAM user in a group, use the following commands:

- Add a user to a group.

  CLI: `aws iam add-user-to-group`

  API: `AddUserToGroup`
- Remove a user from a group.

  CLI: `aws iam remove-user-from-group`

  API: `RemoveUserFromGroup`

Listing IAM Groups

You can list all of the IAM groups in your account or all of the groups that an IAM user belongs to. If you are using the IAM API or CLI, you can get a list of the groups with a certain path prefix.

For information about the permissions that you need to list groups, see Permissions for Administering IAM Users, Groups, and Credentials (p. 95).

Topics

Listing IAM Groups (AWS Management Console)

To list all groups in your AWS account

1. Sign in to the AWS Management Console and open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, click **Groups**.

To list all of the groups a user belongs to

1. Sign in to the AWS Management Console and open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, click **Users**, and then click the user name.
3. Open the **Groups** section.

Listing IAM Groups (CLI or API)

To list IAM groups, use the following commands:

- List all the groups in the AWS account or list all the groups with a particular path prefix.

  CLI: `aws iam list-groups`

  API: `ListGroups`
- List the IAM users in a group.

  CLI: `aws iam get-group`

  API: `ListGroupsForUser`

Changing a Group's Name or Path

You can use the AWS Management Console to change a group's name, or you can use the IAM CLI or API.

For information about the permissions that you need in order to rename a group, see Permissions for Administering IAM Users, Groups, and Credentials (p. 95).

Topics

Overview

When you change a group's name or path, the following happens:

- Any policies attached to the group stay with the group under the new name.
- The group retains all its users under the new name.
- The unique ID for the group remains the same. For more information about unique IDs, see Unique IDs (p. 12).

We do not automatically update policies that refer to the group as a resource to use the new name; you must manually do that. For example, let's say Bob is the manager of the testing part of the organization, and he has a policy attached to his IAM user entity that lets him use `UpdateGroup` with the Test group to add and remove users. Let's say that an admin changes the name of the group to `Test_1` (or changes the path for the group). The admin also needs to update the policy attached to Bob to use the new name (or new path) so that Bob can continue to add and remove users from the group.

Renaming an IAM Group (AWS Management Console)

To change a group's name

1. Sign in to the AWS Management Console and open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, click **Groups** and then select the check box next to the group name.
3. From the **Group Actions** list at the top of the page, select **Edit Group Name**.
4. Enter the new group name, and then click **Yes, Edit**.

Renaming an IAM Group (CLI or API)

To rename a group using the CLI or API, use the following commands:

- CLI: `aws iam update-group`
- API: `UpdateGroup`

Deleting an IAM Group

You can delete a group that you no longer need.

For information about the permissions that you need in order to delete a group, see Permissions for Administering IAM Users, Groups, and Credentials (p. 95).

Topics

Deleting an IAM Group (AWS Management Console)

When you use the AWS Management Console to delete a group, IAM deletes the group and any inline policies embedded in the group, but leaves the users intact, and leaves intact any managed policies that were attached to the group. Permissions the users had because they belonged to the group no longer apply to them.

To use the AWS Management Console to delete a group

1. Sign in to the AWS Management Console and open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, click **Groups**, and then select the check box next to the group name.
3. From the **Group Actions** list at the top of the page, select **Delete Group**, and then click **Yes, Delete**.

Deleting an IAM Group (AWS CLI and API)

When you use the AWS Command Line Interface (AWS CLI) or IAM API to delete a group, you must remove the users in the group, delete any inline policies embedded in the group, and detach any managed policies attached to the group before you can delete the group. Follow these steps:

1. Remove all users from the group.

   CLI: `aws iam get-group` (to get the list of users in the group), and `aws iam remove-user-from-group` (to remove a user from the group)

   API: `GetGroup` (to get the list of users in the group), and `RemoveUserFromGroup` (to remove a user from the group)
2. Delete all inline policies embedded in the group.

   CLI: `aws iam list-group-policies` (to get a list of the group's inline policies), and `aws iam delete-group-policy` (to delete the group's inline policies)

   API: `ListGroupPolicies` (to get a list of the group's inline policies), and `DeleteGroupPolicy` (to delete the group's inline policies)
3. Detach all managed policies attached to the group.

   CLI: `aws iam list-attached-group-policies` (to get a list of the managed policies attached to the group), and `aws iam detach-group-policy` (to detach a managed policy from the group)

   API: `ListAttachedGroupPolicies` (to get a list of the managed policies attached to the group'), and `DetachGroupPolicy` (to detach a managed policy from the group)

4. Delete the group.

   CLI: `aws iam delete-group`

   API: `DeleteGroup`

Credentials (Passwords, Access Keys, and MFA devices)

This section describes how to administer your users' credentials in IAM—that is, their passwords, access keys, and multi-factor authentication (MFA) devices.

To access your AWS account resources, users must have credentials. To use the AWS Management Console, users must have a password. To use the AWS Command Line Interface (AWS CLI) or to make API calls, users must have an access key (an access key ID and secret access key). Users who access your resources only through the API or AWS CLI do not need a password.

For extra security, you can enable multi-factor authentication (MFA) for users. MFA adds security by requiring users to enter an authentication code from a hardware device or virtual device in addition to providing a password or access key.

The following list describes the options for how to administer passwords, access keys, and MFA devices.

Change your AWS account password.
: You can change the password for the AWS account owner. You must be signed in as the account owner.

  For more information, see Changing Your AWS Account Password (p. 58).

Set a password policy for IAM users in the account.
: You can require that IAM users create passwords that contain certain characters, are of a minimum length, are changed after a specified period of time, and so on.

  For more information, see Setting an Account Password Policy for IAM Users (p. 59).

Administer passwords, access keys, and MFA devices for IAM users, either as account owner or as a privileged IAM user.
: An account owner or an IAM user who has the appropriate permissions can create, change, and delete passwords, access keys, and MFA devices for other IAM users.

  For information about the permissions that an IAM user must have to perform these actions, see Permissions for Administering IAM Users, Groups, and Credentials (p. 95).

Let all IAM users change their own passwords.
: You can set a password policy that, among other things, lets all IAM users change their own passwords using the AWS Management Console. After you enable this option, users can sign in to the AWS Management Console and go to the My Password page to change their password.

Note the following about using this option in your password policy:

- You can allow users to change their own passwords, but not administer their own access keys.
- You can allow *all* users to change their own passwords, or *no* users. You cannot allow only a subset of users to change their passwords.

For more information, see Setting an Account Password Policy for IAM Users (p. 59).

Let selected IAM users administer their own passwords, access keys, and certificates.

Even if you don't want to allow all IAM users to administer their own passwords, you can let selected users administer their own passwords and access keys. This option differs from the previous one in these ways:

- It lets you specify a subset of users who can administer their own passwords.
- You can let users administer both their password and their access keys. (The account-wide setting lets users administer only their passwords.)

For this option, you do the following:

- Create an IAM group and add users to it who have this privilege.
- Set permissions on the group that let users manage their passwords, their access keys, their MFA devices, or all of these.

For more information about setting these permissions, see Permissions for Administering IAM Users, Groups, and Credentials (p. 95).

Administer MFA for the AWS account.

When multi-factor authentication (MFA) is enabled, you must retrieve an authentication code from a hardware device or virtual device before you can sign in to your account using the AWS Management Console.

For more information, see Configuring and Managing a Virtual MFA Device for Your AWS Account (AWS Management Console) (p. 73).

For general information about multi-factor authentication (MFA), see Using Multi-Factor Authentication (MFA) Devices with AWS (p. 69).

Let users administer their own MFA device.

If users work with hardware-based MFA devices, an administrator typically manages the devices and enables MFA for individual IAM users. If users work with virtual devices (such as an app on their own smartphone), they typically manage the device themselves. IAM users who manage MFA for themselves or for others must have permissions to do so.

For more information, see Configuring and Enabling a Virtual MFA Device for a User (p. 71).

For general information about multi-factor authentication (MFA), see Using Multi-Factor Authentication (MFA) Devices with AWS (p. 69).

Learn best practices for managing access keys.

Anyone who has access keys for your account or for IAM users in your account has access to your AWS resources. Here is a set of best practices to help you protect your access keys.

Download a credential report for your account.

You can generate and download a credential report that lists all IAM users in your account and the status of their various credentials, including passwords, access keys, MFA devices, and signing certificates. For passwords, the credential report shows how recently a password has been used.

For more information about IAM credential reports, see Getting Credential Reports for Your AWS Account (p. 93).

Managing Passwords

This section describes how to manage passwords for your AWS account and for IAM users in your account. IAM users need passwords in order to access the AWS Management Console. (They do not need passwords if they will access AWS resources programmatically by using the CLI, AWS SDKs, or the APIs.)

Topics

Changing Your AWS Account Password

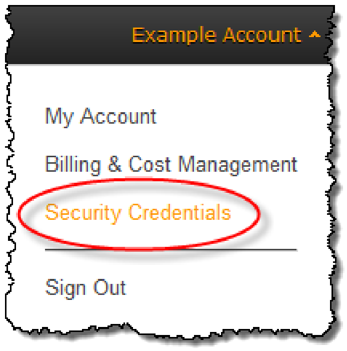
This procedure describes how you can change the password that you use to sign into an AWS account as the account owner (the "root user"). You must be signed in as the account owner in order to change your account password.

To change the password for your AWS account

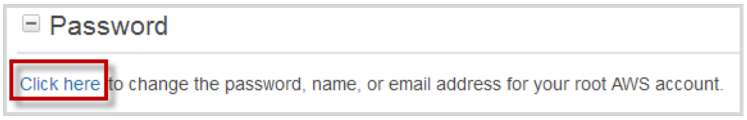
1. Use your AWS account email address and password to sign in to the AWS Management Console.

   Note
   If you previously signed in to the console with IAM user credentials, your browser might remember this preference and open your account-specific sign-in page. You cannot use the user sign-in page to sign in with your account credentials. If you see the user sign-in page, click **Sign in using AWS Account credentials** near the bottom of the page to return to the account sign-in page.

2. In the upper right corner of the console, click the arrow next to the account name or number and then click **Security Credentials**.

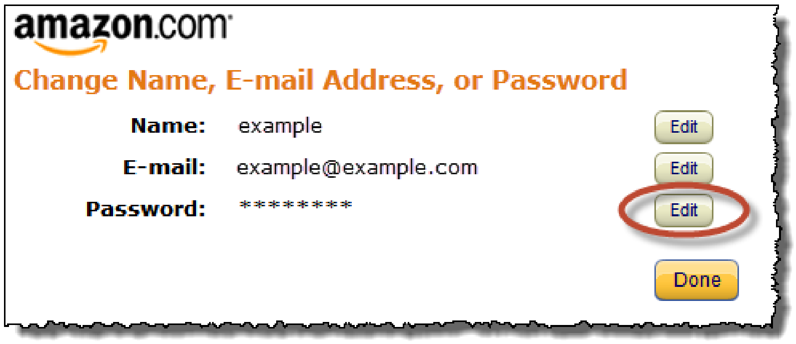


3. On the AWS Security Credentials page, expand the **Password** section and then click **click here**.



Click here to change the password, name, or email address for your root AWS account.

4. Click **Edit** to change your password.


Change Name, E-mail Address, or Password

Name:	example	Edit
E-mail:	example@example.com	Edit
Password:	********	Edit

Setting an Account Password Policy for IAM Users

This topic describes how to set a password policy for your account that lets you specify complexity requirements and rotation periods for passwords for your IAM users. You can use a password policy to do these things:

- Set a minimum password length.
- Require specific character types, including uppercase letters, lowercase letters, numbers, and non-alphanumeric characters. Be sure to remind your users that passwords are case sensitive.
- Allow all IAM users to change their own passwords.
- Require IAM users to change their password after a specified period of time (enable password expiration).
- Prevent IAM users from reusing previous passwords.
- Force IAM users to contact an account administrator when the user has allowed his or her password to expire.

When you create or change a password policy, most of the password policy settings are enforced the next time your users change their passwords. When you set minimum length and character type requirements, they are enforced the next time your users change their passwords—users are not forced to change their existing passwords, even if the pre-existing passwords do not adhere to the updated password policy. When you set a password expiration period, the expiration period is enforced immediately. For example, when you set a password expiration period of 90 days, all IAM users with an existing password that is more than 90 days old are forced to change their password at next sign-in.

Note
For information about the permissions that you need in order to set a password policy, see Permissions for Administering IAM Users, Groups, and Credentials (p. 95).

The IAM password policy does not apply to your AWS root account password.

For enhanced security, use password policies together with multi-factor authentication (MFA). For more information about MFA, see Using Multi-Factor Authentication (MFA) Devices with AWS (p. 69).

Topics

Password Policy Options

The following list describes the options that are available when you configure a password policy for your account.

Minimum password length
: You can specify the minimum number of characters allowed in an IAM user password. You can enter any number from 6 to 128.

Require at least one uppercase letter
: You can require that IAM user passwords contain at least one uppercase character from the ISO basic Latin alphabet (A to Z).

Require at least one lowercase letter
: You can require that IAM user passwords contain at least one lowercase character from the ISO basic Latin alphabet (a to z).

Require at least one number
: You can require that IAM user passwords contain at least one numeric character (0 to 9).

Require at least one non-alphanumeric character
: You can require that IAM user passwords contain at least one of the following non-alphanumeric characters:

```
! @ # $ % ^ & * ( ) _ + - = [ ] { } | '
```

Allow users to change their own password
: Select this option to allow all IAM users in your account to use the IAM console to change their own passwords, as described in Letting IAM Users Change Their Own Passwords (p. 64). (They can change their passwords, but they cannot manage their own access keys or certificates.)

> **Note**
> Alternatively, you can let only some users manage passwords, either for themselves or for others. To do so, you clear the **Allow users to change their own password** option. For more information about setting policies to limit who can manage passwords, see Letting IAM Users Change Their Own Passwords (p. 64).

Enable password expiration
: You can set IAM user passwords to be valid only for the specified number of days. You enter the number of days that passwords will remain valid after they are set. For example, when you enable password expiration and set the password expiration period to 90 days, IAM users can use a password for up to 90 days. After 90 days, the password becomes invalid and the IAM user must set a new password before accessing the AWS Management Console. You can choose a password expiration period between 1 and 1095 days, inclusive.

> **Note**
> The AWS Management Console warns IAM users when they are within 15 days of password expiration. IAM users can change their password at any time (as long as they have been given permission to do so), and when they set a new password the rotation period for that password starts over. An IAM user can have only one valid password at a time.

Prevent password reuse
: You can prevent IAM users from reusing a specified number of previous passwords. You can set the number of previous passwords from 1 to 24, inclusive.

Password expiration requires administrator reset
: You can prevent IAM users from choosing a new password after their current password has expired. For example, if the password policy specifies a password expiration period but an IAM user fails to choose a new password before the expiration period ends, the IAM user cannot set a new password. In that case, the IAM user must request a password reset from an account administrator in order to regain access to the AWS Management Console. If you leave this option cleared and an IAM user allows her or his password to expire, the user will be required to set a new password before accessing the AWS Management Console.

Caution
Before you enable this option, be sure your AWS account has more than one user with administrative permissions (that is, permission to reset IAM user passwords). When this option is enabled and an administrator's password expires, a second administrator is required in order to reset the expired password of the first administrator.

Setting a Password Policy (AWS Management Console)

You can use the AWS Management Console to create, change, or delete a password policy. As part of managing the password policy, you can let all users manage their own passwords.

To create or change a password policy

1. Sign in to the AWS Management Console and open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, click **Account Settings**.
3. In the **Password Policy** section, select the options you want to apply to your password policy.
4. Click **Apply Password Policy**.

To delete a password policy

1. Sign in to the AWS Management Console and open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, click **Account Settings**, and then in the **Password Policy** section, click **Delete Password Policy**.

Setting a Password Policy (CLI or API)

To manage an account password policy using the AWS CLI or APIs, use the following commands:

- Create or change a password policy

  AWS CLI: `aws iam update-account-password-policy`

  API: `UpdateAccountPasswordPolicy`
- Retrieve the password policy

  AWS CLI: `aws iam get-account-password-policy`

  API: `GetAccountPasswordPolicy`
- Delete a password policy

  AWS CLI: `aws iam delete-account-password-policy`

  API: `DeleteAccountPasswordPolicy`

Managing Passwords for IAM Users

IAM users who work with your AWS resources from the AWS Management Console must have a password in order to sign in. This topic explains how to create, change, or delete a password for an IAM user in your AWS account.

After you have assigned a password to a user, the user can sign in to the AWS Management Console using the sign-in URL for your account, which looks like this:

```
https://12-digit-AWS-account-ID.signin.aws.amazon.com/console
```

For more information about how IAM users sign in to the AWS Management Console, see The AWS Management Console Sign-in Page (p. 34).

In addition to manually creating individual passwords for your IAM users, you can create a password policy that applies to all IAM user passwords in your AWS account. You can use a password policy to do these things:

- Set a minimum password length.
- Require specific character types, including uppercase letters, lowercase letters, numbers, and non-alphanumeric characters. Be sure to remind your users that passwords are case sensitive.
- Allow all IAM users to change their own passwords.
- Require IAM users to change their password after a specified period of time (enable password expiration).
- Prevent IAM users from reusing previous passwords.
- Force IAM users to contact an account administrator when the user has allowed his or her password to expire.

For information about managing your account's password policy, see Setting an Account Password Policy for IAM Users (p. 59).

Even if your users have a password, they still need permissions to access your AWS resources. By default, a user has no permissions. To give your users the permissions they need, you assign policies to them or to the groups they belong to. For information about creating users and groups, see IAM Users and Groups (p. 39). For information about using policies to set permissions, see Permissions and Policies (p. 171).

You can grant users permission to change their own passwords. For more information, see Letting IAM Users Change Their Own Passwords (p. 64). For information about how users access your account sign-in page, see The AWS Management Console Sign-in Page (p. 34).

Topics

Creating, Changing, or Deleting an IAM User Password (AWS Management Console)

To set a password for an IAM user

1. Sign in to the AWS Management Console and open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, click **Users**.
3. Click the name of the user who you want to create a password for.
4. Expand the **Security Credentials** section, and then click **Manage Password**.
5. Choose whether to create a custom password or to have IAM generate a password.

   - To create a custom password, select **Assign a custom password**, and enter the password.

Note
The password that you create must meet the account's password policy (p. 59), if you have set a policy.

- To have IAM generate a password, select **Assign an auto-generated password**.

To require the user to choose a new password when they sign in, select **Require user to change password at next sign-in**, and then click **Apply**.

Important
If you select the **Require user to change password at next sign-in** option, make sure the user has permission to change his or her password. For more information, see Letting IAM Users Change Their Own Passwords (p. 64).

6. If you selected the option to auto-generate a password, click **Download Credentials** to save the password as a CSV file to your computer.

   Important
   You will not be able to access the password after completing this step, but you can create a new password at any time.

To change an IAM user's password

1. Sign in to the AWS Management Console and open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, click **Users**.
3. Click the name of the user who you want to change the password for.
4. Expand the **Security Credentials** section, and then click **Manage Password**.
5. Choose whether to replace the existing password with a custom password or to have IAM generate a new password.

   - To create a custom password, select **Replace existing password with new custom password**, and then enter the password.

     Note
     The password that you create must meet the account's password policy (p. 59), if a policy has been set.
   - To have IAM generate a password, select **Replace existing password with new auto-generated password**.

   To require users to choose a new password when they sign in, select **Require user to change password at next sign-in**, and then click **Apply**.

   Important
   If you select the **Require user to change password at next sign-in** option, make sure the user has permission to change his or her password. For more information, see Letting IAM Users Change Their Own Passwords (p. 64).
6. If you selected the option to auto-generate a password, click **Download Credentials** to save the password as a CSV file to your computer.

   Important
   You will not be able to access the password again after completing this step, but you can create a new password at any time.

To delete an IAM user's password

1. Sign in to the AWS Management Console and open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, click **Users**.
3. Click the name of the user who you want to delete the password for.
4. Expand the **Security Credentials** section, and then click **Manage Password**.
5. Click **Remove existing password**, and then click **Apply**.

   Important
   When you remove a user's password, the user cannot sign in to the AWS Management Console.

Creating, Changing, or Deleting an IAM User Password (AWS CLI and API)

To manage passwords for IAM users, use the following commands:

- Create a password

  CLI: aws iam create-login-profile

  API: CreateLoginProfile
- Determine whether a user has a password

  CLI: aws iam get-login-profile

  API: GetLoginProfile
- Determine when a user's password was last used

  CLI: aws iam get-user

  API: GetUser
- Change a user's password

  CLI: aws iam update-login-profile

  API: UpdateLoginProfile
- Delete a user's password

  CLI: aws iam delete-login-profile

  API: DeleteLoginProfile

Note
If you use the API to delete a user from your AWS account, you must delete the password as a separate step in the process of removing the user. For more information about deleting a user, see Deleting an IAM User from Your AWS Account (p. 48).

Letting IAM Users Change Their Own Passwords

This topic describes how to let IAM users change their own passwords for signing into the AWS Management Console. You have these options for allowing user-managed passwords:

- Let all IAM users in the account reset their own passwords.

- Let selected IAM users in the account manage their passwords. In this scenario, you disable the option for all users to reset their passwords and you use an IAM policy to grant permissions to some users to reset their own passwords and optionally other credentials like their own access keys.

> **Important**
> We recommend that you set a password policy (p. 59) so that users create strong passwords.

Topics

Letting IAM Users Change Their Own Passwords

To let all IAM users change their own passwords

1. Sign in to the AWS Management Console and open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, click **Account Settings**.
3. In the **Account Settings** section, select the **Allow users to change their own password** check box, and then click **Apply Password Policy**.
4. Point users to the following instructions that show how they can change their passwords: How IAM Users Change Their Own Password (p. 66).

For information about the CLI and API commands you can use to change the account's password policy (which includes letting all users change their own passwords), see Setting a Password Policy (CLI or API) (p. 61).

Letting Selected IAM Users Change Their Own Passwords

To let selected IAM users change their own passwords

1. Sign in to the AWS Management Console and open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, click **Account Settings**.
3. In the **Account Settings** section, make sure that **Allow users to change their own password** is not selected. If this option is selected, all users can change their own passwords. (See previous procedure.)
4. If you do not already have IAM users for users who should be allowed to change their passwords, create them now. For details, see Creating an IAM User in Your AWS Account (p. 43).
5. Create an IAM group for the users who should be allowed to change their passwords, and then add the users from the previous step to the group. For details, see Creating an Administrators Group Using the Console (p. 20) and Adding and Removing Users in an IAM Group (p. 51).

   This step is optional, but it's a best practice to use groups to manage permissions so that you can add and remove users and change the permissions for the group as a whole.
6. Assign the following policy to the group. (For details, see Attaching a Policy to an IAM Group (AWS Management Console) (p. 50).)

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "iam:ChangePassword",
      "iam:GetAccountPasswordPolicy"
    ],
    "Resource": "*"
  }
}
```

This policy grants access to the ChangePassword action, which lets users use the console, the CLI, or the API to change their passwords. It also grants access to the GetAccountPasswordPolicy action, which lets the user view the current password policy; this permission is required in order to let the user display the password-change page in the console.

7. Point users to the following instructions that show how they can change their passwords: How IAM Users Change Their Own Password (p. 66).

More Information

- Permissions for Administering IAM Users, Groups, and Credentials (p. 95)
- Managing Passwords (p. 58)
- Setting an Account Password Policy for IAM Users (p. 59)
- Managing Policies (p. 192)
- How IAM Users Change Their Own Password (p. 66)

How IAM Users Change Their Own Password

If users have been granted permission to change their own passwords, they can use a special page in the AWS Management Console to do this. They can also use the command line interface or the IAM API.

For information about the permissions that users need in order to change their own passwords, see the following topics:

- Letting IAM Users Change Their Own Passwords (p. 64)
- Permissions for Administering IAM Users, Groups, and Credentials (p. 95)

Topics

How Users Change Their Own Password (AWS Management Console)

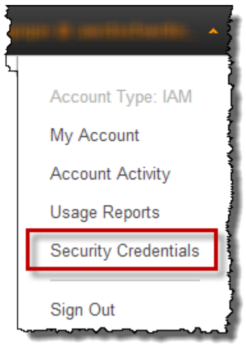
The following procedure shows how an IAM user can change his or her own password using the AWS Management Console

1. Using your IAM user name and password, sign in to the console using the special URL for your account (p. 23), which looks like this:

```
https://your_AWS_Account_ID.signin.aws.amazon.com/console/
```

To get the URL for the sign-in page, contact your administrator.

2. In the navigation bar of the console, click the arrow next to your user name and then click **Security Credentials**.



3. In the **Old Password** box, enter your current password. Enter a new password in the **New Password** and **Confirm New Password** boxes and then click **Change Password**.

   Note
   If the account has a password policy, the new password must meet the requirements of that policy. For more information, see Setting an Account Password Policy for IAM Users (p. 59).

How Users Change Their Own Password (CLI and API)

To change their own password, users can use the `aws iam change-password` CLI command or the `ChangePassword` API action.

Managing Access Keys for IAM Users

This section explains how to create, modify, or view access keys (access key IDs and secret access keys) for IAM users. Users need their own access keys to make programmatic calls to AWS using the AWS Command Line Interface (AWS CLI), the AWS SDKs, or direct HTTP calls using the APIs for individual services.

When you create the access key, IAM returns the access key ID and a secret access key. You should save these in a secure location and give them to the user. To ensure the security of your AWS account, the secret access key is accessible only when you create the access key. If a secret key is lost, you can delete the access key for the associated user and then create a new key.

By default, when you create an access key, its status is `Active`, which means the user can use the access key for API calls. Each user can have two sets of active keys, which is useful when you need to rotate the user's access keys. You can disable a user's access key, which means it can't be used for API calls. You might do this while you're rotating keys (for more information, see Rotating Credentials (p. 92)) or to revoke API access for a user.

You can delete an access key at any time. However, when you delete an access key, it's gone forever and cannot be retrieved. (You can always create new keys.)

You can give your users permission to list and manage their own keys. For more information, see Allow Users to Manage Their Own Passwords and Access Keys (p. 98).

For more information about the credentials used with AWS and IAM, see How Do I Get Credentials? (p. 3) and Creating an IAM User in Your AWS Account (p. 43).

Topics

Creating, Modifying, and Viewing User Access Keys (AWS Management Console)

To list a user's access keys

1. Sign in to the AWS Management Console and open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, click **Users**.
3. Click the name of the user who you want to manage an access key for, and then scroll down to the **Security Credentials** section. The user's access keys and the status of each key is displayed. Users cannot have more than two access keys at a time.

   Note
   Only the user's access key ID is visible. The secret access key can only be retrieved when creating the key.

To create, modify, or delete a user's access keys

1. Sign in to the AWS Management Console and open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, click **Users**.
3. Click the name of the user who you want to create an access key for, and then scroll down to the **Security Credentials** section.
4. Click **Manage Access Keys**.

   a. **To create an access key:**

      Click **Create Access Key**, then click **Download Credentials** to save the access key ID and secret access key to a CSV file on your computer. Store the file in a secure location. You will not have access to the secret access key again after this dialog box closes, and you will need to provide this information to your users before they can begin using an AWS API.

   b. **To disable or re-enable an access key:**

      Click **Make Inactive**. To re-enable the key, click **Make Active**.

   c. **To delete an access key:**

      Click **Delete**.

Creating, Modifying, and Viewing User Access Keys (AWS CLI and API)

To manage a user's access keys, use the following commands:

- To create an access key:

  CLI: `aws iam create-access-key`

  API: `CreateAccessKey`
- To disable or re-enable an access key:

  CLI: `aws iam update-access-key`

  API: `UpdateAccessKey`
- To list a user's access keys:

  CLI: `aws iam list-access-keys`

  API: `ListAccessKeys`
- To delete an access key

  CLI: `aws iam delete-access-key`

  API: `DeleteAccessKey`

Using Multi-Factor Authentication (MFA) Devices with AWS

For increased security, we recommend that you protect your AWS resources by configuring AWS multi-factor authentication (MFA). MFA adds extra security by requiring users to enter a unique authentication code from their authentication device when accessing AWS websites or services.

For MFA to work, you must assign an MFA device (hardware or virtual) to the IAM user or root account. The MFA device must be unique for each user; a user cannot enter a code from another user's device to authenticate. This section shows you how to set up and enable a new MFA device, as well as how to synchronize and deactivate existing devices, and what to do when a device is lost or stops working.

To get started setting up an MFA device for root account or IAM user access to the console, see Setting Up an MFA Device (p. 70).

To set up MFA-protected API access for IAM users with an enabled MFA device, see Configuring MFA-Protected API Access (p. 80).

For answers to commonly asked questions about AWS MFA, go to the AWS Multi-Factor Authentication FAQs.

Topics

Setting Up an MFA Device

The following high-level procedure describes how to set up and use an MFA device, and provides links to related information.

1. *Get an MFA device.* The device can be a hardware MFA device or a virtual MFA device. A virtual device is any device that you can install a TOTP (time-based one-time password) application on. For example, it is common to install MFA applications on a smartphone.

   If you want to use a hardware device, you can find information about where to purchase the devices that AWS supports at http://www.amazonaws.cn/iam/details/mfa/. For more information about using a virtual MFA device, see Using a Virtual MFA Device with AWS (p. 71). For a list of apps that you can use as virtual MFA devices, see the **Virtual MFA Applications** section at http://www.amazonaws.cn/iam/details/mfa/.

   Note
   You can enable one MFA device per account or IAM user.

2. *Enable the MFA device.* You can enable the MFA device for use with AWS using the AWS Management Console, the IAM command line tools, or the IAM API.

   For information about enabling an MFA device, see either Using a Virtual MFA Device with AWS (p. 71) or Enabling a Hardware MFA Device for Use with AWS (p. 75).

3. *Use the MFA device when logging on or accessing AWS resources.* For access to an AWS website, you need a user name, password, and MFA code (an OTP). For access to MFA-protected APIs, you need access keys, the device serial number (hardware) or ARN (virtual device), and an MFA code.

   For information about user passwords, see Credentials (Passwords, Access Keys, and MFA devices) (p. 56). For information about using MFA with the AWS Management Console, see MFA Devices and Your IAM-Enabled Sign-in Page (p. 38). For information about the AWS service APIs that use MFA, go to AWS Multi-Factor Authentication FAQs.

Checking MFA Status

Use the IAM console to verify that an MFA device is enabled and configured for the root account or IAM user. In this section, you'll learn how to check whether a root account or IAM user has a valid MFA device enabled.

To check the MFA status of a root account

1. Sign in to the AWS Management Console using your AWS account (root) credentials and then open the IAM console at https://console.amazonaws.cn//iam/.
2. Check under **Security Status** to see whether MFA is enabled or disabled. If MFA has not been activated, an alert symbol (⚠) is displayed next to **Activate MFA on your root account**.
3. If you want to enable MFA for the account, open the **Activate MFA on your root account** row and then click **Manage MFA**.

To check the MFA status of an IAM user

1. Open the IAM console at https://console.amazonaws.cn//iam/.
2. In the navigation pane, click **Users**.
3. Click the name of the user who you want to check MFA for, and then open the **Security Credentials** section.

4. If no MFA device is active for the user, the console displays **No** next to **Multi-Factor Authentication Device**. If the user has an MFA device enabled, the **Multi-Factor Authentication Device** item shows a value for the device.

   The **Multi-Factor Authentication Device** is either the ARN for a virtual device, such as `arn:aws:iam::123456789012:mfa/user`, or it's the device serial number for a hardware device (usually the number from the back of the device), such as `GAHT12345678`.
5. Click **Manage MFA Device** to change the current setting.

   For virtual device information, see Using a Virtual MFA Device with AWS (p. 71).

   For hardware device information, see Enabling a Hardware MFA Device for Use with AWS (p. 75).

Using a Virtual MFA Device with AWS

To use a virtual MFA device with AWS, you must configure it for use with AWS, and then enable it. In this section you'll learn what a virtual MFA device is and what you need to do to configure and enable it.

A virtual MFA device uses a software application that generates six-digit authentication codes that are compatible with the Time-Based One-Time Password (TOTP) standard, as described in RFC 6238. The software application can run on mobile hardware devices, including smartphones. Most virtual MFA applications allow you to host more than one virtual MFA device, which makes them more convenient than hardware MFA devices. However, you should be aware that because a virtual MFA might be run on a less secure device such as a smartphone, a virtual MFA might not provide the same level of security as a hardware MFA device.

You can enable one MFA device per account.

For a list of virtual MFA apps that you can use on smartphones and tablets (including Google Android, Apple iPhone and iPad, and Windows Phone), see the **Virtual MFA Applications** section at http://www.amazonaws.cn/iam/details/mfa/. Note that AWS requires a virtual MFA app that produces a six-digit OTP.

Topics

Configuring and Enabling a Virtual MFA Device for a User

You can use IAM in the AWS Management Console to configure and enable a virtual MFA device for a user under your account. This section shows you how to use the console to complete these tasks. If you prefer to use the CLI or API, see Configuring and Managing a Virtual MFA Device for Your AWS Account (CLI and API) (p. 74).

Note

You must have physical access to the user's MFA device in order to configure MFA. For example, if you are configuring MFA for a user who will use a smartphone to generate an OTP, you must have the smartphone available in order to finish the wizard. Because of this, you might want to let users configure and manage their own virtual MFA devices. If so, you must grant users the permissions to perform the necessary IAM actions. For more information and for an example of an IAM policy that grants these permissions, see Allow Users to Manage Their Own Virtual MFA

Devices (AWS Management Console) (p. 102) and Allow Users to Manage Their Own Virtual MFA Devices (AWS CLI or API) (p. 103).

To configure and enable a virtual MFA device for a user

1. Sign in to the AWS Management Console and open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, click **Users**.
3. Click the name of the user who you want to enable MFA for, and then open the **Security Credentials** section.
4. Click **Manage MFA Device**.
5. In the **Manage MFA Device** wizard, select **A virtual MFA device** and then click **Next Step**.
6. Confirm that a virtual MFA application is installed on the user's mobile device and then click **Next Step**. (For a list of apps that you can use as virtual MFA devices, see Multi-Factor Authentication.) IAM generates and displays configuration information for the virtual MFA device, including a QR code similar to the following graphic.
7. With the **Manage MFA Device** wizard still open, open the virtual MFA application on the device. If the device supports QR codes, the easiest way to configure the application is to use the application to scan the QR code. If you cannot scan the code, you can enter the secret configuration key manually.

   - To use the QR code to configure the virtual MFA device, follow the app instructions for scanning the code. For example, you might need to tap the camera icon or tap a command like **Scan account barcode**, and then use the device's camera to scan the code.
   - If you cannot scan the code, enter the configuration information manually by typing the **Secret Configuration Key** value into the application. For example, to do this in the AWS Virtual MFA application, tap **Manually add account**, and then type the secret configuration key and click **Create**.

   > **Note**
   > The QR code and secret configuration key are unique and cannot be reused.

   When you are finished configuring the device, the device starts generating six-digit numbers.
8. In the IAM **Manage MFA Device** wizard, in the **Authentication Code 1** box, type the six-digit number that's currently displayed by the MFA device. Wait 30 seconds for the device to generate a new number, and then type the new six-digit number into the **Authentication Code 2** box.
9. Click **Next Step**.

The device is ready for use with AWS. For information about using MFA with the AWS Management Console, see MFA Devices and Your IAM-Enabled Sign-in Page (p. 38).

Configuring and Managing a Virtual MFA Device for Your AWS Account (AWS Management Console)

You can use IAM in the AWS Management Console to configure and enable a virtual MFA device for your root account. This section describes how to use the console to complete these tasks. If you prefer to use the CLI or API, see Configuring and Managing a Virtual MFA Device for Your AWS Account (CLI and API) (p. 74).

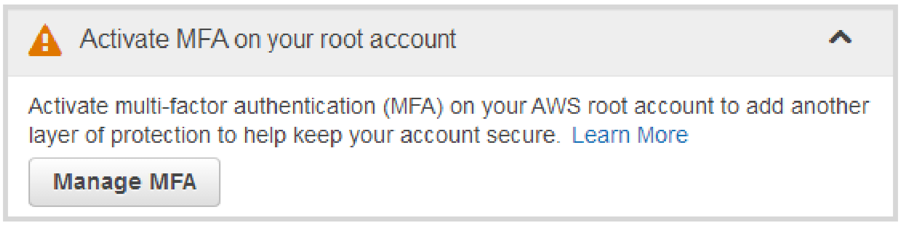
To manage MFA devices for the AWS account, you must be signed in to AWS using your root account credentials. You cannot manage MFA devices for the root account using other credentials.

To configure and enable a virtual MFA device for use with your root account

1. Sign in to the AWS Management Console and open the IAM console at https://console.amazonaws.cn/iam/.

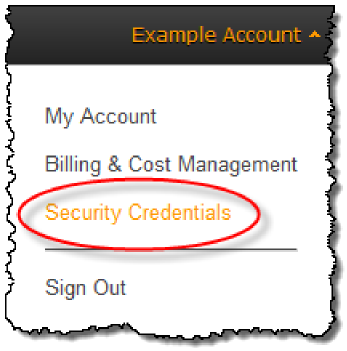
   Important
   To manage MFA devices for the AWS account, you must sign in to AWS using your root account credentials. You cannot manage MFA devices for the root account using other credentials.

2. **Option 1**: Click **Dashboard**, and under **Security Status**, expand **Activate MFA on your root account**.

   
   Activate MFA on your root account

   Activate multi-factor authentication (MFA) on your AWS root account to add another layer of protection to help keep your account secure. Learn More

   Option 2: On the right side of the navigation bar, click your account name, click **Security Credentials**, and then expand the **Multi-Factor Authentication (MFA)** section on the page.

3. Click the **Manage MFA** or **Activate MFA** button, depending on which option you chose in the preceding step.
4. In the wizard, select **A virtual MFA device** and then click **Next Step**.
5. Confirm that a virtual MFA application is installed on the device, then click **Next Step**. IAM generates and displays configuration information for the virtual MFA device, including a QR code similar to the following graphic.

6. With the **Manage MFA Device** wizard still open, open the virtual MFA application on the device. The easiest way to configure the application is to use the application to scan the QR code. If you cannot scan the code, you can enter the configuration information manually.

    - To use the QR code to configure the virtual MFA device, follow the app instructions for scanning the code. For example, you might need to tap the camera icon or tap a command like **Scan account barcode**, and then use the device's camera to scan the code.
    - If you cannot scan the code, enter the configuration information manually by typing the **Secret Configuration Key** value into the application. For example, to do this in the AWS Virtual MFA application, tap **Manually add account**, and then type the secret configuration key and click **Create**.

    Important
    Make a secure backup of the QR code or secret configuration key, or make sure that you enable multiple virtual MFA devices for your account. If the virtual MFA device is unavailable (for example, if you lose the smartphone where the virtual MFA app is configured), you will not be able to sign in to your account and you will have to contact customer service to remove MFA protection for the account.

    Note
    The QR code and secret configuration key generated by IAM are tied to your AWS account, and cannot be used with a different account. They can, however, be reused to configure a new MFA device for your account, in case you lose access to the original MFA device.

    When you are finished configuring the device, the device starts generating six-digit numbers.
7. Type the six-digit number that's currently displayed by the MFA device. Wait up to 30 seconds for the device to generate a new number, and then type the new six-digit number into the **Authentication Code 2** box.
8. Click **Next Step**, and then click **Finish**.

The device is ready for use with AWS. For information about using MFA with the AWS Management Console, see MFA Devices and Your IAM-Enabled Sign-in Page (p. 38).

Configuring and Managing a Virtual MFA Device for Your AWS Account (CLI and API)

The following list shows the command line commands or API actions to use to configure and enable a virtual MFA device.

- Create the configuration information for the virtual MFA device.

  CLI: `aws iam create-virtual-mfa-device`

  API: `CreateVirtualMFADevice`
- Enable the device for use with AWS.

  CLI: `aws iam enable-mfa-device`

  API: `EnableMFADevice`
- Deactivate a device.

  CLI: `aws iam deactivate-mfa-device`

  API: `DeactivateMFADevice`
- List virtual MFA devices.

  CLI: `aws iam list-virtual-mfa-devices`

  API: `ListVirtualMFADevices`
- Re-sync an MFA device.

  CLI: `aws iam resync-mfa-device`

  API: `ResyncMFADevice`
- Delete a virtual MFA device.

  CLI: `aws iam delete-virtual-mfa-device`

  API: `DeleteVirtualMFADevice`

Installing the AWS Virtual MFA Mobile Application

To download and install the AWS Virtual MFA application, go to the Amazon Appstore for Android and locate the AWS Virtual MFA application. Download the application and follow the on-screen instructions to complete the installation.

For information about configuring and enabling the AWS Virtual MFA device for use with AWS, see Using a Virtual MFA Device with AWS (p. 71).

To open the Amazon Appstore for Android on a smartphone, scan this code with any QR code reader application.

Enabling a Hardware MFA Device for Use with AWS

You can enable a hardware MFA device using the AWS Management Console, the command line, or the IAM API. The following procedure shows you how to use the AWS Management Console to enable the device for a user under your AWS account. To enable an MFA device for your root account, see Enabling a Hardware MFA Device for Your AWS Account (p. 76). You can enable one MFA device per account or IAM user.

> **Note**
> If you want to enable the device from the command line, use aws iam enable-mfa-device. To enable the MFA device with the IAM API, use the `EnableMFADevice` action.

Enabling a User's Hardware MFA Device

To use IAM in the AWS Management Console to enable a hardware MFA device for a user

1. Sign in to the AWS Management Console and open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, click **Users**.
3. Click the name of the user who you want to enable MFA for, and then open the **Security Credentials** section.
4. Click **Manage MFA Device**.
5. In the Manage MFA Device wizard, select **A hardware MFA device** and then click **Next Step**.
6. Enter the device serial number. The serial number is usually on the back of the device.
7. In the **Authentication Code 1** box, type the six-digit number displayed by the MFA device. You might need to press the button on the front of the device to display the number.

8. Wait 30 seconds while the device refreshes, and then type the next six-digit number into the **Authentication Code 2** box. You might need to press the button on the front of the device again to display the second number.
9. Click **Associate MFA**.

The device is ready for use with AWS. For information about using MFA with the AWS Management Console, see MFA Devices and Your IAM-Enabled Sign-in Page (p. 38).

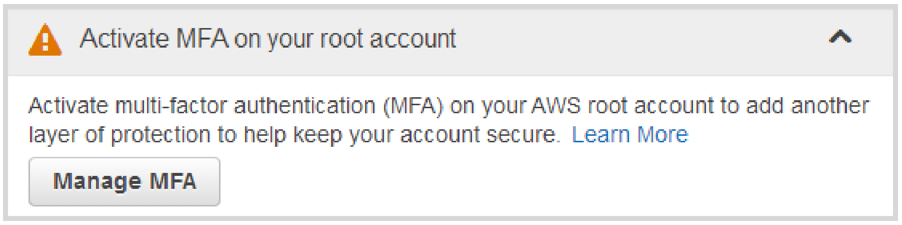
Enabling a Hardware MFA Device for Your AWS Account

To enable the MFA device for your AWS account

1. Sign in to the AWS Management Console and open the IAM console at https://console.amazonaws.cn/iam/.

   Important
   To manage MFA devices for the AWS account, you must sign in to AWS using your root account credentials. You cannot manage MFA devices for the root account using other credentials.

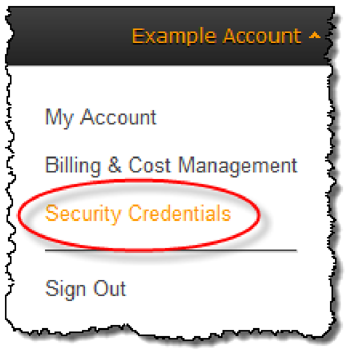
2. **Option 1**: Click **Dashboard**, and under **Security Status**, expand **Activate MFA on your root account**.



Activate MFA on your root account

Activate multi-factor authentication (MFA) on your AWS root account to add another layer of protection to help keep your account secure. Learn More

Option 2: On the right side of the navigation bar, click your account name, click **Security Credentials**, and then expand the **Multi-Factor Authentication (MFA)** section on the page.



3. Click the **Manage MFA** or **Activate MFA** button, depending on which option you chose in the preceding step.
4. In the wizard, select **A hardware MFA device** and then click **Next Step**.
5. In the **Serial Number** box, enter the serial number displayed on the back of the MFA device.
6. In the **Authentication Code 1** box, type the six-digit number displayed by the MFA device. You might need to press the button on the front of the device to display the number.

7. Wait 30 seconds while the device refreshes, and then type the next six-digit number into the **Authentication Code 2** box. You might need to press the button on the front of the device again to display the second number.
8. Click **Next Step**. The MFA device is now associated with the AWS account.

   The next time you sign in using the AWS account credentials, you will need to enter a code from the MFA device.

For information about using MFA with the AWS Management Console, see MFA Devices and Your IAM-Enabled Sign-in Page (p. 38).

Synchronizing an MFA Device

It's possible for an MFA device to get out of synchronization. If the device is not synchronized when the user tries to use it, the user's login will fail. If the user is using the MFA device to sign in to the AWS Management Console, IAM prompts the user to resynchronize the device.

If the user is using the MFA device with the API, IAM has an API call that performs synchronization. In this case, we recommend that you give your users with MFA devices permission to access this API call. You should build a tool based on that API call that lets your users resynchronize their devices whenever they need to. To use the API to synchronize an MFA device for a user, use `ResyncMFADevice`. To use the command line to synchronize the device, use aws iam resync-mfa-device.

You can also use the AWS Management Console to resynchronize the device for a user under your AWS account.

To resynchronize a user's MFA device

1. On the IAM console, click **Users**, and then click the name of the user you want to deactivate a device for.

2. Open the **Security Credentials** section, and then click **Manage MFA Device**.
3. Select **Resynchronize MFA device**.
4. Type the six-digit number the MFA device is displaying into the **Authentication Code 1** box. If you are using a hardware MFA device, you will need to press the button on the front of the device to display the number.
5. Wait 30 seconds while the device refreshes, and then type the next six-digit number into the **Authentication Code 2** box. If you are using a hardware device, you will need to press the device button again to display the second number.
6. Click **Next Step**.

Deactivating an MFA Device

You can temporarily disable an MFA device by deactivating it. If you are using the IAM API or CLI, then you deactivate an MFA device with the IAM `DeactivateMFADevice` API action or the aws iam deactivate-mfa-device command.

> **Note**
> If you use the API or CLI to delete a user from your AWS account, you must deactivate or delete the user's MFA device as part of the process of removing the user. For more information about deleting users, see Deleting an IAM User from Your AWS Account (p. 48).

If you are using the AWS Management Console to deactivate the device for a user under your AWS account, the following procedure describes the steps. The process to deactivate an MFA device for the root account is described in Deactivating Your AWS Account's MFA Device (p. 78).

Deactivating a User's MFA Device

To deactivate a user's MFA device

1. On the IAM console, click **Users**, and then click the name of the user you want to deactivate a device for.
2. Open the **Security Credentials** section, and then click **Manage MFA Device**.
3. Select **Deactivate MFA device**.
4. Click **Next Step**.

The user's device is deactivated.

Deactivating Your AWS Account's MFA Device

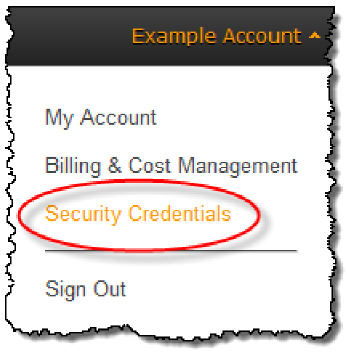
You can temporarily deactivate our AWS account's MFA device.

To deactivate the MFA device for your AWS account

1. Use your root credentials to sign in to the AWS Management Console.

   > **Important**
   > To manage MFA devices for the AWS account, you must sign in to AWS using your root account credentials. You cannot manage MFA devices for the root account using other credentials.

2. On the right side of the navigation bar, click your account name, then click **Security Credentials**.



3. Click to expand the **Multi-Factor Authentication (MFA)** section on the page.
4. In the row for the MFA device you want to deactivate, click **Deactivate**.

The MFA device is deactivated for the AWS account.

What If an MFA Device Is Lost or Stops Working?

If an MFA device stops working, is lost, or is destroyed, and you can't sign in to the AWS portal or the AWS Management Console, then you need to deactivate the device. AWS can help you deactivate the device. The way you get help depends on whether an MFA device is assigned to the root account or to a user under an AWS account.

Note
If the device appears to be functioning properly, but you cannot use it to access your AWS resources, then it simply might be out of synchronization with the AWS system. For information about synchronizing an MFA device, see Synchronizing an MFA Device (p. 77).

To get help for an MFA device associated with an AWS root account

1. Go to the AWS Contact Us page for help with disabling AWS MFA so that you can temporarily access secure pages on the AWS website and the AWS Management Console using just your user name and password.
2. Change your AWS password in case an attacker has stolen the authentication device and might also have your current password.
3. If you are using a hardware MFA device, contact the third-party provider Gemalto using their website for help fixing or replacing the device. If the device is a virtual MFA device, delete the old MFA account entry on the device before creating a new one.
4. After you have the new physical MFA device or you have completed deleting the old entry from the mobile device, return to the AWS website and activate the MFA device to re-enable AWS MFA for your AWS account. To manage a hardware MFA for your AWS account, go to the AWS Security Credentials page.

To get help for an MFA device associated with an IAM user

- Contact the system administrator or other person who gave you the user name and password for the IAM user. They will need to deactivate the MFA device using the procedure described at Deactivating a User's MFA Device (p. 78).

Configuring MFA-Protected API Access

IAM policies let you specify which APIs a user is allowed to call. In some cases, you might want the additional security of requiring a user to be authenticated with AWS multi-factor authentication (MFA) before the user is allowed to perform particularly sensitive actions.

For example, you might have a policy that allows users to perform the Amazon EC2 `RunInstances`, `DescribeInstances`, and `StopInstances` actions. But you might want to restrict a destructive action like `TerminateInstances` and make sure that users can perform that action only if they have been authenticated with an AWS MFA device.

Topics

Overview

Adding MFA protection to APIs involves these tasks:

- An administrator configures an AWS MFA device for each user who will make API requests that require MFA authentication. This process is described at Setting Up an MFA Device (p. 70).
- An administrator creates policies that include a condition that checks whether the user has been authenticated using an AWS MFA device.
- The user calls one of the AWS Security Token Service APIs that support MFA parameters: AssumeRole or GetSessionToken, depending on the scenario for MFA protection, as explained later. As part of the call, the user includes the device identifier for the device that's associated with the user, as well as the time-based one-time password (TOTP) that is generated by the device. In each case, the user gets back temporary security credentials that the user then uses to make additional requests to AWS.

  Note
  MFA protection for an AWS service's APIs is available only if the service supports temporary security credentials. For a list of these services, see Using Temporary Security Credentials to Access AWS.

If the user is denied access to APIs because of an authorization failure, AWS returns an "Access Denied" error message (as it does for any unauthorized access). With MFA-protected API policies in place, AWS denies access to the APIs specified in the policies if the user attempts to use the APIs without valid MFA authentication, or if the time of the request for the APIs is beyond the duration specified in the policy. The user must re-authenticate with MFA by requesting new temporary security credentials using an MFA code and device serial number.

IAM Policies with MFA Conditions

Policies with MFA conditions can be attached to the following:

- An IAM user or group.
- A resource such as an Amazon S3 bucket, Amazon SQS queue, or Amazon SNS topic.
- The trust policy of an IAM role that will be assumed by a user.

An MFA condition in a policy can be used to check the following properties:

- Existence—To simply verify that the user has been authenticated with MFA, check that the `aws:MultiFactorAuthPresent` key `True` in a `bool` condition.
- Duration—If you want to grant access only within a specified time after MFA authentication, use a numeric condition type to compare the `aws:MultiFactorAuthAge` key's age to a value (such as 3600 seconds). Note that the `aws:MultiFactorAuthAge` key is not present if MFA was not used.

The following example shows the trust policy of an IAM role that includes an MFA condition that tests for the existence of MFA authentication. This policy lets users from the AWS account specified in the `Principal` element (replace `ACCOUNT-B-ID` with a valid AWS account ID) assume the role that this policy is attached to, but only if the user has been MFA authenticated.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Principal": {"AWS": "ACCOUNT-B-ID"},
    "Action": "sts:AssumeRole",
    "Condition": {"Bool": {"aws:MultiFactorAuthPresent": "true"}}
  }
}
```

For more information on the condition types for MFA, see Available Keys for Conditions (p. 222), Numeric Conditions (p. 227), and Existence of Condition Keys (p. 231)

Choosing Between `GetSessionToken` and `AssumeRole`

AWS STS provides two APIs that let users pass MFA information: `GetSessionToken` and `AssumeRole`. The API that the user calls to get temporary security credentials depends on which of the following scenarios applies.

You use `GetSessionToken` for these scenarios:

- Access to APIs in the current account, except for IAM and AWS STS APIs. (If you need access to those APIs with MFA protection, you can use `AssumeRole`).
- Access to resources that are protected with resource-based policies that include an MFA condition.

You use `AssumeRole` for these scenarios:

- Cross-account delegation. You can add MFA protection to the process of assuming a role in another account.
- Access to APIs in the current account, including IAM and AWS STS.

Details about how to implement these scenarios are provided later in this document.

Important Points About MFA-Protected API Access

It's important to understand the following aspects of MFA protection for APIs:

- MFA protection is available only by using temporary security credentials, which in turn must be obtained using `AssumeRole` or `GetSessionToken`.
- You cannot use MFA-protected API access with root account credentials.

- Federated users cannot be assigned an MFA device for use with AWS services, so they cannot access AWS resources controlled by MFA. (See next point.)
- Other AWS STS APIs that return temporary credentials do not support MFA authentication. For `AssumeRoleWithWebIdentity` and `AssumeRoleWithSAML`, the user is authenticated using an external provider and AWS cannot determine whether that provider required MFA. For `GetFederationToken`, MFA authentication is not necessarily associated with a specific user.
- Similarly, long-term credentials (IAM user access keys and root account access keys) do not support MFA authentication because these don't expire.
- `AssumeRole` and `GetSessionToken` can also be called without MFA authentication information. In that case, the caller gets back temporary security credentials, but the session information for those temporary credentials does not indicate that the user authenticated using MFA.
- You establish MFA protection for APIs by adding MFA conditions to policies. If a policy doesn't include the condition for MFAs, the policy does not enforce MFA authentication. For cross-account delegation, if the role's trust policy doesn’t include an MFA condition, there is no MFA protection for the API calls made using the role's temporary security credentials.
- When you allow users from another AWS account to access resources in your account, *even when you require multi-factor authentication*, the security of your resources depends on the configuration of the other (or *trusted*) account. Any identity within the trusted account that has permission to create virtual MFA devices can construct an MFA claim to satisfy that part of your role's trust policy. Before you make another account's access to your AWS resources conditional upon multi-factor authentication, ensure that the trusted account's owner follows best practices and restricts access to MFA device-management APIs to specific, trusted administrators.
- If a policy includes an MFA condition, a request is denied if users have not been MFA authenticated, or if they provide an invalid MFA device identifier or invalid TOTP.

Scenario: MFA Protection for Cross-Account Delegation

In this scenario, you want to delegate access to IAM users in another account, but only if the users have been authenticated using an AWS MFA device. (For more information about cross-account delegation, see Roles - Terms and Concepts (p. 116).

Imagine that you have Account A (the trusting account), with the IAM user Alice, who has administrator permission. She wants to grant access to user Bob in Account B (the trusted account), but wants to make sure that Bob had been authenticated with MFA before he assumes the role.

1. In Account A, Alice creates an IAM role named `CrossAccountRole` and sets the principal in the role's trust policy to the account ID of Account B. The trust policy grants permission to the AWS STS `AssumeRole` action.

   Alice also adds an MFA condition to the trust policy, as in the following example.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Principal": {"AWS": "ACCOUNT-B-ID"},
    "Action": "sts:AssumeRole",
    "Condition": {"Bool": {"aws:MultiFactorAuthPresent": "true"}}
  }
}
```

2. Alice adds an access policy to the role that specifies what the role is allowed to do. The access policy for a role with MFA protection is no different than any other role access policy. The following example

shows the policy that Alice adds to the role; it allows Bob to perform any DynamoDB action on the table `Books` in Account A.

> **Note**
> The access policy does not include an MFA condition. It is important to understand that the MFA authentication is used only to determine whether a user can assume the role. Once the user has assumed the role, no further MFA checks are made.

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": ["dynamodb:*"],
    "Resource": ["arn:aws:dynamodb:region:ACCOUNT-A-ID:table/Books"]
  }]
}
```

3. In Account B, an administrator makes sure that user Bob has been configured with an AWS MFA device and that he knows the ID of the device—that is, the serial number if it's a hardware MFA device, or the device's ARN if it's a virtual MFA device.
4. In Account B, the administrator attaches the following policy to user Bob that allows him to call the `AssumeRole` action. The resource is set to the ARN of the role that Alice created in step 1. Notice that this policy does not contain an MFA condition.

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": ["sts:AssumeRole"],
    "Resource": ["arn:aws:iam::ACCOUNT-A-ID:role/CrossAccountRole"]
  }]
}
```

5. In Account B, Bob (or an application that Bob is running) calls `AssumeRole`. The call includes the ARN of the role to assume (`arn:aws:iam::`*`Account-A-ID`*`:role/CrossAccountRole`), the ID of the MFA device, and the current TOTP that Bob gets from his device.

   When Bob calls `AssumeRole`, AWS determines whether he has valid credentials, including MFA authentication. If so, Bob can perform any DynamoDB action on the table named `Books` in Account A.

   For an example of a program that calls `AssumeRole`, see Code Example: Calling AssumeRole with MFA Authentication (Python) (p. 91) later in this document.

Scenario: MFA Protection for Access to APIs in the Current Account

In this scenario, you want to make sure that a user in an account who requests access to sensitive actions has been authenticated using an AWS MFA device.

Imagine that you have Account A that contains a group of developers who need to work with Amazon EC2 instances. Ordinary developers can work with the instances, but they are not granted permissions for the `ec2:StopInstances` or `ec2:TerminateInstances` actions. You want to limit those privileged actions to just a few trusted users. To make absolutely sure that only your trusted users can stop or terminate instances—and not just anyone who might happen to get the access keys for one of those trusted users—you add MFA protection to a policy that allows these sensitive Amazon EC2 actions.

In this scenario, one of those trusted users is user Carol. User Alice is an administrator in Account A.

1. Alice makes sure that Carol has been configured with an AWS MFA device and that Carol knows the ID of the device—the serial number if it's a hardware MFA device, or the device's ARN if it's a virtual MFA device.
2. Alice creates a group named `EC2-Admins` and adds user Carol to the group.
3. Alice attaches the following policy to the `EC2-Admins` group. This policy grants users permission to call the Amazon EC2 `StopInstances` and `TerminateInstances` actions, but only if the user has authenticated using MFA.

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": [
      "ec2:StopInstances",
      "ec2:TerminateInstances"
    ],
    "Resource": ["*"],
    "Condition": {"Bool": {"aws:MultiFactorAuthPresent": "true"}}
  }]
}
```

4. If user Carol needs to stop or terminate an Amazon EC2 instance, she (or an application that she is running) calls `GetSessionToken`. The call includes the ID of the MFA device and the current TOTP that Carol gets from her device.
5. User Carol (or an application that Carol is using) uses the temporary credentials returned by `GetSessionToken` to call the Amazon EC2 `StopInstances` or `StopInstances` action.

   For an example of a program that calls `GetSessionToken`, see Code Example: Calling GetSessionToken with MFA Authentication (Python and C#) (p. 89) later in this document.

Calling IAM or AWS STS APIs in the Same Account

In this scenario, you want to make sure that a user in an account who requests access to IAM and STS actions has been authenticated using an AWS MFA device. This scenario is similar to the previous one, except that the user also needs to be able to request IAM actions or STS actions.

This scenario requires a different approach, because the temporary security credentials returned by `GetSessionToken` are blocked from calling APIs in IAM or AWS STS. However, you can use `AssumeRole` to grant permissions for IAM and STS actions the same way you grant users cross-account access, except that you are granting them permissions to call actions in their own account.

In this scenario, user Alice is an administrator in the account, and can perform all actions including IAM and STS actions. Alice creates two IAM users, Bob and Mary. Both are power users—they can perform any action in Alice's account except IAM and STS actions. Alice also creates a role that gives a user permissions to perform IAM and STS actions. However, to assume the role, the user must be MFA-authenticated. Alice will give Mary—but not Bob—permission to assume the role.

1. Alice creates an IAM group named Power Users and adds the following policy to the group. The policy allows members of the group to perform any actions in the account except IAM and STS actions.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["*"],
      "Resource": ["*"]
    },
    {
      "Effect": "Deny",
      "Action": [
        "iam:*",
        "sts:*"
      ],
      "Resource": ["*"]
    }
  ]
}
```

2. Alice creates IAM users Bob and Mary and adds them to the Power Users group.
3. Alice makes sure that Mary's IAM user is configured with an AWS MFA device and that she knows the ID of the device—that is, the serial number if it's a hardware MFA device, or the device's ARN if it's a virtual MFA device.
4. Alice creates an IAM role named IAMAdminRole and adds the following trust policy to the role. The principal in the role's trust policy is set to the account ID of Account A—that is, Alice's own account. The policy grants permission for any IAM user in Account A to assume the role, but only if the user has been MFA authenticated.

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Principal": {"AWS": "ACCOUNT-A-ID"},
    "Action": "sts:AssumeRole",
    "Condition": {"Bool": {"aws:MultiFactorAuthPresent": "true"}}
  }]
}
```

5. Alice adds an access policy to the role that specifies the permissions that will be granted to users who assume the role. For this scenario, the access policy allows the user who assumes the role to perform any IAM or STS action.

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": [
      "iam:*",
      "sts:*"
    ],
    "Resource": ["*"]
  }]
}
```

6. Alice attaches the following user policy to user Mary (but not to user Bob). This policy allows Mary to assume the IAMAdminRole role if she has been MFA authenticated.

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": ["sts:AssumeRole"],
    "Resource": ["arn:aws:iam::ACCOUNT-A-ID:role/IAMAdminRole"],
    "Condition": {"Bool": {"aws:MultiFactorAuthPresent": "true"}}
  }]
}
```

7. When Mary needs to administer IAM users in Account A, she calls `AssumeRole`. The call includes the ARN of the role to assume (arn:aws:iam::ACCOUNT-A-ID:role/IAMAdminRole), the ID of the MFA device, and the current TOTP that Mary gets from her device. She can then use the temporary security credentials returned by `AssumeRole` to perform IAM or STS tasks. Because he isn't MFA authenticated, Bob cannot assume the IAMAdminRole role, but because he's in the Power Users group, he can perform all other actions in the account.

   For an example of a program that calls `AssumeRole`, see Code Example: Calling AssumeRole with MFA Authentication (Python) (p. 91) later in this document.

Scenario: MFA Protection for Resources That Have Resource-Based Policies

In this scenario, you are the owner of an Amazon S3 bucket, an Amazon SQS queue, or an Amazon SNS topic and you want to make sure that any user who accesses the resource has been authenticated using an AWS MFA device.

This scenario illustrates a way to provide cross-account MFA protection without requiring users to assume a role first. If the user has authenticated using MFA and is able to get temporary security credentials using `GetSessionToken`, and if the user is in account that is trusted by the resource's policy, the user can access the resource.

Imagine that you are in Account A and create an S3 bucket. You want to grant access to this bucket to users who are in several different AWS accounts, but only if those users have MFA authentication.

In this scenario, user Alice is an administrator in Account A. User Charlie is an IAM user in Account C.

1. In Account A, Alice creates a bucket named `Account-A-bucket`.
2. Alice adds the bucket policy to the bucket. The policy allows any user in Account A, Account B, or Account C to perform the Amazon S3 `PutObject` and `DeleteObject` actions in the bucket. The policy includes an MFA condition.

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Principal": {"AWS": [
      "ACCOUNT-A-ID",
      "ACCOUNT-B-ID",
      "ACCOUNT-C-ID"
    ]},
    "Action": [
      "s3:PutObject",
      "s3:DeleteObject"
    ],
    "Resource": ["arn:aws:s3:::ACCOUNT-A-BUCKET-NAME/*"],
    "Condition": {"Bool": {"aws:MultiFactorAuthPresent": "true"}}
  }]
}
```

Note
Amazon S3 offers an MFA Delete feature for *root* account access (only). You can enable Amazon S3 MFA Delete when you set the versioning state of the bucket. Amazon S3 MFA Delete cannot be applied to an IAM user, and is managed independently from MFA-protected API access. An IAM user with permission to delete a bucket cannot delete a bucket with Amazon S3 MFA Delete enabled. For more information on Amazon S3 MFA Delete, see MFA Delete.

3. In Account C, an administrator makes sure that user Charlie has been configured with an AWS MFA device and that he knows the ID of the device—the serial number if it's a hardware MFA device, or the device's ARN if it's a virtual MFA device.
4. In Account C, Charlie (or an application that he is running) calls `GetSessionToken`. The call includes the ID or ARN of the MFA device and the current TOTP that Charlie gets from his device.
5. Charlie (or an application that he is using) uses the temporary credentials returned by `GetSessionToken` to call the Amazon S3 `PutObject` action to upload a file to `Account-A-bucket`.

   For an example of a program that calls `GetSessionToken`, see Code Example: Calling GetSessionToken with MFA Authentication (Python and C#) (p. 89) later in this document.

   Note
   The temporary credentials returned by `AssumeRole` won't work in this case because although the user must have MFA authentication to assume a role, the temporary credentials returned by `AssumeRole` don't include the MFA information that is required in order to meet the MFA condition in the policy.

Additional Examples of Policies with MFA Conditions

The following examples show additional ways that MFA conditions can be added to policies.

Note
All of the following examples show policies attached directly to an IAM user or group in your own AWS account. To MFA-protect APIs across accounts, you use IAM roles and put the MFA condition check in the role trust policy, not in the role access policy. For more information, see Scenario: MFA Protection for Cross-Account Delegation (p. 82).

Topics

Example 1: Granting access after recent MFA authentication (GetSessionToken)

The following example shows a policy attached to a user or group that grants Amazon EC2 access only if the user authenticated with MFA within the last hour (3600 seconds).

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": ["ec2:*"],
    "Resource": ["*"],
    "Condition": {"NumericLessThan": {"aws:MultiFactorAuthAge": "3600"}}
  }]
}
```

Example 2: Denying access to specific APIs without valid MFA authentication (GetSessionToken)

The following example shows a policy attached to a user or group that grants access to the entire Amazon EC2 API, but denies access to `StopInstances` and `TerminateInstances` if the user has not been MFA authenticated.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["ec2:*"],
      "Resource": ["*"]
    },
    {
      "Effect": "Deny",
      "Action": [
        "ec2:StopInstances",
        "ec2:TerminateInstances"
      ],
      "Resource": ["*"],
      "Condition": {"Bool": {"aws:MultiFactorAuthPresent": "true"}}
    }
  ]
}
```

Example 3: Denying access to specific APIs without recent valid MFA authentication (GetSessionToken)

The following example shows a policy attached to a user or group that grants access to the entire Amazon EC2 API, but denies access to `StopInstances` and `TerminateInstances` unless they authenticated using MFA authentication within the last hour.

> **Note**
> MFA-protected API policies using `Deny` statements that check for the numeric value of `aws:MultiFactorAuthAge` should include an existence check. AWS evaluates existence and duration independently; evaluating only for duration does not enforce a `Deny` condition if MFA has not been used at all.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["ec2:*"],
      "Resource": ["*"]
    },
    {
      "Effect": "Deny",
      "Action": [
        "ec2:StopInstances",
        "ec2:TerminateInstances"
      ],
      "Resource": ["*"],
      "Condition": {"Bool": {"aws:MultiFactorAuthPresent": "true"}}
    },
    {
      "Effect": "Deny",
      "Action": [
        "ec2:StopInstances",
        "ec2:TerminateInstances"
      ],
      "Resource": ["*"],
      "Condition": {"NumericGreaterThan": {"aws:MultiFactorAuthAge": "3600"}}
    }
  ]
}
```

Code Example: Calling GetSessionToken with MFA Authentication (Python and C#)

The following examples, written using the AWS SDK for Python (Boto) and AWS SDK for .NET, show how to call `GetSessionToken` and pass MFA authentication information. The temporary security credentials returned by `GetSessionTokenRole` are then used to list all Amazon S3 buckets in the account.

The policy attached to the user who runs this code (or to a group that the user is in) is assumed to include an MFA check. The policy also needs to grant the user permission to request the Amazon S3 `ListBuckets` action.

Example: Calling GetSessionToken with MFA Authentication Using Python

```
import boto
from boto.s3.connection import S3Connection
from boto.sts import STSConnection

# Prompt for MFA time-based one-time password (TOTP)
mfa_TOTP = raw_input("Enter the MFA code: ")

# The calls to AWS STS GetSessionToken must be signed using the access key ID
and secret
# access key of an IAM user. The credentials can be in environment variables
or in
# a configuration file and will be discovered automatically
# by the STSConnection() function. For more information, see the Python SDK
# documentation: http://boto.readthedocs.org/en/latest/boto_config_tut.html

sts_connection = STSConnection()
```

```
# Use the appropriate device ID (serial number for hardware device or ARN for
virtual device).
# Replace ACCOUNT-NUMBER-WITHOUT-HYPHENS and MFA-DEVICE-ID with appropriate
values.

tempCredentials = sts_connection.get_session_token(
    duration=3600,
    mfa_serial_number="arn:aws:iam::ACCOUNT-NUMBER-WITHOUT-HYPHENS:mfa/MFA-
DEVICE-ID",
    mfa_token=mfa_TOTP
)

# Use the temporary credentials to list the contents of an S3 bucket
s3_connection = S3Connection(
    aws_access_key_id=tempCredentials.access_key,
    aws_secret_access_key=tempCredentials.secret_key,
    security_token=tempCredentials.session_token
)

# Replace BUCKET-NAME with an appropriate value.
bucket = s3_connection.get_bucket(bucket_name="BUCKET-NAME")
objectlist = bucket.list()
for obj in objectlist:
    print obj.name
```

Example: Calling GetSessionToken with MFA Authentication Using C#

```
Console.Write("Enter MFA code: ");
string mfaTOTP = Console.ReadLine(); // Get string from user

/* The calls to AWS STS GetSessionToken must be signed using the access key ID
 and secret
   access key of an IAM user. The credentials can be in environment variables
or in
   a configuration file and will be discovered automatically
   by the AmazonSecurityTokenServiceClient constructor. For more information,
see
   http://docs.aws.amazon.com/AWSSdkDocsNET/latest/DeveloperGuide/net-dg-config-
creds.html
*/
AmazonSecurityTokenServiceClient stsClient =
    new AmazonSecurityTokenServiceClient();
GetSessionTokenRequest getSessionTokenRequest = new GetSessionTokenRequest();
getSessionTokenRequest.DurationSeconds = 3600;

// Replace ACCOUNT-NUMBER-WITHOUT-HYPHENS and MFA-DEVICE-ID with appropriate
values
getSessionTokenRequest.SerialNumber = "arn:aws:iam::ACCOUNT-NUMBER-WITHOUT-HY
PHENS:mfa/MFA-DEVICE-ID";
getSessionTokenRequest.TokenCode = mfaTOTP;

GetSessionTokenResponse getSessionTokenResponse =
    stsClient.GetSessionToken(getSessionTokenRequest);

// Extract temporary credentials from result of GetSessionToken call
GetSessionTokenResult getSessionTokenResult =
    getSessionTokenResponse.GetSessionTokenResult;
string tempAccessKeyId = getSessionTokenResult.Credentials.AccessKeyId;
string tempSessionToken = getSessionTokenResult.Credentials.SessionToken;
string tempSecretAccessKey = getSessionTokenResult.Credentials.SecretAccessKey;
SessionAWSCredentials tempCredentials = new SessionAWSCredentials(tempAccessKey
Id,
    tempSecretAccessKey, tempSessionToken);

// Use the temporary credentials to list the contents of an S3 bucket
// Replace BUCKET-NAME with an appropriate value
ListObjectsRequest S3ListObjectsRequest = new ListObjectsRequest();
S3ListObjectsRequest.BucketName = "BUCKET-NAME";
S3Client = AWSClientFactory.CreateAmazonS3Client(tempCredentials);
ListObjectsResponse S3ListObjectsResponse =
    S3Client.ListObjects(S3ListObjectsRequest);
foreach (S3Object s3Object in S3ListObjectsResponse.S3Objects)
{
    Console.WriteLine(s3Object.Key);
}
```

Code Example: Calling AssumeRole with MFA Authentication (Python)

The following example, written using the AWS SDK for Python (Boto), shows how to call `AssumeRole` and pass MFA authentication information. The temporary security credentials returned by `AssumeRole` are then used to list all Amazon S3 buckets in the account.

For more information about this scenario, see Scenario: MFA Protection for Cross-Account Delegation (p. 82).

```
import boto
from boto.s3.connection import S3Connection
from boto.sts import STSConnection

# Prompt for MFA time-based one-time password (TOTP)
mfa_TOTP = raw_input("Enter the MFA code: ")

# The calls to AWS STS AssumeRole must be signed using the access key ID and
secret
# access key of an IAM user. (The AssumeRole API can also be called using tem
porary
# credentials, but this example does not show that scenario.)
# The IAM user credentials can be in environment variables or in
# a configuration file and will be discovered automatically
# by the STSConnection() function. For more information, see the Python SDK
# documentation: http://boto.readthedocs.org/en/latest/boto_config_tut.html

sts_connection = STSConnection()

# Use appropriate device ID (serial number for hardware device or ARN for vir
tual device)
# Replace ACCOUNT-NUMBER-WITHOUT-HYPHENS, ROLE-NAME, and MFA-DEVICE-ID with
appropriate values
tempCredentials = sts_connection.assume_role(
    role_arn="arn:aws:iam::ACCOUNT-NUMBER-WITHOUT-HYPHENS:role/ROLE-NAME",
    role_session_name="AssumeRoleSession1",
    mfa_serial_number="arn:aws:iam::ACCOUNT-NUMBER-WITHOUT-HYPHENS:mfa/MFA-
DEVICE-ID",
    mfa_token=mfa_TOTP
)

# Use the temporary credentials to list the contents of an S3 bucket
s3_connection = S3Connection(
    aws_access_key_id=tempCredentials.credentials.access_key,
    aws_secret_access_key=tempCredentials.credentials.secret_key,
    security_token=tempCredentials.credentials.session_token
)

# Replace BUCKET-NAME with a real bucket name
bucket = s3_connection.get_bucket(bucket_name="BUCKET-NAME")
objectlist = bucket.list()
for obj in objectlist:
    print obj.name
```

Rotating Credentials

As a security best practice, we recommend that you, an administrator, or your users regularly rotate (change) the credentials for IAM users in your account. You can apply a password policy to your account to require all your IAM users to rotate their passwords, and you can choose how often they must do so.

For more information about setting a password policy in your account, see Setting an Account Password Policy for IAM Users (p. 59).

Important
If you use the AWS account credentials on a regular basis, we recommend that you also regularly rotate those. The account password policy does not apply to the AWS account credentials. IAM users cannot manage credentials for the AWS account, so you must use the AWS account's credentials (not a user's) to change the AWS account credentials. Note that we recommend against using the AWS account credentials for everyday work in AWS.

The following steps describe the general process for rotating an access key without interrupting your applications. These steps show the CLI and API commands for rotating access keys. You can also perform these tasks using the console; for details, see Creating, Modifying, and Viewing User Access Keys (AWS Management Console) (p. 68).

1. While the first access key is still active, create a second access key, which will be active by default. At this point, the user has two active access keys.

   CLI: `aws iam create-access-key`

   API: `CreateAccessKey`
2. Update all applications to use the new access key.
3. Change the state of the first access key to `Inactive`.

   CLI: `aws iam update-access-key`

   API: `UpdateAccessKey`
4. Using only the new access key, confirm that your applications are working well. If you need to, you can revert to using the original access key by switching its state back to `Active`.
5. Delete the first access key.

CLI: `aws iam delete-access-key`

API: `DeleteAccessKey`

For more information, see the following:

- How to rotate access keys for IAM users. This entry on the AWS Security Blog provides more information on key rotation.
- Creating, Modifying, and Viewing User Access Keys (AWS Management Console) (p. 68). This page describes how to use the AWS Management Console to manage access keys.
- Permissions for Administering IAM Users, Groups, and Credentials (p. 95). This page discusses how to grant permissions to IAM users so that they can manage their own credentials, including access keys.
- IAM Best Practices (p. 26). This page provides general recommendations for helping to secure your AWS resources.
- Setting an Account Password Policy for IAM Users (p. 59). This topic describes how to set a password policy on your AWS account, including how to require that IAM users rotate their passwords after a specified number of days.

Getting Credential Reports for Your AWS Account

You can generate and download a *credential report* that lists all users in your account and the status of their various credentials, including passwords, access keys, MFA devices, and signing certificates. You can get a credential report using the AWS Management Console, the AWS Command Line Interface (AWS CLI), or the IAM API.

You can use credential reports to assist in your auditing and compliance efforts. You can use the report to audit the effects of credential lifecycle requirements, such as password rotation. You can provide the report to an external auditor, or grant permissions to an auditor so that he or she can download the report directly.

You can generate a credential report as often as once every four hours. When you request a report, IAM first checks whether a report for the account has been generated within the past four hours. If so, the most recent report is downloaded. If the most recent report for the account is more than four hours old, or if there are no previous reports for the account, IAM generates and downloads a new report.

Credential reports are downloaded as comma-separated values (CSV) files. You can open CSV files with common spreadsheet software to perform analysis, or you can build an application that consumes the CSV files programmatically and performs custom analysis.

The following list contains the columns in the CSV file and a short description of the data in each column:

user
: The friendly name of the user.

arn
: The Amazon Resource Name (ARN) of the user. For more information about ARNs, see IAM ARNs (p. 9).

user_creation_time
: The date and time when the user was created, in ISO 8601 date-time format.

password_enabled
: When the user has a password, this value is `TRUE`. Otherwise it is `FALSE`. The value for the AWS account (root) is always `not_supported`.

password_last_used

The date and time when the AWS account (root) or IAM user's password was last used to sign in to an AWS website, in ISO 8601 date-time format. AWS websites that capture a user's last sign-in time are the AWS Management Console, the AWS Discussion Forums, and the AWS Marketplace. If a password is used more than once in a five minute span, only the first use is recorded in this field.

When a user's password has never been used, or when there is no sign-in data associated with the password, the value for this field is `no_information`. One reason for having no sign-in data is that a password has not been used since IAM began capturing sign-in times. If a user does not have a password, the value for this field is `N/A`.

password_last_changed

The date and time when the user's password was last set, in ISO 8601 date-time format. If the user does not have a password, the value in this field is `N/A` (not applicable). The value for the AWS account (root) is always `not_supported`.

password_next_rotation

When the account has a password policy that requires password rotation, this field contains the date and time, in ISO 8601 date-time format, when the user is required to set a new password. The value for the AWS account (root) is always `not_supported`.

mfa_active

When a multi-factor authentication (MFA) device has been enabled for the user, this value is `TRUE`. Otherwise it is `FALSE`.

access_key_1_active

When the user has an access key and that access key's status is `Active`, this value is `TRUE`. Otherwise it is `FALSE`.

access_key_1_last_rotated

The date and time, in ISO 8601 date-time format, when the user's access key was created or last changed. If the user does not have an active access key, the value in this field is `N/A` (not applicable).

access_key_2_active

When the user has a second access key and that key's status is `Active`, this value is `TRUE`. Otherwise it is `FALSE`.

> **Note**
> Users can have up to two access keys, to make rotation easier. For more information about rotating access keys, see Rotating Credentials (p. 92).

access_key_2_last_rotated

The date and time, in ISO 8601 date-time format, when the user's second access key was created or last changed. If the user does not have a second active access key, the value in this field is `N/A` (not applicable).

cert_1_active

When the user has an X.509 signing certificate and that certificate's status is `Active`, this value is `TRUE`. Otherwise it is `FALSE`.

cert_1_last_rotated

The date and time, in ISO 8601 date-time format, when the user's signing certificate was created or last changed. If the user does not have an active signing certificate, the value in this field is `N/A` (not applicable).

cert_2_active

When the user has a second X.509 signing certificate and that certificate's status is `Active`, this value is `TRUE`. Otherwise it is `FALSE`.

> **Note**
> Users can have up to two X.509 signing certificates, to make certificate rotation easier. For more information about rotating signing certificates, see Rotating Credentials (p. 92).

cert_2_last_rotated

The date and time, in ISO 8601 date-time format, when the user's second signing certificate was created or last changed. If the user does not have a second active signing certificate, the value in this field is `N/A` (not applicable).

Getting Credential Reports (AWS Management Console)

You can use the AWS Management Console to download a credential report as a comma-separated values (CSV) file.

To download a credential report using the AWS Management Console

1. Sign in to the AWS Management Console and open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, click **Credential Report**.
3. Click the **Download Report** button.

Getting Credential Reports (CLI or API)

To generate a credential report, use the following commands:

- AWS CLI: `aws iam generate-credential-report`
- IAM API: `GenerateCredentialReport`

To retrieve a credential report, use the following commands:

- AWS CLI: `aws iam get-credential-report`
- IAM API: `GetCredentialReport`

Permissions for Administering IAM Users, Groups, and Credentials

If you are signed in using AWS account (root) credentials, there are no restrictions on administering IAM users or groups, or on managing their credentials. However, IAM users must explicitly be given permissions to administer users or credentials for themselves or for other IAM users. This topic describes IAM policies that let IAM users manage other users and user credentials.

Topics

Overview

In general, the permissions that are required in order to administer users, groups, and credentials correspond to the API actions for the task. For example, in order to create users, a user must have the `iam:CreateUser` permission (API command: `CreateUser`). To allow a user to create other IAM users, you could attach a policy like the following one to that user:

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "iam:CreateUser",
    "Resource": "*"
  }
}
```

In a policy, the value of the `Resource` element depends on the action and what resources the action can affect. In the preceding example, the policy allows a user to create any user (* is a wildcard that matches all strings). In contrast, a policy that allows users to change only their own access keys (API actions `CreateAccessKey` and `UpdateAccessKey`) typically has a `Resource` element where the ARN includes a variable that resolves to the current user's name, as in the following example (replace `ACCOUNT-ID-WITHOUT-HYPHENS` with your AWS account ID):

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "iam:CreateAccessKey",
      "iam:UpdateAccessKey"
    ],
    "Resource": "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:user/${aws:username}"

  }
}
```

In the previous example, `${aws:username}` is a variable that resolves to the user name of the current user. For more information about policy variables, see IAM Policy Variables Overview (p. 232).

Using a wildcard character (*) in the action name often makes it easier to grant permissions for all the actions related to a specific task. For example, to allow users to perform any IAM action, you can use `iam:*` for the action. To allow users to perform any action related just to access keys (create, list, update, and delete), you can use `iam:*AccessKey*` in the `Action` element of a policy statement. This gives the user permission to perform the `CreateAccessKey`, `DeleteAccessKey`, `ListAccessKeys`, and `UpdateAccessKey` actions. (If an action is added to IAM in the future that has "AccessKey" in the name, using `iam:*AccessKey*` for the `Action` element will also give the user permission to that new action.) The following example shows a policy that allows users to perform all actions pertaining to their own access keys (replace `ACCOUNT-ID-WITHOUT-HYPHENS` with your AWS account ID):

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "iam:*AccessKey*",
    "Resource": "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:user/${aws:username}"

  }
}
```

Some tasks, such as deleting a group, involve multiple actions—you have to first remove users from the group, then detach or delete the group's policies, and then actually delete the group. If you want a user to be able to delete a group, you must be sure to give the user permissions to perform all of the related actions.

Permissions for Working in the AWS Management Console

The preceding examples show policies that allow a user to perform the actions using the AWS Command Line Interface (AWS CLI) or the AWS SDKs. If users want to use the AWS Management Console to administer users, groups, and permissions, they need additional permissions. As users work with the console, the console issues requests to IAM to list users and groups, to get the policies associated with a user or group, to get AWS account information, and so on.

For example, if user Bob wants to use the AWS Management Console to change his own access keys, he starts by going to the console and clicking **Users**. This action causes the console to make a `ListUsers` request. If Bob doesn't have permission for the `iam:ListUsers` action, the console will be denied access when it tries to list users. As a result, Bob can't get to his own name and to his own access keys, even if he has permissions for the `CreateAccessKey` and `UpdateAccessKey` actions.

If you want to give users permissions to work in the AWS Management Console to administer users, groups, and credentials, you need to include permissions for the actions that the console performs. For some examples of policies you can use to grant a user the permissions they need to work in the AWS Management Console, see Example Policies for Administering IAM Resources (p. 97).

Example Policies for Administering IAM Resources

This section lists examples of IAM policies that let users perform tasks associated with managing IAM users, groups, and credentials. This includes policies that let users manage their own passwords, access keys, and multi-factor authentication (MFA) devices.

For examples of policies that let users perform tasks with other AWS services, like Amazon S3, Amazon EC2, and DynamoDB, see Example Policies for Administering AWS Resources (p. 203).

Topics

Allow Users to Manage Their Own Passwords (Using the My Password page)

If the account's password policy (p. 59) is set to allow all users to change their own passwords, you don't need to attach any permissions to individual users or groups. All users will be able to go to the My Password page (p. 66) in the AWS Management Console that lets them change their own password.

If the account's password policy is *not* set to allow all users to change their own passwords, you can attach the following policy to selected users or groups to allow those users to change their own passwords.

This policy only allows users to use the special My Password page in the console; it does not give users permissions to work through the dashboard in the IAM console.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "iam:ChangePassword",
      "iam:GetAccountPasswordPolicy"
    ],
    "Resource": "*"
  }
}
```

If users will run the `aws iam change-password` command using the AWS CLI, or make a request using the `ChangePassword` action without using the password page (that is, they will not use the console to change their password), they do not need the `iam:GetAccountPasswordPolicy` permission.

For information about letting selected users manage passwords using the **Users** section of the IAM console, see the next section.

Allow Users to Manage Their Own Passwords and Access Keys

The following policy allows a user to perform all actions in the AWS Management Console that pertain to the following:

- Creating, changing, or removing his or her own password. This includes the `CreateLoginProfile`, `DeleteLoginProfile`, `GetLoginProfile`, and `UpdateLoginProfile` actions.
- Creating or deleting his or her own access key (access key ID and secret access key). This includes the `CreateAccessKey`, `DeleteAccessKey`, `ListAccessKeys`, and `UpdateAccessKey` actions.

To use the following policy, replace `ACCOUNT-ID-WITHOUT-HYPHENS` with your AWS account ID.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iam:*LoginProfile",
        "iam:*AccessKey*"
      ],
     "Resource": "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:user/${aws:username}"

    },
    {
      "Effect": "Allow",
      "Action": [
        "iam:ListAccount*",
        "iam:GetAccount*",
        "iam:ListUsers"
      ],
      "Resource": "*"
    }
  ]
}
```

The actions in the preceding policy include wildcards (for example, `iam:*LoginProfile` and `iam:*AccessKey*`). This is a convenient way to spell out a set of related actions. If you want to remove permissions for any one of the related actions, you must instead list each of the individual actions. For example, if you don't want users to be able to delete a password, you have to individually list `iam:CreateLoginProfile`, `iam:GetLoginProfile`, and `iam:UpdateLoginProfile`, leaving out `iam:DeleteLoginProfile`.

The `iam:GetAccount*` and `iam:ListAccount*` permissions allow the user to see certain information on the IAM console dashboard, such as whether a password policy is enabled, how many groups the account has, what the account URL and alias are, etc. For example, the `GetAccountSummary` action returns a map object that contains a collection of information about the account that is then displayed on the IAM console dashboard.

The following policy is like the previous one, but excludes the permissions that are in the policy only for console access. This policy lets users manage their credentials using the AWS CLI, the AWS SDKs, or the IAM HTTP query API. To use the following policy, replace `ACCOUNT-ID-WITHOUT-HYPHENS` with your AWS account ID.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "iam:*LoginProfile",
      "iam:*AccessKey*"
    ],
    "Resource": "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:user/${aws:username}"

  }
}
```

Allow a User to List the Account's Groups, Users, Policies, and other information for Reporting Purposes

The following policy allows the user to whom it is attached to call any IAM action that starts with the string `Get` or `List`.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "iam:Get*",
      "iam:List*"
    ],
    "Resource": "*"
  }
}
```

The benefit of using `Get*` and `List*` for the actions is that if new types of entities are added to IAM in the future, the access granted in the policy to `Get*` and `List*` all actions would automatically allow the user to list those new entities.

Allow a User to Manage a Group's Membership

The following policy allows the user to whom it's attached to update the membership of the group called *MarketingGroup*. To use the following policy, replace `ACCOUNT-ID-WITHOUT-HYPHENS` with your AWS account ID.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "iam:AddUserToGroup",
      "iam:RemoveUserFromGroup",
      "iam:GetGroup"
    ],
    "Resource": "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:group/MarketingGroup"

  }
}
```

Allow a User to Manage IAM Users

The following policy allows a user to perform all the tasks associated with managing IAM users, but not to perform actions on other entities, such as creating groups or policies. Allowed actions include these:

- Creating the user (the `CreateUser` action).
- Deleting the user. This task requires permissions to perform all of the following actions: `DeleteSigningCertificate`, `DeleteLoginProfile`, `RemoveUserFromGroup`, and `DeleteUser`.
- Listing users in the account and in groups (the `GetUser`, `ListUsers` and `ListGroupsForUser` actions).
- Listing and removing policies for the user (the `ListUserPolicies`, `ListAttachedUserPolicies`, `DetachUserPolicy`, `DeleteUserPolicy` actions)
- Renaming or changing the path for the user (the `UpdateUser` action). The `Resource` element must include an ARN that covers both the source path and the target path.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowUsersToPerformUserActions",
      "Effect": "Allow",
      "Action": [
        "iam:CreateUser",
        "iam:ListUsers",
        "iam:GetUser",
        "iam:UpdateUser",
        "iam:DeleteUser",
        "iam:ListGroupsForUser",
        "iam:ListUserPolicies",
        "iam:ListAttachedUserPolicies",
        "iam:DeleteSigningCertificate",
        "iam:DeleteLoginProfile",
        "iam:RemoveUserFromGroup",
        "iam:DetachUserPolicy",
        "iam:DeleteUserPolicy"
      ],
      "Resource": "*"
    },
    {
      "Sid": "AllowUsersToSeeStatsOnIAMConsoleDashboard",
      "Effect": "Allow",
      "Action": [
        "iam:GetAccount*",
        "iam:ListAccount*"
      ],
      "Resource": "*"
    }
  ]
}
```

A number of the permissions included in the preceding policy are to allow access to user tasks via the AWS Management Console. If users will be performing user-related tasks only using the AWS CLI, the AWS SDKs, or the IAM HTTP query API, users might not need certain permissions. For example, if users already know the ARN of policies to detach from a user, they do not need the `iam:ListAttachedUserPolicies` permission. The exact list of permissions that a user requires depends on the tasks that the user must perform while managing other users.

The following permissions are in the policy to allow access to user tasks via the AWS Management Console:

- `iam:GetAccount*`
- `iam:ListAccount*`

Allow Users to Set Account Password Policy

You might give some users permissions to get and update your AWS account's password policy (p. 59). The following example policy shows how to grant these permissions.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "iam:GetAccountPasswordPolicy",
      "iam:UpdateAccountPasswordPolicy"
    ],
    "Resource": "*"
  }
}
```

Allow Users to Generate and Retrieve IAM Credential Reports

You can give some users permission to generate and download a report that lists all users in your AWS account and the status of their various credentials, including passwords, access keys, MFA devices, and signing certificates. The following example policy shows how to grant these permissions.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "iam:GenerateCredentialReport",
      "iam:GetCredentialReport"
    ],
    "Resource": "*"
  }
}
```

For more information about credential reports, see Getting Credential Reports for Your AWS Account (p. 93).

Allow Users to Manage Their Own Virtual MFA Devices (AWS Management Console)

In order to configure a virtual MFA device (p. 71), you must have physical access to the device. If users are using their own smartphones as virtual MFA devices, you might want to let them configure the devices themselves. If the following policy is attached to a user or to a group that the user is in, the user can configure and manage his or her own virtual MFA device using the AWS Management Console. To use the following policy, replace `ACCOUNT-ID-WITHOUT-HYPHENS` with your AWS account ID.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowUsersToCreateDeleteTheirOwnVirtualMFADevices",
      "Effect": "Allow",
      "Action": "iam:*VirtualMFADevice",
     "Resource": "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:mfa/${aws:username}"

    },
    {
      "Sid": "AllowUsersToEnableSyncDisableTheirOwnMFADevices",
      "Effect": "Allow",
      "Action": [
        "iam:DeactivateMFADevice",
        "iam:EnableMFADevice",
        "iam:ListMFADevices",
        "iam:ResyncMFADevice"
      ],
     "Resource": "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:user/${aws:username}"

    },
    {
      "Sid": "AllowUsersToListVirtualMFADevices",
      "Effect": "Allow",
      "Action": "iam:ListVirtualMFADevices",
      "Resource": "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:mfa/*"
    },
    {
      "Sid": "AllowUsersToListUsersInConsole",
      "Effect": "Allow",
      "Action": "iam:ListUsers",
      "Resource": "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:user/*"
    }
  ]
}
```

Allow Users to Manage Their Own Virtual MFA Devices (AWS CLI or API)

The following policy allows users to configure their own virtual MFA devices. Unlike the previous example, this policy allows the commands to be performed using the AWS CLI, the AWS SDKs, or the IAM HTTP query API, but does not include permissions that allow users to configure virtual MFA devices using the AWS Management Console. To use the following policy, replace `ACCOUNT-ID-WITHOUT-HYPHENS` with your AWS account ID.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowUsersToCreateDeleteTheirOwnVirtualMFADevices",
      "Effect": "Allow",
      "Action": "iam:*VirtualMFADevice",
     "Resource": "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:mfa/${aws:username}"

    },
    {
      "Sid": "AllowUsersToEnableSyncDisableTheirOwnMFADevices",
      "Effect": "Allow",
      "Action": "iam:*MFADevice*",
     "Resource": "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:user/${aws:username}"

    }
  ]
}
```

Allow All IAM Actions (Admin Access)

You might give some users administrative permissions to perform all actions in IAM, including managing passwords, access keys, user certificates, and MFA devices. The following example policy shows how to grant these permissions.

Caution
When you give a user full access to IAM, there is no limit to the permissions that user can grant. The user can create new IAM entities (users or roles) and grant those entities full access to all resources in your AWS account. When you give a user full access to IAM, you are effectively giving them full access to all resources in your AWS account. This includes access to delete all resources. You should grant these permissions to only trusted administrators, and you should enforce multi-factor authentication (MFA) for these administrators.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "iam:*",
    "Resource": "*"
  }
}
```

Using Identity Providers

As an alternative to creating IAM users in your AWS account, IAM lets you use *identity providers*. Using identity providers, you can manage user identities outside of AWS, and you can give these external user identities permissions to use AWS resources in your account. This is useful if your organization has its own identity system, such as a corporate user directory. It is also useful if you are creating a mobile app or web application that requires access to AWS resources. When you use an identity provider, you don't have to create custom sign-in code or manage your own user identities. Instead, users of your app can sign in using a well-known identity provider—such as Login with Amazon, Facebook, Google, and many others—and you can give that external identity permissions to use AWS resources in your account. Using an identity provider helps you keep your AWS account secure, because you don't have to distribute long-term security credentials, such as IAM user access keys, with your application.

Note
For mobile applications, we recommend that you use Amazon Cognito. You can use this service with the AWS Mobile SDK for iOS and the AWS Mobile SDK for Android and Fire OS to create unique identities for users and authenticate them for secure access to your AWS resources. Amazon Cognito supports the same identity providers as IAM, and also supports unauthenticated (guest) access and lets you migrate user information when a user signs in. Amazon Cognito also provides synchronization APIs for end user profiles so that identity information is preserved as users move between devices. For more information, see the following:

- Amazon Cognito Identity in the *AWS Mobile SDK for iOS Developer Guide*
- Amazon Cognito Identity in the *AWS Mobile SDK for Android Developer Guide*

To use an identity provider, you create an IAM identity provider entity to establish trust between your AWS account and the external identity provider. IAM supports identity providers that are compatible with SAML 2.0 (Security Assertion Markup Language 2.0) or with OpenID Connect (OIDC). For more information about using an identity provider, see the following topics:

- Using SAML Providers (p. 105)
- Using OpenID Connect Identity Providers (p. 109)

Using SAML Providers

This section discusses *SAML providers*, which are entities in IAM that describe an identity provider (IdP) that supports SAML 2.0 (Security Assertion Markup Language 2.0). You create a SAML provider in IAM if you want to establish trust between AWS and an IdP such as Shibboleth or Active Directory Federation

Services, and you want to federate user identities so that users in your organization can access AWS resources. SAML providers in IAM are used as principals in an IAM trust policy.

For more information about this scenario, see Using Your Organization's Authentication System and SAML to Grant Access to AWS Resources in the *Using Temporary Security Credentials* guide.

You can create and manage a SAML provider in the AWS Management Console or by using the CLI or API calls.

Before you create a SAML provider, you need a SAML metadata document that you get from your identity provider (IdP) and that includes the issuer's name, expiration information, and keys that can be used to validate the SAML authentication response (assertions) that are received from the IdP. You must generate the metadata document using the identity management software that is used as your organization's IdP.

For instructions on how to configure an identity provider to work with AWS, including how to generate the required SAML metadata document, see the following:

- Integrating SAML Solution Providers with AWS (p. 108). This topic provides links to the documentation from different SAML identity providers for how to set up providers to work with AWS.
- How to use Shibboleth for single sign-on to the AWS Management Console. This entry on the AWS Security Blog provides a step-by-step tutorial on how to set up Shibboleth and configure it as an identity provider for AWS.

For more information, see Creating Temporary Security Credentials for SAML Federation in the *Using Temporary Security Credentials* guide.

Topics

Creating and Managing a SAML Provider (AWS Management Console)

To create a SAML provider

1. Sign in to the AWS Management Console and open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, click **Identity Providers** and then click **Create SAML Provider**.
3. Enter a name for the provider and then click **Next Step**.
4. Click the **Choose File** button, upload the SAML metadata document, and then click **Create**. You get this metadata document from your identity provider. For more information about configuring identity providers to work with AWS, see the introduction to this topic.

After you create the SAML provider, you should create an IAM role that references the new provider as a principal in its trust policy. The role also defines the permissions that federated users will have when they access AWS. For more information, see Creating a Role for Third-Party Identity Provider (Federation) (p. 127).

To delete a SAML provider

1. Sign in to the AWS Management Console and open the IAM console at https://console.www.amazonaws.cn/iam/.
2. In the navigation pane, click **Identity Providers**.
3. Select the box next to the identity provider that you want to delete.
4. Click **Delete Providers**.

Creating a Role and Configuring Relying Party Trust

When you're done creating a SAML provider, you create an IAM role that lets your organization's IdP get temporary security credentials for access to AWS, and that determines what federated users from that organization are allowed to do. For details, see Creating a Role for Third-Party Identity Provider (Federation) (p. 127).

Finally, when you've created the role, you configure relying party trust between your IdP and AWS. For more information, see Configuring a Relying Party and Adding Claims (p. 107).

Managing a SAML Provider (CLI and API)

To create and manage a SAML provider, use the following commands.

- Create an identity provider and upload a metadata document

  CLI: `aws iam create-saml-provider`

  API: `CreateSAMLProvider`
- Upload a new metadata document for an IdP

  CLI: `aws iam update-saml-provider`

  API: `UpdateSAMLProvider`
- Get information about a specific provider, such as the ARN, creation date, and expiration

  CLI: `aws iam get-saml-provider`

  API: `GetSAMLProvider`
- List information for all IdPs, such as the ARN, creation date, and expiration

  CLI: `aws iam list-saml-providers`

  API: `ListSAMLProviders`
- Delete an IdP

  CLI: `aws iam delete-saml-provider`

  API: `DeleteSAMLProvider`

Configuring a Relying Party and Adding Claims

When you've finished creating a SAML provider and the role for SAML access in IAM, you configure relying party trust between your IdP and AWS. The exact process for adding relying party trust depends on what IdP you're using; for details, see the documentation for your identity management software.

Many IdPs allow you to specify a URL from which the IdP can read an XML document that contains relying party information and certificates. For AWS, that endpoint is the following:

```
https://signin.aws.amazon.com/static/saml-metadata.xml
```

If you can't specify a URL directly, in your IdP, download the XML document from the preceding URL, and then import it into your IdP software.

You also need to create appropriate claim rules in your IdP with AWS as a relying party. These rules map information about users and groups in your organization into appropriate SAML attributes. This lets you make sure that SAML authentication responses (assertions) from your IdP contain the necessary attributes that AWS uses in IAM policies to check permissions for federated users. For details, see Configure Assertions for the SAML Authentication Response in the *Using Temporary Security Credentials* guide.

Additional Resources

For more information, see the following resources:

- Creating Temporary Security Credentials for SAML Federation. This topic discusses SAML-specific keys in IAM policies and how to use them to restrict permissions for SAML-federated users.
- Integrating SAML Solution Providers with AWS (p. 108). This topic provides links to documentation provided by third-party organizations about how to integrate identity solutions with AWS.

Integrating SAML Solution Providers with AWS

The following section provides links to information about how to configure identity provider solutions for use with AWS Security Assertion Markup Language (SAML) 2.0 federation.

Solution	More information
Auth0	AWS Integration in Auth0 – This page on the Auth0 documentation website describes how to set up single sign-on (SSO) with the AWS Management Console and includes a JavaScript example.
Bitium	Configuring SAML for AWS – This article on the Bitium support site explains how to set up Amazon Web Services (AWS) with SAML single sign-on (SSO) using Bitium.
Identacor	Configuring Single Sign-On (SAML) for Amazon Web Services – This article on the Identacor website describes how to set up and enable single sign-on (SSO) for Amazon Web Services.
Microsoft Active Directory Federation Services (ADFS)	Enabling Federation to AWS using Windows Active Directory, ADFS, and SAML 2.0 – This post on the AWS Security Blog shows how to set up ADFS on an Amazon EC2 instance and enable SAML federation with AWS. PowerShell Automation to Give AWS Console Access – This post on Sivaprasad Padisetty's blog describes how to use Windows Powershell commands to automate the process of setting up Active Directory and ADFS, and enabling SAML federation with AWS.

Solution	More information
Okta	AWS SAML integration with Okta – This article on the Okta support site describes how to configure Okta for use with AWS.
OneLogin	Configuring SAML for Amazon Web Services (AWS) – This article on the OneLogin support site describes how to set up single sign-on (SSO) functionality between OneLogin and AWS.
Ping Identity	PingFederate Amazon Web Services Connector – From this page you can download a PDF file that describes how to configure a PingFederate server to enable single sign-on (SSO) for user accounts with Amazon Web Services (AWS). PingOne: Configuring an SSO connection to AWS – This page describes how to configure a single sign-on (SSO) connection from PingOne to Amazon Web Services.
RadiantLogic	Radiant Logic Technology Partners – Under Amazon Web Services, click the **Learn More** link to view a tutorial that shows how to configure RadiantOne and SAML to enable single sign-on (SSO) with AWS.
Salesforce.com	How to configure SSO from Salesforce to Amazon Web Services – This how-to article on the Salesforce.com developer site describes how to set up an identity provider (IdP) in Salesforce and configure AWS as a service provider (SP).
SecureAuth	AWS (Amazon Web Services) - SecureAuth SAML SSO – This article describes how to set up SAML integration with AWS for SecureAuth Appliance 6.x.
Shibboleth	How to use Shibboleth for single sign-on to the AWS Management Console – This entry on the AWS Security Blog provides a step-by-step tutorial on how to set up Shibboleth and configure it as an identity provider for AWS.

For a list of companies who offer single sign-on (SSO) capabilities, see the IAM Partners page on the AWS website.

Using OpenID Connect Identity Providers

OIDC providers are entities in IAM that describe an identity provider that supports the OpenID Connect (OIDC) standard. You use an OIDC provider when you want to establish trust between an OIDC-compatible identity provider—such as Google, Salesforce, and many others—and your AWS account. This is useful if you are creating a mobile app or web application that requires access to AWS resources, but you don't want to create custom sign-in code or manage your own user identities. For more information about this scenario, see Creating Temporary Security Credentials for Mobile Apps Using Identity Providers in the *Using Temporary Security Credentials* guide.

Before you create an OIDC provider in IAM, you must register your application with the OIDC identity provider to receive a *client ID*. The client ID (also known as *audience*) is a unique identifier for your app that is issued to you when you register your app with the identity provider. For more information about obtaining a client ID, see the documentation for your identity provider.

You can create and manage an OIDC provider using the AWS Management Console, the AWS Command Line Interface, or the IAM API.

Topics

Creating and Managing an OIDC Provider (AWS Management Console)

To create an OIDC provider

1. Open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, click **Identity Providers**, then click **Create Provider**.
3. In the **Provider Type** menu, choose **OpenID Connect**.
4. In the **Provider URL** field, enter the URL of the identity provider. The URL is case-sensitive and must begin with `https://`. Within your AWS account, each OIDC provider must use a unique URL.
5. In the **Audience** field, enter the client ID of the application that you registered with the identity provider, and that will make requests to AWS. If you have additional client IDs (also known as *audiences*) for this identity provider, you can enter them later on the provider detail page. Click **Next Step**.
6. Use the **Thumbprint** to verify the server certificate of your identity provider. To learn how, see Obtaining the Thumbprint for an OpenID Connect Provider (p. 112). Click **Create**.
7. In the confirmation message at the top of the screen, click **Do this now** to go to the **Roles** tab to create a role for this OIDC provider. For more information about creating a role for an OIDC provider, see Creating a Role for Third-Party Identity Provider (Federation) (p. 127). OIDC providers must have a role in order to access your AWS account. To skip this step and create the role later, click **Close**.

To add or remove a thumbprint or client ID (also known as audience) for an OIDC provider

1. Open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, click **Identity Providers**, then click the name of the identity provider that you want to update.
3. To add a thumbprint or audience, click **Add a Thumbprint** or **Add an Audience**. To remove a thumbprint or audience, click **Remove** next to the item that you want to remove.

   Note
   An OIDC provider can have a maximum of 5 thumbprints and 100 audiences. An OIDC provider must have at least 1 thumbprint. When you configure an OIDC provider using the AWS Management Console, the OIDC provider must have at least 1 audience.

   When you are done, click **Save Changes**.

To delete an OIDC provider

1. Open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, click **Identity Providers**.
3. Select the box next to the identity provider that you want to delete.
4. Click **Delete Providers**.

Creating and Managing an OIDC Provider (AWS Command Line Interface)

To create and manage an OIDC provider using the AWS Command Line Interface, use the following commands.

- To create a new OIDC provider:

  `aws iam create-open-id-connect-provider`
- To add a new client ID to an existing OIDC provider:

  `aws iam add-client-id-to-open-id-connect-provider`
- To remove a client ID from an existing OIDC provider:

  `aws iam remove-client-id-from-open-id-connect-provider`
- To update the list of server certificate thumbprints for an existing OIDC provider:

  `aws iam update-open-id-connect-provider-thumbprint`
- To get a list of all the OIDC providers in your AWS account:

  `aws iam list-open-id-connect-providers`
- To get detailed information about an OIDC provider:

  `aws iam get-open-id-connect-provider`
- To delete an OIDC provider:

  `aws iam delete-open-id-connect-provider`

Creating and Managing an OIDC Provider (IAM API)

To create and manage an OIDC provider using the IAM API, use the following API actions.

- To create a new OIDC provider:

  `CreateOpenIDConnectProvider`
- To add a new audience (also known as a client ID) to an existing OIDC provider:

  `AddClientIDToOpenIDConnectProvider`
- To remove a client ID from an existing OIDC provider:

  `RemoveClientIDFromOpenIDConnectProvider`
- To update the list of server certificate thumbprints for an existing OIDC provider:

  `UpdateOpenIDConnectProviderThumbprint`

- To get a list of all the OIDC providers in your AWS account:

  `ListOpenIDConnectProviders`
- To get detailed information about an OIDC provider:

  `GetOpenIDConnectProvider`
- To delete an OIDC provider:

  `DeleteOpenIDConnectProvider`

Obtaining the Thumbprint for an OpenID Connect Provider

When you create an OpenID Connect (OIDC) provider in IAM, you must supply a thumbprint for the OIDC provider. The thumbprint is a signature for the unique server certificate that is used by the OIDC provider. When you create an OIDC provider in IAM, you are trusting identities from that provider with access to your AWS account. By supplying the OIDC provider's thumbprint, you assert to AWS that you wish to trust a particular OIDC provider with this access.

When you create an OIDC provider using the AWS Command Line Interface or using the IAM API, you must obtain the thumbprint manually and supply it to AWS. When you create an OIDC provider using the IAM console, the console will attempt to fetch the thumbprint for you. We recommend that you obtain the thumbprint for your OIDC provider manually and verify that the thumbprint obtained by the IAM console matches the one you expect for your OIDC provider.

You use a web browser and the OpenSSL command-line tool to obtain the thumbprint for an OIDC provider. For more information, see the following sections.

Prerequisites

Verifying the OIDC provider thumbprint requires the OpenSSL command-line tool. You use this tool to download the OIDC provider's certificate chain and produce a thumbprint of the final certificate in the certificate chain. If you need to install and configure OpenSSL, follow the instructions at Install OpenSSL (p. 114) and Configure OpenSSL (p. 114).

Obtaining the Certificate Thumbprint

To obtain the thumbprint for an OIDC provider

1. Start with the OIDC provider's URL (for example, `https://server.example.com`), then add `/.well-known/openid-configuration` to form the URL for the OIDC provider's configuration document, like the following:

   ```
   https://server.example.com/.well-known/openid-configuration
   ```

   Open this URL using a web browser, replacing *`server.example.com`* with your OIDC provider's server name.
2. In the document displayed in your web browser, find `"jwks_uri"`. (Use your web browser's *Find* feature to locate this text on the page.) Immediately following the text `"jwks_uri"` you will find a colon (:) followed by a URL. Copy the fully-qualified domain name of the URL—do not include the `https://` or any path that comes after the top-level domain.
3. Using the OpenSSL command-line tool, execute the following command. Replace *`keys.example.com`* with the domain name you obtained in the previous step.

```
openssl s_client -showcerts -connect keys.example.com:443
```

4. In your command window, scroll up until you see a certificate similar to the following example. If you see more than one certificate, find the last certificate that is displayed (at the bottom of the command output).

```


-----BEGIN CERTIFICATE-----
MIICiTCCAfICCQD6m7oRw0uXOjANBgkqhkiG9w0BAQUFADCBiDELMAkGA1UEBhMC
VVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBbWF6
b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMxHzAd
BgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5jb20wHhcNMTEwNDI1MjA0NTIxWhcN
MTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAwDgYD
VQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC0lBTSBDb25z
b2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEWEG5vb25lQGFt
YXpvbi5jb20wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ
21uUSfwfEvySWtC2XADZ4nB+BLYgVIk60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9T
rDHudUZg3qX4waLG5M43q7Wgc/MbQITxOUSQv7c7ugFFDzQGBzZswY6786m86gpE
Ibb3OhjZnzcvQAaRHhdlQWIMm2nrAgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4
nUhVVxYUntneD9+h8Mg9q6q+auNKyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0Fkb
FFBjvSfpJIlJ00zbhNYS5f6GuoEDmFJl0ZxBHjJnyp378OD8uTs7fLvjx79LjSTb
NYiytVbZPQUQ5Yaxu2jXnimvw3rrszlaEXAMPLE=
-----END CERTIFICATE-----
```

Copy the certificate (including the `-----BEGIN CERTIFICATE-----` and `-----END CERTIFICATE-----` lines) and paste it into a text file, then save the file using the file name **`certificate.crt`**.

5. Using the OpenSSL command-line tool, execute the following command.

```
openssl x509 -in certificate.crt -fingerprint -noout
```

Your command window will display the certificate thumbprint, which looks similar to the following example:

```
SHA1 Fingerprint=99:0F:41:93:97:2F:2B:EC:F1:2D:DE:DA:52:37:F9:C9:52:F2:0D:9E
```

Remove the colon characters (:) from this string to produce the final thumbprint, like this:

```
990F4193972F2BECF12DDEDA5237F9C952F20D9E
```

6. If you are creating the OIDC provider using the AWS CLI or the IAM API, supply this thumbprint when creating the provider.

If you are creating the OIDC provider using the IAM console, compare this thumbprint to the thumbprint that you see in the console on the **Verify Provider Information** page when creating an OIDC provider.

Important
If the thumbprint you obtained does not match the one you see in the console, you should not create the OIDC provider in IAM. Instead, you should wait a while and then try again to create the OIDC provider, ensuring that the thumbprints match before you create the provider. If the thumbprints still do not match after a second attempt, we recommend that you contact us using the IAM Forum.

Install OpenSSL

If you don't already have OpenSSL installed, follow the instructions in this section.

To install OpenSSL on Linux or Unix

1. Go to OpenSSL: Source, Tarballs (http://www.openssl.org/source/).
2. Download the latest source and build the package.

To install OpenSSL on Windows

1. Go to OpenSSL: Binary Distributions (http://www.openssl.org/related/binaries.html).
2. Click **OpenSSL for Windows**.

   A new page displays with links to the Windows downloads.
3. If it is not already installed on your system, select the **Microsoft Visual C++ 2008 Redistributables** link appropriate for your environment and click **Download**. Follow the instructions provided by the **Microsoft Visual C++ 2008 Redistributable Setup Wizard**.

   Note
   If you are not sure if the Microsoft Visual C++ 2008 Redistributables is already installed on your system, you can try installing OpenSSL first. The OpenSSL installer will display an error if the Microsoft Visual C++ 2008 Redistributables is not yet installed. Make sure you install the architecture (32-bit or 64-bit) that matches the version of OpenSSL that you install.
4. After you have installed the Microsoft Visual C++ 2008 Redistributables, select the appropriate version of the OpenSSL binaries for your environment and save the file locally. Launch the **OpenSSL Setup Wizard**.
5. Follow the instructions described in the **OpenSSL Setup Wizard**.

Configure OpenSSL

Before you use OpenSSL commands, you must configure the operating system so that it has information about the location where OpenSSL is installed.

To configure OpenSSL on Linux or Unix

1. At the command line, set the `OpenSSL_HOME` variable to the location of the OpenSSL installation:

```
export OpenSSL_HOME=path_to_your_OpenSSL_installation
```

2. Set the path to include the OpenSSL installation:

```
export PATH=$PATH:$OpenSSL_HOME/bin
```

   Note
   Any changes you make to environment variables using the `export` command are valid only for the current session. You can make persistent changes to the environment variables by setting them using your shell configuration file. For more information, see the documentation for your operating system.

To configure OpenSSL on Windows

1. Open a **Command Prompt** window.
2. Set the `OpenSSL_HOME` variable to the location of the OpenSSL installation:

```
set OpenSSL_HOME=path_to_your_OpenSSL_installation
```

3. Set the `OpenSSL_CONF` variable to the location of the configuration file in your OpenSSL installation:

```
set OpenSSL_CONF=path_to_your_OpenSSL_installation\bin\openssl.cfg
```

4. Set the path to include the OpenSSL installation:

```
set Path=%Path%;%OpenSSL_HOME%\bin
```

Note
Any changes you make to Windows environment variables in a **Command Prompt** window are valid only for the current command line session. You can make persistent changes to the environment variables by setting them as system properties. The exact procedures depends on what version of Windows you're using. (For example, in Windows 7, open **Control Panel** > **System and Security** > **System**. Then choose **Advanced system settings** > **Advanced** tab > **Environment Variables**.) For more information, see the Windows documentation.

IAM Roles (Delegation and Federation)

Sometimes you want to *delegate* access to users, applications, or services that don't normally have access to your AWS resources. For example, you might want to grant a user in your AWS account access to resources they don't usually have, or grant a user in one AWS account access to resources in another account. Or you might want to allow a mobile app to use AWS resources, but not want to store AWS keys within the app (where they can be difficult to rotate and where users can potentially extract them). Sometimes you want to give users who already have identities outside of AWS, such as through your corporate directory, access to AWS resources, that is, create *federated identities*. Or, you might want to grant access to your account to a third party, for example, so that they can perform an audit on your resources.

For these scenarios, you can delegate access to AWS resources using an *IAM role*. This section introduces roles and the different ways you can use them, when and how to choose among approaches, and how to create, manage, switch to (or assume), and delete roles.

Topics

Roles - Terms and Concepts

Before we dig into scenarios for using roles, here are some basic terms that are useful to know before you start.

Role

A role is essentially a set of permissions that grant access to actions and resources in AWS. These permissions are attached to the role, not to an IAM user or group. Roles can be used by an IAM user in the same AWS account as the role or a different account, an AWS service such as Amazon EC2, or an identity provider (IdP) that is compatible with SAML 2.0 or OpenID Connect.

Policy

You define the permissions for a role in an IAM policy, which is a JSON document written in the IAM Policy Language.

When you create the role, you create two separate policies for it: The *trust policy*, which specifies who is allowed to assume the role (the trusted entity, or *principal* - see next term), and the *permissions policy*, which defines what actions and resources the principal is allowed to use.

Principal

A principal is an entity in AWS which can perform action and access resources. A principal can be an AWS account (the "root" user), an IAM user, group, or role. You can use a policy to grant permissions to a principal by attaching the policy to the principal, or by referencing the principal in a policy that is attached to a resource. If you reference an AWS account, it generally means any principal defined in that account.

Cross-Account Access: Roles Versus Resource-Based Policies

Granting access to resources in one account to a trusted principal in a different account is often referred to as *cross-account access*. Roles are the primary way to grant cross-account access. However, some AWS services enable you to attach a policy directly to a resource (instead of using a role as a proxy). These polices are called resource-based policies, and you can use them to grant another principals in another AWS account access to the resource. The following services support resource-based policies for the specified resources: Amazon S3 buckets, Amazon SNS topics, and Amazon SQS queues. For more information, see How Roles Differ from Resource-Based Policies (p. 160).

There are two ways to use roles, in the IAM console, or programmatically in the AWS CLI or API:

- IAM users in your account can *switch to* a role while they are working in the IAM console to temporarily use the permissions of the role. The user gives up their original permissions and takes on the permissions assigned to the role. When the user exits the role, their original permissions are reinstated.
- An application or AWS service (like Amazon EC2) can *assume* a role by requesting temporary security credentials with which to make programmatic requests to AWS. You might use roles programmatically, for example, so that you don't have to share or maintain long-term security credentials (for example, by creating an IAM user) for each entity that requires access to a resource.

Use Cases: Roles for Users, Applications, and Services

The simplest way to use roles is to grant your IAM users permissions to switch to roles that you create within your own or another AWS account. They can switch roles easily using the IAM console to use permissions that you don't ordinarily want them to have, and then exit the role to surrender those permissions. This can help prevent accidental access to or modification of sensitive resources.

For more complex uses of roles, such as granting access to applications and services, or federated users, you can call the `AssumeRole` API function. The function returns a set of temporary credentials that the application can use in subsequent function calls. Actions attempted with the temporary credentials have only the permissions granted by the role. An application doesn't have to "exit" the role the way a user in the console does, rather the code simply stops using the temporary credentials and resumes making calls with the original credentials.

Federated users sign in using credentials provided by an identity provider (IdP). AWS then provides the IdP with temporary credentials associated with a role to pass on to the user for including in subsequent AWS resource requests. Those credentials provide the permissions granted to the assigned role.

This section discusses the following scenarios:

- Switch to a role as an IAM user in one AWS account to access resources in another account that you own
- Provide access for AWS services to AWS resources
- Provide access for externally authenticated users (identity federation)
- Provide access to third parties

Switching to a role as an IAM user between accounts you own

You can grant your IAM users permission to switch roles within your AWS account or to roles defined in other AWS accounts you own.

> **Note**
> If the "other" account to which you want to grant access to your resources is owned by a different organization not in your control, see the section Providing access to third parties (p. 121) later in this topic.

For example, you might have Amazon EC2 instances that are critical to your organization. Instead of directly granting your users permission to terminate the instances, you can create a role with those privileges and allow administrators to switch to the role when they need to terminate an instance. This adds the following layers of protection to the instances:

- You must explicitly grant your users permission to assume the role.
- Your users must actively switch to the role using the AWS console.
- You can add multi-factor authentication (MFA) protection to the role so that only users who sign in with an MFA device can assume the role.

We recommend using this approach to enforce the principle of least access, that is, restricting the use of elevated permissions to only those times when they are needed for specific tasks. With roles you can help prevent accidental changes to sensitive environments, especially if you combine them with auditing to help ensure that roles are only used when needed.

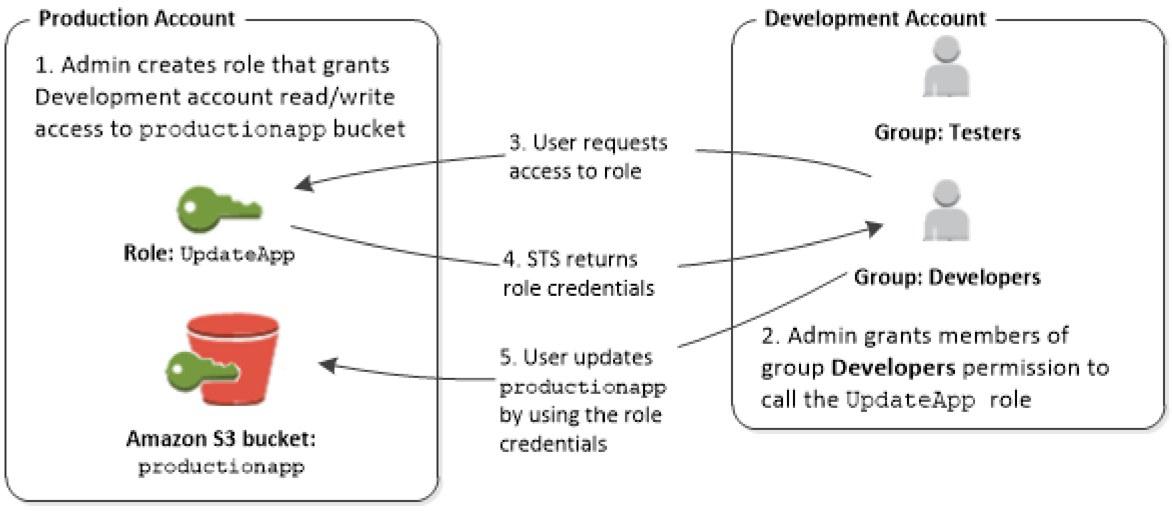
To create a role for this purpose, specify the account IDs that contain the users as principals in the role's trust policy. You can then grant specific users in those accounts permissions to use the role.

A user in one account can switch to a role in the same or a different account. While using the role, the user can perform the actions and access the resources permitted by the role, but their original user permissions are inactive. When the user exits the role, the original user permissions take effect again.

For example, you might have multiple AWS accounts to isolate a development environment from a production environment. Users from one account might occasionally need to access resources in the other account, such as when you are promoting an update from the development environment to the production environment. Although users who work in both accounts could have separate identities (and passwords) in each account, managing credentials for multiple accounts makes identity management difficult.

Suppose that your organization has two AWS accounts: Production and Development. The Production account is where live applications are managed, and the Development account is a sandbox where developers and testers can freely test applications. In the following figure, all users are managed in the Development account, but some developers require limited access to the Production account. The Development account has two groups: Testers and Developers, and each group has its own policy.

Use a role to delegate permissions to a user in a different account


Production Account
1. Admin creates role that grants Development account read/write access to productionapp bucket
Role: UpdateApp
Amazon S3 bucket: productionapp
3. User requests access to role
4. STS returns role credentials
5. User updates productionapp by using the role credentials
Development Account
Group: Testers
Group: Developers
2. Admin grants members of group Developers permission to call the UpdateApp role

1. An administrator in the Production account uses IAM to create the `UpdateAPP` role. In the role, the administrator defines a trust policy that specifies the Development account as a `Principal`, meaning that authorized users from the Development account can use the `UpdateAPP` role. The administrator also defines a permissions policy for the role that specifies users of the role have read and write permissions for the `productionapp` bucket.

   The administrator then shares the ARN of the role with anyone who needs to assume the role. The role ARN might look like `arn:aws:iam::123456789012:role/UpdateAPP`, where the role is named `UpdateAPP` and the role was created in account number 123456789012.

   Note
   The administrator can optionally configure the role so that users who assume the role must first be authenticated using multi-factor authentication (MFA). For more information, see Configuring MFA-Protected API Access (p. 80).
2. In the Development account, an administrator grants members of the Developer group permission to switch to the role. This is done by granting the Developer group permission to call the AWS Security Token Service (AWS STS) `AssumeRole` API for the `UpdateAPP` role. Any IAM user that belongs to the Developer group in the Development account can now switch to the `UpdateAPP` role in the Production account. Other users who are not in the Developer group do not have permission to switch to the role, and therefore cannot access the Amazon S3 bucket in the Production account.
3. The user requests switches to the role:
   - *AWS console*: The user selects the **Switch Role** option in the Identity menu. The user specifies the account ID (or alias) and role name. The user can alternatively click on a link sent in email by the administrator. The link takes the user to the Switch Role page with the details already filled in.
   - *API/CLI*: A user in the Developer group of the Development account calls the `AssumeRole` function to obtain the `UpdateAPP` role credentials. The user specifies the ARN of the `UpdateAPP` role as part of the call. If a user in the Testers group makes the same request, the request fails because Testers do not have permission to call `AssumeRole` by using the `UpdateAPP` role ARN.
4. Temporary credentials are returned:
   - *AWS console*: AWS STS verifies the request against the role's trust policy to ensure that the request is from a trusted entity (which it is: the Development account). After verification, AWS STS returns temporary security credentials to the AWS console.

  - *API/CLI*: AWS STS verifies the request against the role's trust policy to ensure that the request is from a trusted entity (which it is: the Development account). After verification, AWS STS returns temporary security credentials to the application.

5. The temporary credentials are used to access the AWS resource:
   - *AWS console*: The AWS console uses the temporary credentials on the behalf of the user on all subsequent console actions, in this case, to read and write to the `productionapp` bucket. The console cannot access any other resource in the Production account. When the user exits the role, the user's permissions revert to the original permissions held before switching to the role.
   - *API/CLI*: The application uses the temporary security credentials to update the `productionapp` bucket. With the temporary security credentials, the application can only read and write to the `productionapp` bucket and cannot access any other resource in the Production account. The application does not have to exit the role, but instead stops using the temporary credentials and reverts to using the original credentials in subsequent API calls.

Providing access to an AWS service

Some AWS services use roles to control what the service can access. Each service is different in terms of how it uses roles and how the roles are assigned to the service. When an AWS service, such as an Amazon EC2 instance that runs your application, needs to make requests to an AWS resource, such as an Amazon S3 bucket or a DynamoDB table, the service must have security credentials that grant permissions to the resource. Do not embed or pass IAM user credentials directly into an instance, because distributing and rotating long-term credentials to multiple instances is challenging to manage and a potential security risk. A better strategy is to create a role that is assigned to the Amazon EC2 instance when it is launched. AWS automatically provides temporary security credentials for the Amazon EC2 instance to use on behalf of its applications. For more information, see Using IAM Roles to Delegate Permissions to Applications that Run on Amazon EC2 (p. 140).

See the documentation for a specific service to see if it uses roles and how to assign a role for the service to use.

Providing access to externally authenticated users (identity federation)

Your users might already have identities outside of AWS, such as in your corporate directory. If those users need to work with AWS resources (or work with applications that access those resources) then those users also need AWS security credentials. You can use an IAM role to specify permissions for users whose identity is federated from your organization or a third-party identity provider (IdP). For more information, see Using Your Company's Own Authentication System to Grant Access to AWS Resources and Creating Temporary Security Credentials to Enable Access for Federated Users in the *Using Temporary Security Credentials* guide.

- If your organization supports SAML 2.0 (Security Assertion Markup Language 2.0), you can create trust between your organization as an identity provider (IdP) and AWS as the service provider. You can then use SAML to provide your users with federated single-sign on (SSO) to the AWS Management Console or federated access to call AWS APIs. For more information, see Creating Temporary Security Credentials for SAML Federation in the *Using Temporary Security Credentials* guide.
- If you create a mobile or web-based app that accesses AWS resources, the app needs security credentials in order to make programmatic requests to AWS. But you shouldn't embed long-term security credentials in the app, because they are accessible to the app's users and can be difficult to rotate. Instead, you can let users sign in to your app using Login with Amazon, Amazon Cognito, Facebook, or Google, and then use their authentication information to assume a role and get temporary security credentials. We recommend that you use Amazon Cognito with the AWS SDKs for mobile development. For more information, see the following:
  - Amazon Cognito Overview in the *AWS SDK for Android Developer Guide*

- Amazon Cognito Overview in the *AWS SDK for iOS Developer Guide*

Providing access to third parties

When third parties require access to your organization's AWS resources, you can use roles to delegate access to them. For example, a third party might provide a service for managing your AWS resources. With IAM roles, you can grant these third parties access to your AWS resources without sharing your AWS security credentials. Instead, they can assume a role that you created to access your AWS resources.

Third parties must provide you the following information for you to create a role that they can assume:

- The third party's AWS account ID that contains the IAM users that can use your role. You specify their AWS account ID as the principal when you define the trust policy for the role.
- An external ID that the third party can use to associate you with your role. You specify the ID that is provided by the third party as a condition when you define the trust policy for the role. For more information about the external ID, see About the External ID.
- The permissions that the third party requires to work with your AWS resources. You specify these permissions when defining the role's permission policy. This policy defines what actions they can take and what resources they can access.

After you create the role, you must share the role's Amazon Resource Name (ARN) with the third party. They require your role's ARN in order to use the role.

Important
When you grant third parties access to your AWS resources, they can access any resource that you give them permissions to and their use of your resources is billed to you. Ensure that you limit their use of your resources appropriately.

Creating IAM Roles

You can create a role by using the AWS Management Console, the AWS CLI, or the IAM API.

If you use the AWS Management Console, you use a wizard that guides you through the steps for creating a role. The wizard has slightly different steps depending on whether you're creating a role for an AWS service, for an AWS account, or for a federated user. Therefore, we provide separate procedures for each scenario.

Topics

Creating a Role to Delegate Permissions to an AWS Service

You create a role for an AWS service when you want to grant permissions to a service like Amazon EC2, AWS Data Pipeline, Amazon Elastic Transcoder, or AWS OpsWorks. These services can access AWS resources, so you create a role to determine what the service is allowed to do with those resources. In many scenarios, you can select an AWS managed policy that contains predefined permissions. However, if you have requirements that are not covered by an AWS managed policy, you can create a custom policy or start with a copy of an AWS managed policy.

For information about how roles enable you to delegate permissions, see Roles - Terms and Concepts (p. 116).

Topics

Creating a role for a service using the console

To create a role for an AWS service using the AWS Console

1. In the navigation pane of the console, click **Roles**, and then click **Create New Role**.
2. In the **Role name** box, enter a role name that can help you identify the purpose of this role. Role names must be unique within your AWS account. After you enter the name, click **Next Step** at the bottom of the page.

   Because various entities might reference the role, you cannot edit the name of the role after it has been created.
3. Expand the **AWS Service Roles** section, and then select the service that you will allow to assume this role.
4. Select the managed policy that enables the permissions you want the service to have.
5. Click **Next Step** to review the role and then click **Create Role**.

Creating a role for a service using the AWS Command Line Interface

Creating a role using the CLI involves multiple steps. When you use the console to create a role, many of the steps are done for you, but with the CLI you must explicitly perform each step yourself. You must create the policy, and assign a permissions policy to the role. If the service you are working with is Amazon EC2 then you must also create an instance profile and add the role to it.

To create a role for an AWS service using the AWS CLI, use the following commands:

- Create a role: `aws iam create-role`
- Attach a permissions policy to the role: `aws iam put-role-policy`

If you are going to use the role with Amazon EC2 or another AWS service that uses Amazon EC2, you must store the role in an instance profile. An instance profile is a container for a role. An instance profile can contain only one role. If you create the role using the AWS Management Console, the instance profile is created for you. For more information about instance profiles, see Instance Profiles (p. 144). For information about how to launch an Amazon EC2 instance with a role, see Using IAM roles with Amazon EC2 instances in the *Amazon EC2 User Guide for Linux Instances*.

- Create an instance profile: `aws iam create-instance-profile`
- Add the role to the instance profile: `aws iam add-role-to-instance-profile`

The following example shows all four steps. The example assumes that you are running on a client computer running Windows, and have already configured your command line interface with your account credentials and region. For more information, see Configuring the AWS Command Line Interface.

The sample trust policy referenced in the first command contains the following JSON code to enable the Amazon EC2 service to assume the role.

```
{

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

  "Statement": {

    "Effect": "Allow",

    "Principal": {"Service": "ec2.amazonaws.com"},

    "Action": "sts:AssumeRole"

  }

}
```

The sample permissions policy referenced in the second command allows the role to perform only the `ListBucket` action on an S3 bucket named `example_bucket`.

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

The AWS CLI commands to run for this example are the following:

```
# Create the role and attach the trust policy that enables EC2 to assume this
role.

aws iam create-role --role-name Test-Role-for-EC2 --assume-role-policy-document
 file://C:\policies\trustpolicyforec2.json



# Attach the permissions policy to the role to specify what it is allowed to
do.

aws iam put-role-policy --role-name Test-Role-for-EC2 --policy-name Permissions-
Policy-For-Ec2 --policy-document file://c:\policies\permissionspolicyforec2.json



# Create the instance profile required by EC2 to contain the role

aws iam create-instance-profile --instance-profile-name EC2-ListBucket-S3
```

```
# Finally, add the role to the instance profile

aws iam add-role-to-instance-profile --instance-profile-name EC2-ListBucket-S3
 --role-name Test-Role-for-EC2
```

When you launch the EC2 instance, specify the instance profile name in the **Configure Instance Details** page if you use the AWS console, or the `--iam-instance-profile` parameter if you use the `aws ec2 run-instances` CLI command.

Creating a role for a service using the AWS API

To create a role in code using the API, use the following commands.

- Create a role: `CreateRole`

  For the role's trust policy, you can specify a file location.
- Attach a permissions policy to the role: `PutRolePolicy`

If you are going to use the role with Amazon EC2 or another AWS service that uses Amazon EC2, you must store the role in an instance profile. An instance profile is a container for a role. Each instance profile can contain only one role. (If you create the role using the AWS Management Console, the instance profile is created for you.) For more information about instance profiles, see Instance Profiles (p. 144). For information about how to launch an Amazon EC2 instance with a role, see Using IAM roles with Amazon EC2 instances in the *Amazon EC2 User Guide for Linux Instances*.

- Create an instance profile: `CreateInstanceProfile`
- Add the role to the instance profile: `AddRoleToInstanceProfile`

Creating a Role to Delegate Permissions to an IAM User

You can create a role that delegates access for IAM users to AWS resources. The user and the resources being accessed can be in the same or different accounts.

For information about how roles enable you to delegate permissions, see Roles - Terms and Concepts (p. 116).

Creating a role for delegating user permissions using the AWS Management Console

To create a role that an IAM users can switch to using the AWS Management Console

1. In the navigation pane of the console, click **Roles** and then click **Create New Role**.
2. In the **Role name** box, enter a role name that can help you identify the purpose of this role. Role names must be unique within your AWS account. After you enter the name, click **Next Step** at the bottom of the page.

   Role names have character limitations. The number of roles in an AWS account, and the policy size for policies attached to roles are also limited. For more information, see Limitations on IAM

Entities (p. 13). Because various entities might reference the role, you cannot edit the name of the role after it has been created.

3. Click **Roles for Cross-Account Access**, and then select the type of role that you want to create:

   - Select **Provide access between AWS accounts you own** if you are the administrator of both the user account and the resource account, or both accounts belong to the same company. This is also the option to select when the users, role, and resource to be accessed are all in the same account.
   - Select **Allows IAM users from a 3rd party AWS account to access this account** if you are the administrator of the account that owns the resource and you want to grant permissions to users from an account that you do not control. This option enables you to specify an "external ID" to provide additional control over who can use the role to access your resources. For more information, see About the External ID.

     Important
     Note that selecting this option enables access to the role only through the AWS CLI and API. You cannot switch roles at the AWS console to a role that has an external ID condition in its trust policy.

4. Specify the AWS account ID that you want to grant access to your resources.

   Any IAM user from the trusted AWS account can assume this role if that user has a policy that grants permission for the AWS STS `AssumeRole` action and that specifies your role as the resource.
5. If you selected **Allows IAM users from a 3rd party AWS account to access this account**, enter the external ID provided by the administrator of the third party account. This automatically adds a condition to the trust policy that allows the user to assume the role only if the request includes the correct `sts:ExternalID`.
6. If you want to restrict the role to users who provide multi-factor authentication, select the **Require MFA** option. This adds an MFA condition to the role's trust policy. A user who wants to assume the role must provide a temporary one-time password (TOTP) from the configured device. Users without MFA authentication cannot assume the role.
7. Click **Next Step**.
8. Set the permissions for the role to specify what actions can be done on specific resources (similar to setting permissions for IAM groups). You specify permissions by selecting a policy. For information about managing permissions by using policies, see http://docs.amazonaws.cn/IAM/latest/UserGuide/PoliciesOverview.html.

   The permissions that you specify are available to any entity that uses the role. By default, roles have no permissions.

   Select the box next to the policy that assigns the permissions that you want the users to have, and then click **Attach Policy**.
9. Click **Next Step** to review the role. Note the link provided for you to give to users who can use the role. When the user clicks this link, the user is taken directly to the Switch Role page with the **Account ID** and **Role Name** already filled in. The user can optionally set a **Display Name** and can select a **Display Color**. When the user clicks **Switch Role**, the user immediately begins operating with the new permissions.
10. Click **Create Role** to complete the creation of the role.

    Important
    Remember that this is only the first half of the configuration required. You must also enable individual users in the trusted account with permissions to switch to the role. For more information about this step, see Granting a User Permissions to Switch Roles (p. 136).

Creating a role for that users can switch to using the AWS Command Line Interface

Creating a role using the CLI involves multiple steps. When you use the console to create a role, many of the steps are done for you, but with the CLI you must explicitly perform each step yourself. You must create the policy, and assign a permissions policy to the role.

To create a role for cross-account access using the AWS CLI, use the following commands:

- Create a role: `aws iam create-role`
- Attach a permissions policy to the role: `aws iam put-role-policy`

The following example shows all both steps in a simple environment. The example assumes that you are running on a client computer running Windows, and have already configured your command line interface with your account credentials and region. For more information, see Configuring the AWS Command Line Interface.

The sample trust policy referenced in the first command contains the following JSON code to enable users in the account 123456789012 to assume the role, but only if the user provides MFA authentication.

```
{

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

  "Statement": {

    "Effect": "Allow",

    "Principal": { "AWS": "arn:aws:iam::123456789012:root" },

    "Action": "sts:AssumeRole",

    "Condition": { "Bool": { "aws:MultiFactorAuthPresent": "true" } }

  }

}
```

The sample permissions policy referenced in the second command allows the user who assumes the role to perform only the `ListBucket` action on an S3 bucket named `example_bucket`.

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

The commands to run are the following:

```
# Create the role and attach the trust policy that enables users in an account
 to switch to the role.
```

```
aws iam create-role --role-name Test-UserAccess-Role --assume-role-policy-docu
ment file://C:\policies\trustpolicyforacct123456789012.json



# Attach the permissions policy to the role to specify what it is allowed to
do.

aws iam put-role-policy --role-name Test-UserAccess-Role --policy-name Perms-
Policy-For-Acct-123456789012 --policy-document file://c:\policies\permspolicy
foracct123456789012.json
```

Important
Remember that this is only the first half of the configuration required. You must also enable individual users in the trusted account with permissions to switch to the role. For more information about this step, see Granting a User Permissions to Switch Roles (p. 136).

Creating a role that users can switch to using the AWS API

To create a role in code using the API, use the following commands.

- Create a role: `CreateRole`

  For the role's trust policy, you can specify a file location.
- Attach a permissions policy to the role: `PutRolePolicy`

Important
Remember that this is only the first half of the configuration required. You must also enable individual users in the trusted account with permissions to switch to the role. For more information about this step, see Granting a User Permissions to Switch Roles (p. 136).

For information about using multi-factor authentication from API calls, see Configuring MFA-Protected API Access (p. 80).

Creating a Role for Third-Party Identity Provider (Federation)

By using identity federation, you can provide access to AWS resources for users who sign in using a third-party identity provider (IdP). To configure identity federation, you configure the provider and then you create an IAM role that determines what permissions a federated user will have.

- **Web identity federation**

  With web identity federation, you can provide access to AWS resources for users who have signed in using Login with Amazon, Facebook, Google, or an IdP that is with the Open ID standard. To configure web identity federation, you create and configure the identity provider in the AWS Management Console and then create an IAM role that determines who can assume the role and what permissions the federated user will have.

  Important
  We recommend that you use Amazon Cognito to manage user identities for mobile apps or web applications that need access to AWS resources. Amazon Cognito can create IAM roles for you; when you use Amazon Cognito, you typically need to use the process described in this topic only if you are creating a custom role for advanced scenarios. To learn more about

Amazon Cognito, see Amazon Cognito Identity in the *AWS SDK for Android Developer Guide* and Amazon Cognito Identity in the *AWS SDK for iOS Developer Guide*.

- **Security Assertion Markup Language (SAML) 2.0 federation**

  SAML-based federation lets you provide access to AWS resources for users in an organization that uses SAML 2.0 (Security Assertion Markup Language 2.0) to exchange authentication and authorization information. To configure SAML-based federation in AWS, you create and configure an identity provider in the AWS Management Console and then create an IAM role that identifies the organization that can assume the role and what federated users from that organization are allowed to do.

Topics

Creating a Role for Web Identity (OIDC) Providers Using the AWS Management Console

Before you create a role for web identity federation, you must register your application with the identity provider, who will give you an application ID (also known as a client ID, or audience). For more information, see Creating Temporary Security Credentials for Mobile Apps Using Third-Party Identity Providers in the *Using Temporary Security Credentials* guide.

After getting the required information from the identity provider, you must then create an *identity provider* in IAM. For more information, see Using OpenID Connect Identity Providers (p. 109).

After creating the identity provider, you can then create the role that is assumed by users who are authenticated by IdP.

To create an IAM role for web identity federation using the IAM console

1. Open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, click **Roles** and then click **Create New Role**.
3. In the **Role name** box, enter a role name that can help you identify the purpose of this role. Role names must be unique within your AWS account. Because various entities might reference the role, you cannot edit the name of the role after you create it. After you enter the name, click **Next Step** at the bottom of the page.
4. Select **Roles for Identity Provider Access** and then click the **Select** button for **Grant access to web identity providers**.
5. In the **Identity Provider** list, select the identity provider that you're creating the role for:

   - Select **Amazon Cognito** if you're creating a role for Amazon Cognito.

     Note
     Remember that you must create a role for use with Amazon Cognito only for advanced scenarios, because Amazon Cognito can create roles for you.
     For more information about Amazon Cognito, see Amazon Cognito Identity in the *AWS Mobile SDK for iOS Developer Guide* and Amazon Cognito Identity in the *AWS Mobile SDK for Android Developer Guide*.

   - If you're creating a role for an individual web identity provider, select the name of the provider. Remember that you must create a separate role for each identity provider that you want to support.

6. Enter the client ID that identifies your application, or enter the ID for your Amazon Cognito identity pool. The name of the box for the ID changes depending on which provider you select.

   - If you're creating a role for Amazon Cognito, enter the ID of the identity pool you have created for your Amazon Cognito applications into the **Identity Pool ID** box.
   - If you're creating a role for an individual web identity provider, enter or select the client ID that the provider gave you when you registered your application with the provider.

7. (Optional) Click **Add Conditions** to create additional conditions that must be met before users of your application can use the permissions granted by the role. For example, you can add a condition that grants access to AWS resources only for a specific user ID.
8. Click **Next Step** to review the role's **Trust Policy Document** and then click **Next Step**.
9. Select the managed policy that assigns the permissions that you want the federated users to have, and then click **Attach Policy**.
10. Click **Next Step** to review the role and then click **Create Role**.

Creating a Role for SAML 2.0-Based Federated Access Using the AWS Management Console

Before you create a role for SAML-based federation, you must create a SAML provider in IAM. For more information, see Using SAML Providers (p. 105).

The role-creation wizard in the IAM console provides two paths. One path is for creating a role for single sign-on (SSO) to the AWS Management Console. The other path is for creating a role that can be assumed programmatically. The following procedure describes both paths. The roles created by both paths are similar, but the path for SSO creates a role whose trust policy includes a condition that explicitly ensures that the SAML audience (`aud` attribute) is set to the AWS sign-in endpoint for SAML (https://signin.www.amazonaws.cn/saml).

To create a role for SAML-based federation using the IAM console

1. Make sure you've created a SAML provider in IAM, as described in Using SAML Providers (p. 105).
2. In the navigation pane of the console, click **Roles** and then click **Create New Role**.
3. In the **Role name** box, enter a role name that can help you identify the purpose of this role. Role names must be unique within your AWS account. After you enter the name, click **Next Step** at the bottom of the page.

   Because various entities might reference the role, you cannot edit the name of the role after it has been created.
4. Click **Role for Identity Provider Access**.
5. Select the type of role that you're creating: **Grant Web Single Sign-On (SSO) access to SAML identity providers** or **Grant API access to SAML identity providers**.
6. In the **SAML Provider** list, select the provider that you're creating the role for.
7. If you're creating a role for API access, select an attribute from the **Attribute** list. Then in the **Value** box, enter a value that will be included in the role. This lets you restrict access to the role only to users from the identity provider whose SAML authentication response (assertion) includes the attributes you select. You must specify at least one attribute, which ensures that your role is scoped to a subset of users at your organization.

   If you're creating a role for SAML single sign-on, the `SAML:aud` attribute is automatically added and set to the URL of the AWS SAML endpoint (https://signin.aws.amazon.com/saml).
8. To add more attribute-related conditions to the trust policy, click **Add Conditions**, select the additional condition, specify a value, and then click **Add Condition**.

The list displays a selected set of the most commonly used SAML attributes. IAM supports additional attributes that you can use to create conditions. (For a list of the supported attributes, see "Available Keys for SAML Federation" in the topic IAM Policy Elements Reference (p. 209).) If you need create a condition for a supported SAML attribute that's not displayed in the list, you can edit the policy later in the wizard in order to add that condition.

9. Click **Next Step**. The wizard displays the trust policy for the role in an editable box. The policy includes the condition or conditions based on what you entered.
10. When you've reviewed the policy and finished making any changes, click **Next Step** again.
11. Set the permissions that determine what the federated user will be allowed to do. By default, roles have no permissions. Select the managed policy that assigns the permissions that you want the federated users to have from the list, and then click **Attach Policy**.
12. Click **Next Step** to review the role and then click **Create Role**.

After you've created the role, you configure relying party trust between your IdP and AWS. For more information, see Configuring a Relying Party and Adding Claims (p. 107).

Creating a Role for Federated Access Using the AWS Command Line Interface

Before you create the role, you must create the identity provider entity in IAM.

For more information about creating OIDC providers, see Using OpenID Connect Identity Providers (p. 109).

For more information about creating SAML providers, see Using SAML Providers (p. 105).

Creating a role for the supported identity providers is identical—the difference is in the contents of the trust policy associated with the role.

- For web identity providers, we recommend that you manage identities using Amazon Cognito. In that case, the trust policy for the role must include a `Statement` similar to the following:

```
{

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

  "Statement": {

    "Effect": "Allow",

    "Principal": {"Federated": "cognito-identity.amazonaws.com"},

    "Action": "sts:AssumeRoleWithWebIdentity",

    "Condition": {

      "StringEquals": {"cognito-identity.amazonaws.com:aud": "us-east-
1:12345678-abcd-abcd-abcd-123456"},

      "ForAnyValue:StringLike": {"cognito-identity.amazonaws.com:amr": "unau
thenticated"}

    }

  }
}
```

Replace `us-east-1:12345678-abcd-abcd-abcd-123456` with the actual identity pool ID that was assigned to you by Amazon Cognito.

- For SAML 2.0 providers, the policy must include a `Statement` element similar to the following:

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "sts:AssumeRoleWithSAML",
   "Principal": {"Federated": "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:saml-
provider/PROVIDER-NAME"},
    "Condition": {"StringEquals": {"SAML:aud": "ht
tps://signin.aws.amazon.com/saml"}}
  }
}
```

Replace the principal ARN with the actual ARN for the SAML provider that you created in IAM. It will have your own account ID and the actual provider name.

Creating a role using the AWS CLI involves multiple steps. When you use the console to create a role, many of the steps are done for you, but with the CLI you must explicitly perform each step yourself. You must create the trust policy first, then create the role, and then assign a permissions policy to the role.

To create a role using the AWS CLI, use the following commands:

- To create a role: `aws iam create-role`
- To attach a permissions policy to the role: `aws iam attach-role-policy` or `aws iam put-role-policy`

The following example shows all of the steps in a simple environment. The example assumes that you are running the AWS CLI on a computer running Windows, and have already configured the CLI with your credentials. For more information, see Configuring the AWS Command Line Interface.

The sample trust policy referenced in the first command allows users authenticated by Amazon Cognito and belonging to the Amazon Cognito identity pool `us-east-1:12345678-abcd-abcd-abcd-123456` to assume the role.

```
{

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

  "Statement": {

    "Effect": "Allow",

    "Principal": {"Federated": "cognito-identity.amazonaws.com"},

    "Action": "sts:AssumeRoleWithWebIdentity",

    "Condition": {
     "StringEquals": {"cognito-identity.amazonaws.com:aud": "us-east-1:12345678-
abcd-abcd-abcd-123456"},

      "ForAnyValue:StringLike": {"cognito-identity.amazonaws.com:amr": "unau
thenticated"}

    }

  }

}
```

The sample permissions policy referenced in the second command allows the user who assumes the role to perform only the `ListBucket` action on an Amazon S3 bucket named `example_bucket`.

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

The commands to run are the following:

```
# Create the role and attach the trust policy that enables users in an account
 to assume the role.

aws iam create-role --role-name Test-CrossAcct-Role --assume-role-policy-document
 file://C:\policies\trustpolicyforcognitofederation.json



# Attach the permissions policy to the role to specify what it is allowed to
do.

aws iam put-role-policy --role-name Test-CrossAcct-Role --policy-name Perms-
Policy-For-CognitoFederation --policy-document file://c:\policies\permspolicy
forcognitofederation.json
```

Creating a Role for Federated Access Using the IAM API

To create a role for identity federation using the IAM API, use the following API calls. For details about the policy files that are supplied by parameters in the API calls, see the the section called "Creating a Role for Federated Access Using the AWS Command Line Interface" (p. 130).

- Create a role: `CreateRole`
- Attach a permissions policy to the role: `AttachRolePolicy` or `PutRolePolicy`

Examples of Policies for Delegating Access

Topics

Example: Using roles to delegate access to another AWS account's resources

For a complete walkthrough that shows how to use IAM roles to grant users in one account access to AWS resources that are in another account, see Walkthrough: Delegating Access Across AWS Accounts Using IAM Roles (p. 151).

Example: Using a resource-based policy to delegate access to an Amazon S3 bucket in another account

In this example, Account A uses a resource-based policy (an Amazon S3 bucket policy) to grant Account B full access to Account A's Amazon S3 bucket. Then Account B creates an IAM user policy to delegate that access to Account A's bucket to one of the users in Account B.

The Amazon S3 bucket policy in Account A might look like the following policy. In this example, Account A's Amazon S3 bucket is named *mybucket*, and Account B's account number is 111122223333.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Sid": "AccountBAccess1",
    "Effect": "Allow",
    "Principal": {"AWS": "111122223333"},
    "Action": "s3:*",
    "Resource": [
      "arn:aws:s3:::mybucket",
      "arn:aws:s3:::mybucket/*"
    ]
  }
}
```

Alternatively, Account A can use Amazon S3 Access Control Lists (ACLs) to grant Account B access to an Amazon S3 bucket or a single object within a bucket. In that case, the only thing that changes is how Account A grants access to Account B. Account B still uses a policy to delegate access to a user in Account B, as described in the second part of this example. For more information about controlling access on Amazon S3 buckets and objects, go to Access Control in the *Amazon Simple Storage Service Developer Guide*.

The next example shows the IAM user (or group) policy that Account B might create to delegate read access to a user in Account B. In this policy, the `Action` element is explicitly defined to allow only `List` actions, and the `Resource` element of this policy matches the `Resource` for the bucket policy implemented by Account A.

Account B implements this policy by using IAM to attach it to the appropriate user (or group) in Account B.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "s3:List*",
    "Resource": [
      "arn:aws:s3:::mybucket",
      "arn:aws:s3:::mybucket/*"
    ]
  }
}
```

Example: Using a resource-based policy to delegate access to an Amazon SQS queue in another account

In the following example, Account A has an Amazon SQS queue that uses a resource-based policy attached to the queue to grant queue access to Account B. Then, Account B uses an IAM user policy to delegate access to a user in Account B.

The following example queue policy gives Account B permission to perform the `SendMessage` and `ReceiveMessage` actions on Account A's queue named *queue1*, but only between noon and 3:00 p.m. on November 30, 2014. Account B's account number is 1111-2222-3333. Account A uses Amazon SQS to implement this policy.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Principal": {"AWS": "111122223333"},
    "Action": [
      "sqs:SendMessage",
      "sqs:ReceiveMessage"
    ],
    "Resource": ["arn:aws:sqs:*:123456789012:queue1"],
    "Condition": {
      "DateGreaterThan": {"aws:CurrentTime": "2014-11-30T12:00Z"},
      "DateLessThan": {"aws:CurrentTime": "2014-11-30T15:00Z"}
    }
  }
}
```

Account B's policy for delegating access to a user in Account B might look like the following example. Account B uses IAM to attach this policy to a user (or group).

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "sqs:*",
    "Resource": "arn:aws:sqs:*:123456789012:queue1"
  }
}
```

In the preceding IAM user policy example, Account B uses a wildcard to grant its user access to all Amazon SQS actions on Account A's queue. But, because Account B can delegate access only to the extent that Account B has been granted access, Account B's user can access the queue only between noon and 3:00 p.m. on November 30, 2014, and the user can only perform the `SendMessage` and `ReceiveMessage` actions, as defined in Account A's Amazon SQS queue policy.

Example: Cannot delegate access when the account is denied access

By default, other AWS accounts and their users cannot access your AWS account resources. But, when you use a policy to explicitly deny an AWS account access to your resources, the deny propagates to the users under that account regardless of whether the users have existing policies granting them access. This means that an AWS account cannot delegate access to another account's resources if the other account has explicitly denied access to the user's parent account.

For example, Account A writes a bucket policy on Account A's Amazon S3 bucket that explicitly denies Account B access to Account A's bucket. But Account B writes an IAM user policy that grants a user in Account B access to Account A's bucket. The explicit deny applied to Account A's Amazon S3 bucket propagates to the users in Account B and overrides the IAM user policy granting access to the user in Account B. (For detailed information how permissions are evaluated, see IAM Policy Evaluation Logic (p. 243).)

Account A's bucket policy might look like the following policy. In this example, Account A's Amazon S3 bucket is named *mybucket*, and Account B's account number is 1111-2222-3333. Account A uses Amazon S3 to implement this policy.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Sid": "AccountBDeny",
    "Effect": "Deny",
    "Principal": {"AWS": "111122223333"},
    "Action": "s3:*",
    "Resource": "arn:aws:s3:::mybucket/*"
  }
}
```

Account B implements the following IAM user policy by using IAM to attach it to the a user in Account B.

```
{
   "Version": "2012-10-17",
   "Statement":{
      "Effect":"Allow",
      "Action":"s3:*",
      "Resource":"arn:aws:s3:::mybucket/*"
   }
}
```

Account A's bucket policy explicitly denies Account B access to *mybucket*. Because you only delegate a subset of permissions that have been granted to you, Account B's IAM user policy granting the user in Account B access to Account A's bucket has no effect. The user in Account B cannot access Account A's bucket.

Using IAM Roles

Before an IAM user, application, or service can use a role that you created, you must grant the user permissions to switch to the role. You can use any policy attached to the user or one of the user's groups to grant the necessary permissions. This section describes how to grant users permission to use a role, and then how the user can switch to a role using the AWS Management Console, the AWS Command Line Interface (AWS CLI) and the `AssumeRole` API.

Topics

Related Information

- For details about using roles to delegate permissions to applications that run on Amazon EC2 instances, see Using IAM Roles to Delegate Permissions to Applications that Run on Amazon EC2 (p. 140).
- For a detailed walkthrough of cross-account access using roles, see Walkthrough: Delegating Access Across AWS Accounts Using IAM Roles (p. 151).

Granting a User Permissions to Switch Roles

When you create a role for cross-account access, you establish trust from the account that owns the role and the resources to the account that contains the user by specifying the users' account number as the Principle in the role's trust policy. That enables potentially any user in the trusted account to assume the role. To complete the picture, however, the administrator of the trusted account must also enable specific users by granting them permissions to switch to the role.

To grant a user permission to switch to a role, you create a new policy for the user or edit an existing policy to add the required element. You can then send the user a link that takes the user to the Switch Role page with all the details already filled in. Alternatively, you can provide the user with the Account ID number or account alias that contains the role, and the role name. The user then goes to the Switch Role page and enters the details manually.

Creating or Editing the Policy

A policy that grants a user permission to use a role must include a statement with the `Allow` effect on the `sts:AssumeRole` action and the appropriate `Resource`, as shown in the following example.

In the example, the `Resource` element specifies the ARN of the role that the user is allowed to assume. Note that if `Resource` is set to `*`, the user can assume any role in any account as long as that role trusts the user's account.

As a best practice, we recommend that you follow the principle of least privilege and limit the user to only the role they need access to by specifying the complete ARN for the role.

The following example shows a policy that lets the user assume roles in only one account and additionally specifies by wildcard (*) that the user can only switch to a role in that account if the name begins with "Test".

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "sts:AssumeRole",
    "Resource": "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:role/Test*"
  }
}
```

Note
The permissions that are granted to the user by the role replace, and do not add to, the permissions granted to the user. For example, if the user's permissions allow working with Amazon EC2 instances, but the role's permissions policy does not grant those permissions, then while using the role the user cannot work with Amazon EC2 instances in the console, and temporary credentials obtained via `AssumeRole` do not work with Amazon EC2 instances programmatically.

Providing Information to the User

After you create a role and granted your user permissions to switch to it, you must provide the user with the Account ID number or account alias that contains the role, and the role name. You can make things even easier for your users by sending them a link that is preconfigured with the account ID and role name. You can see the role link on the final page of the **Create Role** wizard, or in the **Role Summary** page for any cross-account enabled role.

Note
If you create the role by using the AWS CLI or the AWS API then you are able to create the role with a *path* in addition to a name. If you do so then you must provide the complete path and role name to your users to enter on the **Switch Role** page of the AWS Management Console. For example: `division_abc/subdivision_efg/roleToDoXYZ`.

You can also create the link by using the following format, substituting your Account ID or alias and the role name for the two parameters in the request:

```
https://signin.aws.amazon.com/switchrole?account=YourAccountIDorAliasHere&roleName=YourRoleNameHere
```

We recommend that you direct your users to the topic Switching to a Role in the AWS Management Console (p. 137) to step them through the process.

Note
Switching roles can be audited for security purposes by using AWS CloudTrail. If CloudTrail is turned on for the account, IAM logs actions in CloudTrail that are performed using the temporary security credentials. For more information, see CloudTrail Event Reference in the AWS CloudTrail User Guide.

Switching to a Role in the AWS Management Console

A *role* specifies a set of permissions that you can use to access AWS resources that you need. In that sense, it is similar to a user in AWS Identity and Access Management (IAM). When you sign in as a user, you get a specific set of permissions. However, you don't sign in to a role, but once signed in you can switch to a role. This temporarily sets aside your original user permissions and instead gives you the permissions assigned to the role. The role can be in your own account or any other AWS account.

This helps you to apply security best practices that implement the principle of least privilege and state that you should only have the permissions to perform the task at hand. Only take on 'elevated' permissions while doing a task that requires them, and then give them up when the task is done.

For example, a developer might have full access to the Test application environment, but not normally have access to the Production environment. That developer can switch roles to temporarily get deployment permissions in the Production environment to push out a new version of the application. When done, the developer can switch back to the normal user and give up those extra permissions. This helps ensure that the developer doesn't accidentally make changes to the Production environment.

Important
The permissions of your IAM user and any roles you switch to are not cumulative. Only one set of permissions is active at a time: the one role that is currently active, or the IAM user you used to sign in.

This section describes how to use the IAM console to switch roles, and is provided for users who have already been granted access to switch to a role by an account administrator.

- If your administrator provides you with a link, click the link and then skip to Step 4 (p. 138) in the following procedure. The link takes you to the page and fills in the account ID or alias and the role name for you.
- If your administrator provides you with the account and role information, then you can switch roles by following the steps in the following procedure.

To switch to a role

1. In the IAM console, click your user name in the navigation bar in the upper right corner. It typically looks like this: ***`UserName`***@***`Account ID number or alias`***.
2. In the Identity drop-down menu that appears, select **Switch Role**. If this is the first time you've ever selected this option, then a page appears with more information. After reading it, click **Switch Role** again.
3. On the **Switch Role** page, enter the Account ID number or the account alias and the name of the role that was provided by your administrator.

   Note
   If your administrator created the role with a path, such as `division_abc/subdivision_efg/roleToDoXYZ`, then you must enter that complete path and name in the **Role name** text box. If you enter only the role name, it is not found and switch role fails.
4. You can optionally enter the text that you want to appear in the navigation bar in place of your user name when this role is active. A name is suggested, based on the Account and Role information, but you can change it to whatever has meaning for you. You can also select the color that is used to highlight the display name. The name and color are to help remind you when this role is active that you have different permissions then when the role is not active. For example, a role that gives you access to the Test environment you might specify a **Display Name** of "Test Environment" and select the green **Display Color**. For the role that gives you access to Production, you might specify a **Display Name** of "PRODUCTION Environment" and select red as the **Display Color**.
5. Click **Switch Role**. The display name and color replace your user name in the navigation bar and you can start using the permissions granted to you by the role.

The last several roles you've used are included on the Identity drop-down menu. The next time you need to switch roles to one of them you can simply click the role that you want to use. You only need to enter the account and role information if the role is not displayed on the Identity menu.

To stop using a role

1. In the IAM console, select your role's **Display Name** on the right side of the navigation bar.

2. Select **Back to *`UserName`***. The role is deactivated and the permissions associated with your IAM user and groups are reapplied.

Switching IAM Roles by Using the API

An application receives permissions to carry out required tasks and interact with AWS resources by assuming a role. To assume a role, an application calls the AWS STS `AssumeRole` API and passes the ARN of the role to use. The `AssumeRole` API returns a set of temporary security credentials that you can use in subsequent AWS API calls to access resources in the account that owns the role. The temporary credentials have whatever permissions are defined in the role's permissions policy. The call to `AssumeRole` can optionally pass a supplemental policy that can be used to further restrict the permissions of the temporary security credentials returned by the `AssumeRole` API.

Note
The use of roles can be audited for security purposes by enabling AWS CloudTrail in the account. The call to `AssumeRole` must include a role session name between 2 and 32 characters long that can include letters, numbers, and the `=,.@-` characters. The role session name is used when CloudTrail logs actions that are performed using the temporary security credentials. For more information, see CloudTrail Event Reference in the *AWS CloudTrail User Guide*.

The following example in Python (using the AWS SDK for Python (Boto)) shows how to call `AssumeRole` and how to use the temporary security credentials returned by `AssumeRole` to list all Amazon S3 buckets in the account that owns the role.

```
import boto
from boto.sts import STSConnection
from boto.s3.connection import S3Connection

# The calls to AWS STS AssumeRole must be signed using the access key ID
# and secret access key of an IAM user or using existing temporary
# credentials. (You cannot call AssumeRole using the access key for an
# account.) The credentials can be in environment variables or in a
# configuration file and will be discovered automatically by the
# STSConnection() function. For more information, see the Python SDK
# documentation: http://boto.readthedocs.org/en/latest/boto_config_tut.html

sts_connection = STSConnection()
assumedRoleObject = sts_connection.assume_role(
    role_arn="arn:aws:iam::account-of-role-to-assume:role/name-of-role",
    role_session_name="AssumeRoleSession1"
)

# Use the temporary credentials returned by AssumeRole to call Amazon S3
# and list all buckets in the account that owns the role (the trusting account)
s3_connection = S3Connection(
    aws_access_key_id=assumedRoleObject.credentials.access_key,
    aws_secret_access_key=assumedRoleObject.credentials.secret_key,
    security_token=assumedRoleObject.credentials.session_token
)
bucketList = s3_connection.get_all_buckets()
for bucket in bucketList:
 print bucket.name
```

Using IAM Roles to Delegate Permissions to Applications that Run on Amazon EC2

Using roles to grant permissions to applications running in Amazon EC2 instances requires a bit of extra configuration. An application running in an Amazon EC2 instance is abstracted from AWS by the virtualized operating system. Because of this extra separation, an additional step is needed to assign an AWS role and its associated permissions to an Amazon EC2 instance and therefore to its applications. This extra step is the creation of an instance profile attached to the instance that contains the role. The instance profile then can provide the role's credentials to an application. Those credentials are used in API calls made by the application to access resources and limit access to only those resources defined by the role. Note that only one role can be assigned to an Amazon EC2 at a time, and all applications on the instance share the same role and permissions.

For example, imagine that you are an AWS administrator. Developers in your organization have applications that run on Amazon EC2 instances. These applications require access to other AWS resources—for example, to read and write to Amazon S3 buckets.

Applications that run on an Amazon EC2 instance must include AWS credentials when making AWS API requests. We refer to this as 'signing' an API request. You could have developers store AWS credentials directly within the Amazon EC2 instance, allowing applications to use those credentials to sign requests, but developers would then have to manage the credentials. For example, they would have to securely pass the credentials to each instance and update each Amazon EC2 instance when it's time to rotate the credentials. That's a lot of additional work.

Instead, you can use an IAM role to manage credentials for applications that run on an Amazon EC2 instance. When you use roles, you don't have to distribute AWS credentials to Amazon EC2 instances. Instead, the role supplies temporary permissions that applications can use when they make calls to other AWS resources. When a developer launches an Amazon EC2 instance, they specify the role you create to associate with the instance. Applications that run on the instance then use the role-supplied temporary credentials to sign API requests.

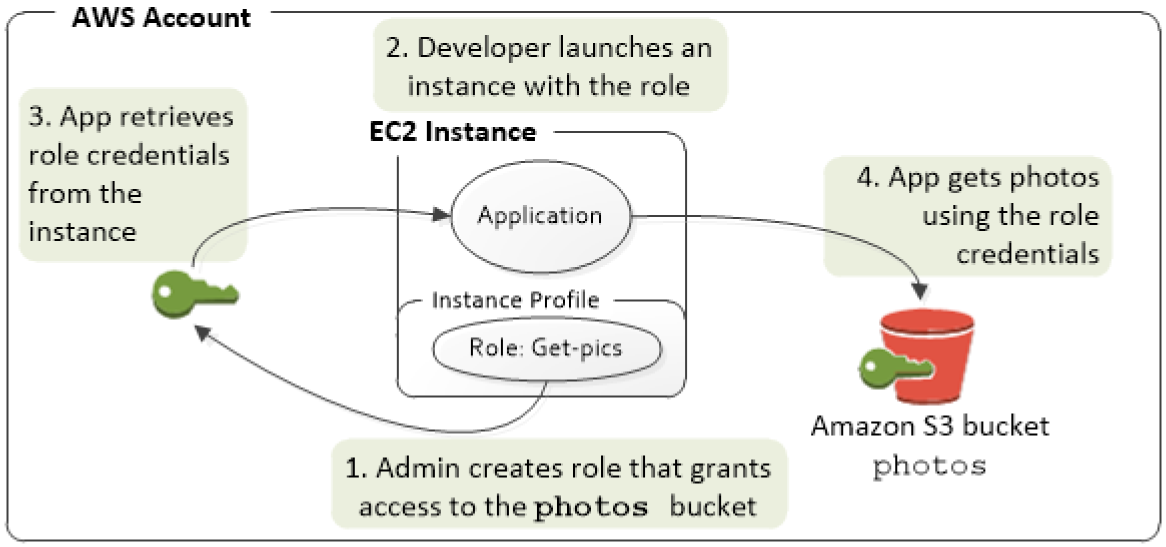
Using roles in this way has several benefits. Because role credentials are temporary and rotated automatically, you don't have to manage credentials, and you don't have to worry about long-term security risks. In addition, if you use a single role for multiple instances, you can make a change to that one role and the change is propagated automatically to all the instances, simplifying credential management.

Important
A role is assigned to an Amazon EC2 instance when you launch it, and cannot be assigned to an instance that is already running. If you need to add a role to an instance that is already running, you can create an image of the instance, then launch a new instance using the image you created and with the desired role assigned.

How Do Roles for Amazon EC2 Instances Work?

In the following figure, a developer is running an application on an Amazon EC2 instance that requires access to the Amazon S3 bucket named `photos`. An administrator creates the `Get-pics` role. The role includes policies that grant read permissions for the bucket and that allow the developer to launch the role with an Amazon EC2 instance. When the application runs on the instance, it can access the photos bucket by using the role's temporary credentials. The administrator doesn't have to grant the developer permission to access the photos bucket, and the developer never has to share or manage credentials.

Application on an Amazon EC2 instance accessing an AWS resource


AWS Account
2. Developer launches an instance with the role
3. App retrieves role credentials from the instance
EC2 Instance
Application
Instance Profile
Role: Get-pics
4. App gets photos using the role credentials
Amazon S3 bucket
photos
1. Admin creates role that grants access to the photos bucket

1. The administrator uses IAM to create the **`Get-pics`** role. In the role, the administrator uses a policy that specifies that only Amazon EC2 instances can assume the role and that specifies read-only permissions for the `photos` bucket.
2. A developer launches an Amazon EC2 instance and assigns the `Get-pics` role to that instance.

   Note
   If you use the IAM console, the instance profile is managed for you and is mostly transparent to you. However, if you use the CLI or API to create and manage the role and instance, then you must create the instance profile and assign the role to it as separate steps. Then, when launching the instance, you must specify the instance profile name instead of the role name.

3. When the application runs, it uses the AWS API to retrieve credentials from its Amazon EC2 instance. These are temporary security credentials that represent the role and are valid for a limited period of time.

   With some AWS SDKs, the developer can use a provider that manages the temporary security credentials transparently. (The documentation for individual AWS SDKs describes the features supported by that SDK for managing credentials.) Alternatively, the application can get the temporary credentials directly from the instance metadata of the Amazon EC2 instance— credentials and related values are available from the `iam/security-credentials/`*`role-name`* category (in this case, `iam/security-credentials/Get-pics`) of the metadata. If the application gets the credentials from the instance metadata, it can cache the credentials.
4. Using the retrieved credentials, the application accesses the photo bucket. Because of the policy attached to the **`Get-pics`** role, the application has read-only permissions.

   The temporary security credentials that are available in the instance are automatically rotated before they expire so that a valid set is always available. The application just needs to make sure that it gets a new set of credentials from the instance metadata before the current ones expire. If the AWS SDK is managing credentials, the application doesn't need to include additional logic to refresh the credentials. However, if the application gets temporary security credentials from the instance metadata and has cached them, it should get a refreshed set of credentials every hour, or at least 15 minutes before the current set expires. The expiration time is included in the information that's returned in the `iam/security-credentials/`*`role-name`* category.

Permissions Required for Using Roles with Amazon EC2

To launch an instance with a role, the developer must have permission to launch Amazon EC2 instances and permission to pass IAM roles.

The following sample policy allows users to use the AWS Management Console to launch an instance with a role. The policy allows a user to pass any role and to perform all Amazon EC2 actions by specifying an asterisk (*****). The `ListInstanceProfiles` action allows users to view all the roles that are available on the AWS account.

Example Policy that grants a user permission to launch an instance with any role by using the Amazon EC2 console

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": [
      "iam:PassRole",
      "iam:ListInstanceProfiles",
      "ec2:*"
    ],
    "Resource": "*"
  }]
}
```

Restricting Which Roles Can Be Passed to Amazon EC2 Instances (Using PassRole)

You can use the `PassRole` permission to restrict which role a user can pass to an Amazon EC2 instance when the user is launching the instance. This helps prevent the user from running applications that have more permissions than the user has been granted—that is, from being able to obtain elevated privileges. For example, imagine that user Alice has permissions only to launch Amazon EC2 instances and to work with Amazon S3 buckets, but the role she passes to an Amazon EC2 instance has permissions to work with IAM and DynamoDB. In that case, Alice might be able to launch the instance, log into it, get temporary security credentials, and then perform IAM or DynamoDB actions that she's not authorized for.

To restrict which roles a user can pass to an Amazon EC2 instance, you create a policy that allows the `PassRole` action and attach the policy to the user (or to an IAM group that the user belongs to) who will launch EC2 instances. In the `Resource` element of the policy, you list the role or roles that the user is allowed to pass to EC2 instances. When the user launches an instance and associates a role with it, EC2 checks that the user is allowed to pass that role. Of course, you should also ensure that the role that you're allowing the user to pass does not include more permissions than the user is supposed to have.)

Note

`PassRole` is not an API in the same way that `RunInstances` or `ListInstanceProfiles` is. Instead, it's a permission that Amazon EC2 checks when an instance is launched and when a role is added to an instance profile.

Example policy that grants a user permission to launch an Amazon EC2 instance with a specific role

The following sample policy allows users to use the Amazon EC2 API to launch an instance with a role. The `Resource` element specifies the Amazon Resource Name (ARN) of a role. By specifying the ARN, the policy grants the user the permission only to pass the `Get-pics` role. If the user tries to specify a different role when launching an instance, the action fails.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "ec2:RunInstances",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "iam:PassRole",
      "Resource": "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:role/Get-pics"
    }
  ]
}
```

How Do I Get Started?

To understand how roles work with Amazon EC2 instances, you need to use the IAM console to create a role, launch an Amazon EC2 instance that uses that role, and then examine the running instance. You can see how the role credentials are made available to an instance via the instance metadata, and you can see how an application that runs in an instance can use the role. Use the following resources to learn more.

- Getting Started with IAM Roles for EC2 InstancesThe following video shows you how to use an IAM role with an Amazon EC2 instance to control what an app can do when it runs on the instance. The video shows how the app (written using the AWS SDK) can get temporary security credentials via the role.
- SDK walkthroughs. The AWS SDK documentation includes walkthroughs that show an application running in an Amazon EC2 instance that uses role credentials to read an Amazon S3 bucket. Each of the following walkthroughs presents a similar walkthrough using a different programming language:
  - Using IAM Roles for EC2 Instances with the SDK for Java in the *AWS SDK for Java Developer Guide*
  - Using IAM Roles for EC2 Instances with the SDK for .NET in the *AWS SDK for .NET Developer Guide*
  - Using IAM Roles for EC2 Instances with the SDK for Ruby in the *AWS SDK for Ruby Developer Guide*

  The walkthroughs provide complete step-by-step instructions for creating and compiling the example program, creating the role, launching the instance, connecting to the instance, deploying the example program, and testing it.

Related Information

For more information about creating roles or roles for Amazon EC2 instances, see the following information:

- For more information about using IAM roles with Amazon EC2 instances, go to the *Amazon EC2 User Guide for Linux Instances*.

- To create a role, see Creating IAM Roles (p. 121)
- For more information about using temporary security credentials, go to *Using Temporary Security Credentials*.
- If you work with the IAM API or CLI, you must create and manage IAM instance profiles. For more information about instance profiles, see Instance Profiles (p. 144).
- For more information about role credentials in the instance metadata, see IAM security credentials in the Metadata Categories section of Instance Metadata.

Instance Profiles

An instance profile is a container for an IAM role and enable you to pass role information to an Amazon EC2 instance when the instance starts.

Managing Instance Profiles (AWS Management Console)

If you use the AWS Management Console to create a role, the console automatically creates an instance profile and gives it the same name as the role. When you then use the Amazon EC2 console to launch an instance with an IAM role, you can select a role to associate with the instance. In the console, the list that's displayed is actually a list of instance profile names.

Managing Instance Profiles (AWS CLI and API)

If you manage your roles by using the AWS CLI, you create roles and instance profiles as separate actions. You can give the roles and instance profiles different names, so you have to know the names of your instance profiles as well as the names of roles they contain so that you can choose the correct instance profile when you launch an Amazon EC2 instance.

Note
An instance profile can contain only one role. However, a role can be included in multiple instance profiles.

You can use the following commands to work with instance profiles in an AWS account.

- Create an instance profile

  CLI command: `aws iam create-instance-profile`

  API command: `CreateInstanceProfile`
- Add a role to an instance profile

  CLI command: `aws iam add-role-to-instance-profile`

  API command: `AddRoleToInstanceProfile`
- List instance profiles

  CLI commands: `aws iam list-instance-profiles`, `aws iam list-instance-profiles-for-role`

  API command: `ListInstanceProfiles`, `ListInstanceProfilesForRole`
- Get information about an instance profile

  CLI command: `aws iam get-instance-profile`

  API command: `GetInstanceProfile`
- Remove a role from an instance profile

  CLI command: `aws iam remove-role-from-instance-profile`

API command: `RemoveRoleFromInstanceProfile`
- Delete an instance profile

  CLI command: `aws iam delete-instance-profile`

  API command: `DeleteInstanceProfile`

Managing IAM Roles

In addition to creating roles and using them to access resources, the roles in your account need to be managed. You can change a role by modifying the policies associated with the role, changing who can access the role and the permissions the role grants to users. You can also delete roles that are no longer needed. You can manage your roles by using the AWS Management Console, the AWS CLI, and the API.

Topics

Modifying a Role

You can change who can access a role and the permissions associated with the role.

- You change the accounts and services that can use a role by modifying the trust policy attached to the role. (You change which users in the trusted account are delegated permissions to use the role by modifying their user or group policies.)
- You change the permissions allowed by the role by modifying the permissions policy attached to the role.

Both policies attached to a role are JSON files. You can make the changes by using the AWS Management Console, AWS CLI, or API. Any changes to the role are propagated in near real time.

Topics

Modify a Role (AWS Management Console)

To change which trusted accounts or services can access the role (AWS Management Console)

1. In the navigation pane of the IAM console, click **Roles**.

   The console displays the roles for your account.
2. Click the name of the role that you want to modify, and then open the **Trust Relationship** section in the details pane.
3. Click **Edit Trust Relationship**.
4. Edit the trust policy as needed. You can add additional principals by adding them to the list. You can only specify the account at this point, not a specific user. Remember that JSON arrays are surrounded

by [ ] brackets and separated by commas. As an example, the following JSON snippet shows how to reference two AWS accounts in the `Principal` element:

```
"Principal": {
  "AWS": [
    "arn:aws:iam::111122223333:root",
    "arn:aws:iam::444455556666:root"
  ]
},
```

If your role can be used by one or more AWS services rather than accounts, then the policy might contain an element similar to the following:

```
"Principal": {
  "Service": [
    "opsworks.amazonaws.com",
    "ec2.amazonaws.com"
  ]
},
```

5. When you are done editing, click **Update Trust Policy** to save your changes. For more information about policy structure and syntax, see AWS IAM Policy Reference (p. 208)

To change which users in a trusted account are delegated permission to use the role (AWS Management Console)

1. Begin by choosing to add the permissions directly to a user or to a group. In the navigation pane of the IAM console, select **Users** or **Groups.**
2. Click the name of the user or group to which you want to grant access, and then open an existing policy by clicking **Manage Policy** or open a new policy to attach to the user or group **Attach User Policy** or **Attach Another Policy**.
3. In the policy editor, add a new `Statement` element that specifies the following:

```
{
  "Effect": "Allow",
  "Action": "sts:AssumeRole",
  "Resource": "arn:aws:iam::AccountID-With-Role:role/RoleName"
}
```

   Remember that you can have only one `Statement` keyword. However, a statement can have several elements in an array, each element comma separated and in its own curly braces { }, and all of the elements surrounded by square brackets [ ].
4. When you are done editing, click **Apply Policy** to save your changes. For more information about policy structure and syntax, see AWS IAM Policy Reference (p. 208)

To change the permissions allowed by a role (AWS Management Console)

1. In the navigation pane of the IAM console, click **Roles**.
2. Click the name of the role that you want to modify, and then open the **Permissions** section in the details pane.
3. Find the policy that you want to edit, and in the **Actions** column, click **Manage Policy**.
4. Edit the policy document, and then click **Apply Policy**.

5. To add a new policy to the role (roles can have multiple permissions policies attached to them), click **Attach Role Policy**, select or enter the policy, and then click **Apply Policy**.

Modify a Role (AWS CLI)

To change the accounts that can access the role (AWS CLI);

1. If you don't know the name of the role that you want to modify, list the roles in your account by using the following command:

   - aws iam list-roles.

   A list of roles and information about them is displayed. Note that you use the role name, not the ARN, to refer to roles with the AWS CLI commands. For example, if a role has the following ARN: `arn:aws:iam::123456789012:role/myrole`, you refer to the role as **`myrole`**.
2. (Optional) To view the current trust policy for a role, use the following commands:

   - aws iam list-role-policies
   - aws iam get-role-policy

3. Create a text file with the updated trust policy. You can use any text editor to construct the policy.

   Add a new `Statement` element that specifies the following:

   ```
   {
     "Effect": "Allow",
     "Action": "sts:AssumeRole",
     "Resource": "arn:aws:iam::AccountID-With-Role:role/RoleName"
   }
   ```

   Remember that you can have only one `Statement` keyword, a statement can have several elements in an array, each element comma separated and in its own curly braces { }, and all of the elements surrounded by square brackets [ ].
4. To update the trust policy, use the following command:

   - aws iam update-assume-role-policy.

   Changes to the role are propagated in near real time. You can verify that the role has been updated by using the following command:

   - aws iam get-role-policy

     The return value from this command lists the trust policy.

To change the users in the trusted account who can use the role (AWS CLI)

1. Begin by choosing to add the permissions directly to a user or to a group.
2. If you don't know the name of the user or group that you want to modify, list the users or group in your account by using the following commands:

   - aws iam list-users

aws iam list-groups

3. To view the current policies for a user or group, use the following commands:
   - aws iam list-user-policies aws iam get-user-policy
   - aws iam list-group-policies aws iam get-group-policy

4. Create a text file with the updated permissions policy.

   You can use any text editor to construct the policy. The trust policy must contain the entities that can assume the role. Optionally, you can include any conditions for assuming the role.
5. To update the trust policy, use the following commands:
   - aws iam put-user-policy
   - aws iam put-group-policy

   Changes to the role are propagated in near real time. You can verify that the role has been updated by using the `get-user-policy` or `get-group-policy` commands again.

To change the permissions allowed by a role by using the AWS CLI

1. If you don't know the name of the role that you want to modify, list the roles in your account by using the following command: aws iam list-roles.

   A list of roles and information about them is displayed. Note that you use the role name, not the ARN, to refer to roles with the AWS CLI commands. For example, if a role had the following ARN: `arn:aws:iam::123456789012:role/myrole`, you refer to the role as **`myrole`**.
2. (Optional) To view the current permissions associated with a role, use the aws iam list-role-policies to list the policies for a role. Then use the aws iam get-role-policy command to see details about an individual policy.
3. Create a text file with the updated permissions for the role.

   You can use any text editor to construct the policy. The policy defines the actions that are allowed or denied on a given resource.
4. To update the permissions on the role, use the following command: aws iam put-role-policy.

   Changes to the role are propagated in near real time. You can verify that the role has been updated by using the aws iam get-role-policy command.

Modify a Role (API)

To change who can access the role by using the API

1. If you don't know the name of the role that you want to modify, list the roles in your account by calling **`ListRole`**.

   A response is returned with information for each role. Use the role name, not the ARN, to refer to roles with the API actions.
2. (Optional) To view the current trust policy for a role, call **`GetRole`** with the role name.
3. Call **`UpdateAssumeRolePolicy`** to update the trust policy.

   You must include the policy and role name with the API call. The policy document must contain the entities that can assume the role. Optionally, you can include any conditions for assuming the role.

Changes to the role are propagated in near real time. You can verify that role has been updated by calling `GetRole`.

To change the permissions allowed by a role by using the API

1. If you don't know the name of the role that you want to modify, list the roles in your account by calling `ListRole`.

   A response is returned with information for each role. Use the role name, not the ARN, to refer to roles with the API actions.
2. (Optional) If you want to view the current permissions associated with a role, call `GetRolePolicy`
3. Call `PutRolePolicy` to update the permissions on the role.

   You must include the policy, policy name, and role name with the API call. The policy defines the actions that are allowed or denied on a given resource.

Deleting Roles or Instance Profiles

If you are not using a role, delete the role and its associated permissions so that you don’t have an unused entity that is not actively monitored or maintained.

If a role was associated with an Amazon EC2 instance, then you can also remove the roles from the instance profile and then delete the instance profile.

Caution
Make sure you do not have any Amazon EC2 instances running with the role or instance profile you are about to delete. Deleting a role or instance profile that is associated with a running instance will break any applications running on the instance.

Important
If a role is associated with an Amazon EC2 instance profile then the role can only be deleted using the AWS console if the role and the instance profile have the exact same name. This happens automatically if they are created using the console. If the role was created by using the AWS CLI or programmatically using the API, then the role and the instance profile might have different names, and the console cannot delete them. You must use the AWS CLI or API to first remove the role from the instance profile, and then as separate steps, delete them.

Topics

Delete a Role (AWS Management Console)

When you use the AWS Management Console to delete a role, IAM also automatically deletes the policies associated with the role, and any Amazon EC2 instance profile that contains the role.

To delete a role by using the AWS Management Console

1. In the navigation pane of the IAM Dashboard, click **Roles**.
2. Select the role or roles you want to delete.
3. From the **Role Actions** list, select **Delete Role**.
4. Review your changes, and then click **Yes, Delete**.

Note
You cannot use the console to delete an instance profile, except when you delete it as part of the process of deleting a role as described in the preceding procedure. To delete an instance profile without also deleting the role, you must use the CLI or API. For information about using the CLI or API to remove a role from an instance profile, see the following two sections.

Delete a Role (AWS CLI)

When you use the AWS CLI to delete a role, you must first delete the policies associated with the role. Also, if you want to delete the associated instance profile that contains the role, you must delete it separately.

To delete a role by using the AWS CLI

1. If you don't know the name of the role that you want to delete, list the roles in your account by entering the following command:

   ```
   aws iam list-roles
   ```

   A list of roles with their Amazon Resource Name (ARN) is displayed. Use the role name, not the ARN, to refer to roles with the CLI commands. For example, if a role has the following ARN: `arn:aws:iam::123456789012:role/myrole`, you refer to the role as **`myrole`**.
2. Remove the role from all instance profiles that the role is in.

   a. List all instance profiles that the role is associated with by entering the following command:

   ```
   aws iam list-instance-profiles-for-role --role-name role-name
   ```

   b. To remove the role from an instance profile, enter the following command for each instance profile:

   ```
   aws iam remove-role-from-instance-profile --instance-profile-name in
   stance-profile-name --role-name role-name
   ```

3. Delete all policies that are associated with the role.

   a. List all policies that are in the role by entering the following command:

   ```
   aws iam list-role-policies --role-name role-name
   ```

   b. To delete each policy from the role, enter the following command for each policy:

   ```
   aws iam delete-role-policy --role-name role-name --policy-name policy-
   name
   ```

4. Delete the role by entering the following command:

   ```
   aws iam delete-role --role-name role-name
   ```

5. If you do not plan to reuse the instance profiles that were associated with the role, you can delete them by entering the following command:

   ```
   aws iam delete-instance-profile --instance-profile-name instance-profile-
   name
   ```

Delete a Role (API)

When you use the IAM API to delete a role, you must first delete the policies associated with the role. Also, if you want to delete the associated instance profile that contains the role, you must delete it separately.

To delete a role by using the API

1. Remove the role from all instance profiles that the role is in by calling `RemoveRoleFromInstanceProfile`.

   You must pass the role name and instance profile name. You can list all instance profiles that a role is in by calling `ListInstanceProfilesForRole`.
2. Delete all policies that are associated with the role by calling `DeleteRolePolicy`.

   You must pass the role name and policy name. You can list all policies for a role by calling `ListRolePolicies`.
3. Delete the role by calling `DeleteRole`.
4. If you are not going to reuse the instance profiles that were associated with the role, you can delete them by calling `DeleteInstanceProfile`.

Related Information

For general information about instance profiles, see Instance Profiles (p. 144).

Walkthrough: Delegating Access Across AWS Accounts Using IAM Roles

In this walkthrough, you will learn how to use a role to delegate access to resources that are in different AWS accounts. You'll share resources in one account with users in a different account. By setting up cross-account access in this way, you don't need to create individual IAM users in each account, and users don't have to sign out of one account and sign into another in order to access resources that are in different AWS accounts. After configuring the role, you'll see how to use the role from the AWS Management Console, the AWS CLI, and the API.

Overview

Suppose that your organization has two AWS accounts: Production (account ID number 999999999999) and Development (account ID number 111111111111). The Production account is where live applications are managed, and the Development account is a sandbox where developers and testers can freely test applications. In each account, application information is stored in Amazon S3 buckets.

This walkthrough assumes that you are an administrator of both accounts. You manage IAM users in the Development account, where you have two IAM groups: Developers and Testers. Users in both groups have permissions to work in the Development account and access resources there. David is in the Developers group, and Teresa is in the Testers group.

From time to time, David or another developer must update the live applications in the Production account. In this walkthrough, we use an Amazon S3 bucket called `productionapp` to represent the production resources. You don't want to create additional IAM users for the developers in the Production account because as your organization grows, creating multiple IAM users for each developer would be costly to manage. Instead, you choose to delegate access to developers from the Development account. You establish trust between the two accounts, and allow only developers from the Development account to access the `productionapp` bucket in the Production account.

This walkthrough shows you how you can use an IAM roles to provide access to the `productionapp` bucket in the Production account to the Developers group while preventing access to the Testers group in the same account.

- For more information about IAM users and groups, see IAM Users and Groups (p. 39) .
- For more information about Amazon S3 buckets, see Create a Bucket with Amazon Simple Storage Service in the *Amazon Simple Storage Service Getting Started Guide*.

What You Will Accomplish

By the end of the walkthrough, you will have a role in the Production account (the trusting account) that allows users from the Development account (the trusted account) to access the `productionapp` bucket in the Production account. David can use the role in the AWS Management Console to access the `productionapp` bucket in the Production account. He can also access the bucket by using API calls that are authenticated by temporary credentials provided by the role. Similar attempts by Teresa to use the role fail.

The following list outlines the steps that you will complete:

1. Walkthrough: Cross-Account Delegation Part 1 - Creating a Role (p. 153)

   First, you use the AWS Management Console to establish trust between the Production and Development account by creating an IAM role named *UpdateApp*. When you create the role, you define the Development account as a trusted entity and specify a permissions policy that allows trusted users to update the `productionapp` bucket.
2. Walkthrough: Cross-Account Delegation Part 2 - Granting Users Access to the Role (p. 155)

   In this part of the walkthrough, you modify the IAM group policy so that testers are denied access to the `UpdateAPP` role. Because Testers have PowerUser access in this scenario, we must explicitly deny the ability to use the role.
3. Walkthrough: Cross-Account Delegation Part 3 - Switching to a Role (p. 156)

   Finally, as developer David, you use the `UpdateAPP` role to update the `productionapp` bucket in the Production account. You see how to access the role through the AWS console, the AWS CLI, and the API.

Walkthrough: Cross-Account Delegation Part 1 - Creating a Role

To allow users from one AWS account to access resources in another AWS account, create a role that defines who can access it and what permissions it grants to users that switch to it. For an overview of this walkthrough, see Walkthrough: Delegating Access Across AWS Accounts Using IAM Roles (p. 151).

In this part of the walkthrough, you create the role in the Production account and specify the Development account as a trusted entity. You also limit the role's permissions to only read and write access to the `productionapp` bucket. Anyone who is granted permission to use the role can read and write to the `productionapp` bucket.

Before you can create a role, you need the account ID of the Development AWS account. The account ID is a unique identifier assigned to each AWS account.

To obtain the Development AWS account ID

1. Go to the Amazon Web Services website, hover over **My Account**, click **AWS Management Console**, and then sign in to the AWS Management Console for the Development account.
2. In navigation bar, click **Support**, then **Support Center**. The Account Number is in the upper-right corner immediately below the **Support** menu. The account ID is a 12-digit number. For this scenario, we pretend the Development account ID is 111111111111; however, you should use a valid account ID if you are reconstructing the scenario in your test environment.

To create a role in the Production account that can be used by the Development account

1. Sign in to the AWS Management Console as an administrator of the Production account, and open the IAM console.
2. Before creating the role, prepare the managed policy that defines the permissions that the role requires. You attach this policy to the role in a later step.

   You want to set read and write access to the `productionapp` bucket. Although AWS provides some Amazon S3 managed policies, there isn't one that provides read and write access to a single Amazon S3 bucket, so you can create your own policy instead.

   In the navigation pane on the left, click **Policies** and then click **Create Policy**.
3. Next to **Create Your Own Policy**, click **Select**.
4. Enter **`read-write-app-bucket`** for the policy name.
5. Add the following permissions to the policy document. Ensure that you replace the resource ARN (`arn:aws:s3:::productionapp`) with a real one appropriate to your environment.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:ListAllMyBuckets",
      "Resource": "arn:aws:s3:::*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket",
        "s3:GetBucketLocation"
        ],
      "Resource": "arn:aws:s3:::productionapp"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObject"
      ],
      "Resource": "arn:aws:s3:::productionapp/*"
    }
  ]
}
```

The `ListBucket` permission allows users to view objects in the `productionapp` bucket. The `GetObject`, `PutObject`, `DeleteObject` permissions allows users to view, update, and delete contents in the `productionapp` bucket.

6. Click **Create Policy**.

   The new policy appears in the list of managed policies.
7. In the navigation pane on the left, click **Roles** and then click **Create New Role**.
8. Enter **`UpdateAPP`** for the role name, and then click **Next Step**.
9. Under **Select Role Type**, select **Role for Cross-Account Access**, and then click the **Select** button next to **Provide access between AWS accounts you own**.
10. Enter the Development account ID.

    For this walkthrough, we're using the example account ID **`111111111111`** for the Development account. You should use a valid account ID. If you use an invalid account ID, such as **`111111111111`**, IAM will not let you create the new role.

    For now you do not need to require users to have multi-factor authentication (MFA) in order to assume the role, so leave that option unselected. If you select this option in your environment, then only users who signed in using a one-time password (OTP) from a multi-factor authentication program or device can assume the role. Note that the user cannot enter the OTP at when switching roles - it must be entered when the user initially signs in. For more information, see Using Multi-Factor Authentication (MFA) Devices with AWS (p. 69)
11. Click **Next Step** to set the permissions that will be associated with the role.
12. Select the box next to the policy that you created previously and then click **Next Step**.
13. The Review page appears so you can confirm the settings for the role before it's created. One very important item to note on this page is the link that you can send to your users who need to use this role. Users who click the link go straight to the **Switch Role** page with the **Account ID** and **Role Name** fields already filled in. You can also see this link later on the **Role Summary** page for any cross-account role.
14. After reviewing the role, click **Create Role**.

    The `UpdateAPP` role is displayed in the list of roles.

Now, you must obtain the role's Amazon Resource Name (ARN), which is a unique identifier for the role. When you modify the Developers and Testers group's policy, you will specify the role's ARN to grant or deny permissions.

To obtain the ARN for UpdateAPP

1. In the navigation pane of the IAM console, click **Roles**.

2. From the list of roles, click the `UpdateAPP` role.
3. In the **Summary** section of the details pane, copy the **Role ARN** value.

   The Production account has an account ID of 999999999999, so the role ARN is `arn:aws:iam::999999999999:role/UpdateAPP`. Ensure that you supply the real AWS account ID for your 'production' account.

Summary and What's Next?

At this point, you have established trust between the Production and Development accounts by creating a role in the Production account that identifies the Development account as a trusted principal. You also defined what users who switch to the `UpdateApp` role can do.

Next, modify the permissions for the groups.

Walkthrough: Cross-Account Delegation Part 2 - Granting Users Access to the Role

This is part two of a walkthrough. For the overview, see Walkthrough: Delegating Access Across AWS Accounts Using IAM Roles (p. 151)

At this point, both Testers and Developers group members have permissions that allow them to freely test applications in the Development account. Here are the steps required to add permissions to allow switching to the role.

To modify the Developers group to allow them to switch to the UpdateApp role

1. Sign in as an administrator in the Development account, and open the IAM console.
2. Click **Groups**, then select **Developers**.
3. Expand the **Permissions** section and then click **Attach Another Policy**.
4. Select **Custom Policy** and then click the **Select** button.
5. Enter a policy name like **`allow-assume-S3-role-in-production`**.
6. Add the following policy statement to allow the `AssumeRole` action on the `UpdateAPP` role in the Production account. Be sure that you change *`PRODUCTION-ACCOUNT-ID`* in the `Resource` element to the actual AWS account ID of the Production account.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "sts:AssumeRole",
    "Resource": "arn:aws:iam::PRODUCTION-ACCOUNT-ID:role/UpdateAPP"
  }
}
```

   The `Allow` effect explicitly allows the Developers group access to the `UpdateAPP` role in the Production account. Any developer who tries to access the role will succeed.
7. Click **Apply Policy** to add the policy to the Developer group.

In most environments, the following procedure is likely not needed. If, however, you use Power User permissions, then some groups might already have the ability to switch roles. The following procedures shows how to add a "Deny" permission to the Testers group to ensure that they cannot assume the role.

If this procedure is not needed in your environment, then we recommend that you do not add it - "Deny" permissions make the overall permissions picture more complicated to manage and understand. Use "Deny" permissions only when there is not a better option.

To modify the Testers group to deny permission to assume the UpdateApp role

1. Click **Groups**., then select **Testers**.
2. Expand the **Permissions** section and then click **Attach Another Policy**.
3. Select **Custom Policy** and then click the **Select** button.
4. Enter a policy name like **`deny-assume-S3-role-in-production`**.
5. Add the following policy statement to deny the `AssumeRole` action on the `UpdateAPP` role. Be sure that you change *`PRODUCTION-ACCOUNT-ID`* in the `Resource` element to the actual AWS account ID of the Production account.

   ```
   {
     "Version": "2012-10-17",
     "Statement": {
       "Effect": "Deny",
       "Action": "sts:AssumeRole",
       "Resource": "arn:aws:iam::PRODUCTION-ACCOUNT-ID:role/UpdateAPP"
     }
   }
   ```

   The `Deny` effect explicitly denies the Testers group access to the `UpdateAPP` role in the Production account. Any tester who tries to access the role will get an access denied message.
6. Click **Apply Policy** to add the policy to the Tester group.

Summary and what's next?

The Developers group now has permissions to use the `UpdateAPP` role in the Production account. The Testers group is prevented from using the `UpdateAPP` role.

Next, you'll learn how David, a developer, can access the `productionapp` bucket in the Production account by using the AWS Management Console, the AWS CLI commands, and the `AssumeRole` API call.

Walkthrough: Cross-Account Delegation Part 3 - Switching to a Role

This is part three of a walkthrough. For the overview, see Walkthrough: Delegating Access Across AWS Accounts Using IAM Roles (p. 151)

After completing the first two parts of this walkthrough, you have a role that grants access to a resource in the Production account, and one group in the Development account whose users are allowed to use that role. The role is now ready to use, and this part discusses switching to the role from the AWS Management Console, the AWS CLI and the AWS API.

Important
Only IAM users can use a role. If you use AWS root account credentials, access is denied.

Topics

Switching Roles in the AWS Management Console

If David needs to work with in the Production environment in the AWS Management Console, he can do so by using **Switch Role**. He specifies the account ID or alias and the role name, and his permissions immediately switch to those permitted by the role. He can then use the console to work with the `productionapp` bucket, but cannot work with any other resources in Production. While David is using the role, he also cannot make use of his power-user privileges in the Development account because only one set of permissions can be in effect at a time.

There are two ways that David might enter the Switch Role page

- David receives a link from his administrator that points to a pre-defined Switch Role configuration. The link is provided to the administrator on the final page of the **Create Role** wizard or on the **Role Summary** page for a cross-account role. Clicking this link takes David to the **Switch Role** page with the **Account ID** and **Role Name** fields already filled in. All David needs to do is click the **Switch Role** button and he's done.
- The administrator does not send the link in email, but instead sends the Account ID number and Role Name values. David must manually enter them to switch roles. This is illustrated in the following procedure.

To use a role in the AWS Management Console

1. David logs into the AWS console using his normal user that is in the Development group.
2. He clicks on the link provided to him by his administrator in email. This takes him to the **Switch Role** page with the Account ID or alias and the role name information already filled in.

   —or—

   He clicks on his name (the Identity menu) in the navigation bar, and then clicks **Switch Role**. Since this is the first time David accesses the role, he must manually enter the Production account ID number (`999999999999`) and the role name (`UpdateApp`).
3. To help him stay aware of which role (and associated permissions) are currently active, he enters `PRODUCTION` in the **Display Name** text box, selects the red color option, and then clicks **Switch Role**.
4. David can now use the Amazon S3 console to work with the Amazon S3 bucket, or any other resource to which the `UpdateApp` role has permissions.
5. When he is done with the work he needs to do, David can return to his original permissions by clicking the **PRODUCTION** role display name in the navigation bar, and then clicking **Back to David @ 111111111111**.
6. The next time David wants to switch roles and selects the Identity menu in the navigation bar, he sees the PRODUCTION entry still there from last time. He can simply click that entry to switch roles immediately without having to reenter the account ID and role name.

Switching Roles with the AWS CLI

If David needs to work with in the Production environment at the command line, he can do so by using the AWS CLI. He runs the `aws sts assume-role` command and passes the role ARN to get temporary security credentials for that role. He then configures those credentials in environment variables so subsequent AWS CLI commands work using the role's permissions. While David is using the role, he cannot make use of his power-user privileges in the Development account because only one set of permissions can be in effect at a time.

Note that all access keys and tokens are examples only and cannot be used as shown. Replace with the appropriate values from your live environment.

To use a role in the AWS CLI

1. David opens a command prompt window, and confirms that the AWS CLI client is working by running the command:

   ```
   aws help
   ```

   Note
   David's default environment uses the `David` user credentials from his default profile that he created with the `aws configure` command. For more information, see Configuring the AWS Command Line Interface in the *AWS Command Line Interface User Guide*.

2. He begins the switch role process by running the following command to switch to the `UpdateApp` role in the Production account. He got the role ARN from the administrator that created the role. The command requires that you provide a session name as well, you can choose any text you like for that.

   ```
   aws sts assume-role --role-arn "arn:aws:iam::999999999999:role/UpdateApp"
   --role-session-name "David-ProdUpdate"
   ```

   David then sees the following in the output:

   ```
   {
       "Credentials": {
           "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
           "SessionToken": "AQoDYXdzEGcaEXAMPLE2gsY
   ULo+Im5ZEXAMPLEeYjs1M2FUIgIJx9tQqNMBEXAMPLE
   CvSRyh0FW7jEXAMPLEW+vE/7s1HRpXviG7b+qYf4nD00EXAMPLEmj4wxS04L/uZEXAMPLE
   CihzFB5lTYLto9dyBgSDy
   EXAMPLE9/g7QRUhZp4bqbEXAMPLENwGPyOj59pFA4lNKCIkVgkREXAMPLE
   jlzxQ7y52gekeVEXAMPLEDiB9ST3Uuysg
   sKdEXAMPLE1TVastU1A0SKFEXAMPLEiywCC/Cs8EXAMPLEpZgOs+6hz4AP4KEXAMPLERbASP+4eZ
   ScEXAMPLEsnf87e
   NhyDHq6ikBQ==",
           "Expiration": "2014-12-11T23:08:07Z",
           "AccessKeyId": "AKIAIOSFODNN7EXAMPLE"
       }
   }
   ```

3. David sees the three pieces he needs in the Credentials section of the output.

   - AccessKeyId
   - SecretAccessKey
   - SessionToken

   David needs to configure the AWS CLI environment to use these parameters in subsequent calls. For information about the various ways to configure your credentials, see Configuring the AWS Command Line Interface. You cannot use the `aws configure` command because it does not support capturing the session token. However, you can manually enter the information into a configuration file. Because these are temporary credentials with a relatively short expiration time, it is easiest to add them to the environment of your current command line session.

4. To add the three values to the environment, David cuts and pastes the output of the previous step into the following commands. Note that you might want to cut and paste into a simple text editor to address line wrap issues in the output of the session token. It must be entered as a single long string, even though it is shown line wrapped here for clarity.

   ```
   set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
   set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
   set AWS_SESSION_TOKEN=AQoDYXdzEGcaEXAMPLE2gsY
   ULo+Im5ZEXAMPLEeYjs1M2FUIgIJx9tQqNMBEXAMPLECvS
   Ryh0FW7jEXAMPLEW+vE/7s1HRpXviG7b+qYf4nD00EXAMPLEmj4wxS04L/uZEXAMPLE
   CihzFB5lTYLto9dyBgSDyEXA
   MPLEKEY9/g7QRUhZp4bqbEXAMPLENwGPyOj59pFA4lNKCIkVgkREXAMPLE
   jlzxQ7y52gekeVEXAMPLEDiB9ST3UusKd
   EXAMPLE1TVastU1A0SKFEXAMPLEiywCC/Cs8EXAMPLEpZgOs+6hz4AP4KEXAMPLERbASP+4eZS
   cEXAMPLENhykxiHen
   DHq6ikBQ==
   ```

   At this point, any following commands will run under the permissions of the role identified by those credentials. In David's case, the `UpdateApp` role.

5. Run the command to access the resources in the Production account. In this example, David simply lists the contents of his S3 bucket with the following command.

   ```
   aws s3 ls s3://productionapp
   ```

   Because Amazon S3 bucket names are universally unique, there is no need to specify the account ID that owns the bucket. To access resources for other AWS services, refer to the AWS CLI documentation for that service for the commands and syntax required to reference its resources.

Using AssumeRole From the API

When David needs to make an update to the Production account from code, he makes an `AssumeRole` call to assume the `UpdateAPP` role. The call returns temporary credentials that he can use to access the `productionapp` bucket in the Production account. With those credentials, David can make API calls to update the `productionapp` bucket but cannot make API calls to access any other resources in the Production account, even though he has power-user privileges in the Development account.

To use a role by making API calls

1. David calls `AssumeRole` as part of an application. He must specify the `UpdateAPP` ARN: `arn:aws:iam::999999999999:role/UpdateAPP`.

   The response from the `AssumeRole` call includes the temporary credentials with an `AccessKeyId`, a `SecretAccessKey`, and an `Expiration` time that indicates when the credentials expire and you must request new ones.
2. With the temporary credentials, David makes an `s3:PutObject` call to update the `productionapp` bucket. He would pass the credentials to the API call as the `AuthParams` parameter. Because the temporary role credentials have only read and write access to the `productionapp` bucket, any other actions in the Production account are denied.

For a code sample (using Python), see Switching IAM Roles by Using the API (p. 139).

Summary

You have completed the cross-account API access walkthrough. You created a role to establish trust with another account and defined what actions trusted entities can take. Then, you modified a group policy to control which IAM users can access the role. As a result, developers from the Development account can make updates to the `productionapp` bucket in the Production account by using temporary credentials.

How Roles Differ from Resource-Based Policies

For some AWS services, you can grant cross-account access to your resources by putting a policy directly on the resource you want to share, rather than using a role as a proxy. The resource you want to share must support resource-based policies (p. 172). The policy specifies who (in the form of a list of AWS account IDs) can access that resource.

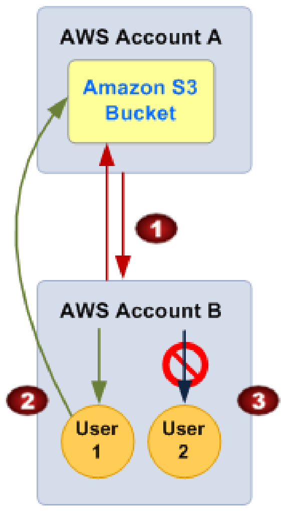
Cross-account access using resource-based policies has some practical benefits over using roles. While using resources that are granted using resource-based policies, the user who has access to the shared resource is still working in the trusted account; in effect, the user has access to both accounts concurrently. This is useful for tasks such as copying information to or from the shared resource in the other account.

Resource-based policies are supported by the following AWS services and resources:

- Amazon S3 buckets. The policy is attached to the bucket, but the policy controls access to both the bucket and the objects in it; you can use ACLs to control access to individual objects. For more information, go to Access Control in the *Amazon Simple Storage Service Developer Guide*.
- Amazon SNS topics. For more information, go to Managing Access to Your Amazon SNS Topics in the *Amazon Simple Notification Service Developer Guide*.
- Amazon SQS queues. For more information, go to Appendix: The Access Policy Language in the *Amazon Simple Queue Service Developer Guide*.

About Delegating AWS Permissions by Using a Resource-based Policy

After a resource grants permissions to your AWS account as a principal in it's resource-based policy, you can then delegate permissions to specific users or groups under your AWS account. You attach a policy to the user or group that you want to delegate the permissions to. Note that you can only delegate permissions equivalent to, or less than, the permissions granted to your account by the resource owning account. For example, if your account is granted full access to the resources of another AWS account then you can delegate full access, list access, or any other partial access to users under your AWS account. If, on the other hand, your account is granted list access only then you can delegate only list access. If you try to delegate more permissions than your account has, your users will still have only list access only. This is illustrated in the following figure. For information about attaching a policy to a user or group, see Managing Policies (p. 192).



1. Account A gives Account B full access to Account A's Amazon S3 bucket by naming Account B as a principal in the policy. As a result, Account B is authorized to perform any action on Account A's bucket, and the Account B administrator can delegate access to its users in Account B.
2. The Account B administrator grants User 1 read only access to Account A's Amazon S3 bucket. User 1 can view the objects in Account A's bucket. The level of access Account B can delegate is equivalent to, or less than, the access the account has. In this case, the full access granted to Account B is filtered to read only for User 1.
3. The Account B administrator does not give access to User 2. Because users by default do not have any permissions except those explicitly granted, User 2 does not have access to Account A's Amazon S3 bucket.

Important
In the preceding example, if Account B had used wildcards (*) to give User 1 full access to all its resources, User 1 would automatically have access to any resources that Account B has access to, including access granted by other accounts to those accounts' resources. In this case, User 1 would have access to any Account A resources granted to Account B, in addition to those explicitly granted to User 1.
IAM evaluates a user's permissions at the time the user makes a request. Therefore, if you use wildcards (*) to give users full access to your resources, users are able to access any resources your AWS account has access to, even resources you add or gain access to after creating the user's policy.

For information about permissions, policies, and the access policy language you use to write policies, see Permissions and Policies (p. 171).

Important
Give access only to entities you trust, and give the minimum amount of access necessary. Whenever the trusted entity is another AWS account, that account can in turn delegate access to any of its IAM users. The trusted AWS account can delegate access only to the extent that it has been granted access; it cannot delegate more access than the account itself has been granted.

Managing Server Certificates

Topics

This section lists the IAM actions for working with X.509 public key certificates, which we refer to as *server certificates*.

Server certificates are used in some AWS services, including the following:

- AWS Elastic Beanstalk. For more information, see Configuring HTTPS for your AWS Elastic Beanstalk Environment in the *AWS Elastic Beanstalk Developer Guide*.
- Elastic Load Balancing. For more information, see Update an SSL Certificate for a Load Balancer in the *Elastic Load Balancing Developer Guide*.
- CloudFront. For more information, see the Using an HTTPS Connection to Access Your Objects in the *Amazon CloudFront Developer Guide*.
- AWS OpsWorks. For more information, see Using SSL in the *AWS OpsWorks User Guide*.

Creating, Uploading, and Deleting Server Certificates

This section describes the process of generating a server certificate and preparing it to use with AWS products through IAM. To create a certificate, you perform the following series of tasks.

Topics

Prerequisites

Creating and uploading a certificate requires the following command-line tools:

- OpenSSL. You use this tool to generate a public/private key pair and a certificate signing request. Instructions for installing and configuring OpenSSL are provided in Install OpenSSL (p. 168) and Configure OpenSSL (p. 168) sections.
- The AWS command-line interface (CLI). You use a CLI command to upload the certificate to AWS. For information about installing the CLI, see Installing the AWS Command Line Interface.

Create a Private Key

You need a unique private key to create your Certificate Signing Request (CSR).

To create a private key

- At the command line, use the `openssl genrsa` command and the following syntax:

```
openssl genrsa 2048 > private-key.pem
```

For *`private-key.pem`*, specify your own file name. In the example, `2048` represents 2048-bit encryption. AWS also supports 1024-bit and 4096-bit encryption.

Note
Amazon CloudFront enforces a maximum key size of 2048 bits. For more information, see Requirements and Limits on Using SSL Certificates with CloudFront in the *Amazon CloudFront Developer Guide*.

We recommend you create an RSA key that is 2048 bits.

Create a Certificate Signing Request

The next step is to create a Certificate Signing Request (CSR). This is a file that you can send to a certificate authority (CA) to apply for a server certificate.

To create a CSR

- Use the `openssl req` command to create a CSR and the following syntax:

```
openssl req -new -key private-key.pem -out csr.pem
```

The output will look similar to the following example:

```
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank.
```

```
For some fields there will be a default value.
If you enter '.', the field will be left blank.
```

The following table can help you create your certificate request.

Name	Description	Example
Country Name	The two-letter ISO abbreviation for your country.	US = United States
State or Province	The name of the state or province where your organization is located. This name cannot be abbreviated.	Washington
Locality Name	The name of the city where your organization is located.	Seattle
Organization Name	The full legal name of your organization. Do not abbreviate your organization name.	Example Corp.
Organizational Unit	Optional, for additional organization information.	Marketing
Common Name	The fully qualified domain name for your CNAME. You will receive a certificate name check warning if this is not an exact match.	www.example.com
Email address	The server administrator's email address	someone@example.com

Note
The Common Name field is often misunderstood and is completed incorrectly. The common name is typically your host plus domain name. It will look like "www.example.com" or "example.com". You need to create a CSR using your correct common name.

Submit the CSR to a Certificate Authority

Your CSR contains information identifying you. To apply for a server certificate, send your CSR to a certificate authority (CA). The CA might require other credentials or proofs of identity.

If the request for a certificate is successful, the CA returns an identity certificate (and possibly a chain certificate) that is digitally signed.

AWS does not recommend a specific CA. For a partial listing of available CAs, see Third-Party Certificate Authorities.

Upload the Signed Certificate

When you receive your server certificate from the certificate authority (CA), you can upload the certificate to IAM along with the private certificate and a certificate chain. After you upload the certificates to IAM, the certificates are available for other AWS services to use.

To upload server certificates on IAM, you use the AWS command line interface. For more information, see the AWS Command Line Interface User Guide.

Note
A certificate authority might return a certificate in a format that is not supported by IAM. You can convert the certificate to the correct format (X.509 PEM) by using OpenSSL. The specific command depends on the current format of your certificate.

To upload your certificate, you need three files:

- Your server certificate in PEM format.
- Your private key in PEM format.
- A certificate chain file. This contains all the intermediate certificates and the root certificate of the CA. The certificate chain lets an end user's browser build a certificate chain to a root certificate it trusts. As a result, the browser can implicitly trust your certificate.

You can upload all three files from the command line with one command.

To upload a server certificate

- Use the `aws iam upload-server-certificate` command to upload a signed certificate:

```
aws iam upload-server-certificate --server-certificate-name certificate_ob
ject_name --certificate-body file://public_key_certificate_file --private-
key file://privatekey.pem --certificate-chain file://certificate_chain_file
```

Note
Notice that when you specify a file as a parameter (for example, for the `certificate-body` and `private-key` parameters), you include `file://` as part of the file name.

If you are uploading a self-signed certificate and it's not important that browsers implicitly accept the certificate, you can omit the `--certificate-chain` option and upload just the server certificate and private key, as shown in the following example:

```
aws iam upload-server-certificate --server-certificate-name certificate_ob
ject_name --certificate-body file://public_key_certificate_file --private-
key file://privatekey.pem
```

CloudFront: If you are uploading a server certificate specifically for use with Amazon CloudFront distributions, you must specify a path using the `--path` option. The path must begin with `/cloudfront` and must include a trailing slash (for example, `/cloudfront/test/`). Enter the following command:

```
aws iam upload-server-certificate --server-certificate-name certificate_ob
ject_name --certificate-body file://public_key_certificate_file --private-
key file://privatekey.pem --certificate-chain file://certificate_chain_file
 --path /cloudfront/path/
```

You assign your own name to the certificate (the *`certificate_object_name`* parameter in the preceding commands). For information about limitations on server certificate names, see Limitations on IAM Entities (p. 13).

For the Amazon CloudFront example, the command displays the ARN and unique ID (p. 12) of the uploaded certificate. If you use the CloudFront API to add a certificate to a distribution, you need the certificate ID. (If you use the CloudFront console for this task, the console displays the certificate name that you specified in the `--server-certificate-name` option.) For more information about using server certificates with CloudFront, see Using an HTTPS Connection to Access Your Objects in the *Amazon CloudFront Developer Guide*.

When you upload your certificates, IAM validates the certificates with the following criteria:

- Certificates must follow the X.509 PEM format.
- The current date must be between the certificate's start and end date.
- Public and private certificate files must contain only a single certificate.
- The private key must match the public key that is in the certificate.
- The private key must be an RSA private key in PEM format, where the PEM header is BEGIN RSA PRIVATE KEY and the footer is END RSA PRIVATE KEY (as shown in Sample Certificates (p. 167)).
- The private key cannot be encrypted with a password.
- The certificate chain must include all of your CA's intermediary certificates that lead to the root certificate, and ends with your CA's root certificate. Typically, both intermediary and root certificates are provided by a CA in a bundled file with the proper chained order. If a certificate bundle is not available or not available in the required order, you can create your own file similar to the sample certificate chain in Sample Certificates (p. 167). Use the intermediary certificates that were provided by your CA. Any intermediaries that are not involved in the chain of trust path must not be included.

  After you upload your certificate chain to AWS, you can use SSL Checker to verify it.

  Note

  - The order of intermediary certificates should be documented by the CA. AWS does not recommend any one CA. For a listing of some CAs, see Third-Party Certificate Authorities.
  - Although the root certificate is optional, you can include it so that you can run full chain of trust verifications, such as SSL Checker.

If you have certificates that result in an error when you upload them, ensure that they meet the criteria, and then try uploading them again.

To see sample certificates that are valid with IAM, see Sample Certificates (p. 167).

Note
If you are having difficulties uploading a server certificate, you might find it helpful to follow the steps outlined in the following blog post: Setting up SSL on an Amazon Elastic Load Balancer.

Verify the Certificate Object

After the server certificate is uploaded, you can verify that the information is stored in IAM. Each certificate object has a unique Amazon Resource Name (ARN) and ID. You can request these details for a specific certificate object by referencing the name of the certificate object.

To view the certificate object's ARN and ID

- Use the `aws iam get-server-certificate` command to verify the certificate object:

```
aws iam get-server-certificate --server-certificate-name certificate_ob
ject_name
```

  The output will look similar to the following example.

```
arn:aws:iam::Your_AWS_Account_ID:server-certificate/Your_Certificate_Ob
ject_Name Certificate_Object_GUID
```

You have now completed the process for creating and uploading a signed certificate. For information about setting up an Elastic Load Balancing load balancer with HTTPS support, see the AWS Command Line Interface (AWS CLI) examples in the How to Set Up a Load Balancer with HTTPS Support section of the *Elastic Load Balancing Developer Guide*.

Delete a Certificate Object

If you no longer need a certificate, you can delete it.

To delete a certificate object

- Use the aws iam delete-server-certificate command to remove an individual certificate.

  ```
  aws iam delete-server-certificate --server-certificate-name certificate_ob
  ject_name
  ```

  If the command is successful, no output is displayed.

Sample Certificates

The following certificates show the valid format that IAM accepts for server certificates and their associated private key and certificate chain.

The server certificate associates your public key with your identity. When you submit your Certificate Signing Request (CSR) to a certificate authority (CA), a server certificate is returned to you by the CA. The following example shows the format of a server certificate:

```
-----BEGIN CERTIFICATE-----
your-certificate-here
-----END CERTIFICATE-----
```

The private key allows you to decrypt messages that are encrypted with your public key. The following example shows the format of a key:

```
-----BEGIN RSA PRIVATE KEY-----
your-key-here
-----END RSA PRIVATE KEY-----
```

The certificate chain includes all intermediary certificates that lead to the root certificate, as shown in the following example. Intermediaries that are not involved in the trust path must not be included. The chain ends with your CA's root certificate. Typically, both intermediary and root certificates are provided by a CA in a bundled file with the proper chained order.

Sample certificate chain

```
-----BEGIN CERTIFICATE-----
Intermediate certificate 2
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
Intermediate certificate 1
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
Optional: Root certificate
-----END CERTIFICATE-----
```

Install OpenSSL

If you don't already have OpenSSL installed, follow the instructions in this section.

To install OpenSSL on Linux or Unix

1. Go to OpenSSL: Source, Tarballs (http://www.openssl.org/source/).
2. Download the latest source and build the package.

To install OpenSSL on Windows

1. Go to OpenSSL: Binary Distributions (http://www.openssl.org/related/binaries.html).
2. Click **OpenSSL for Windows**.

   A new page displays with links to the Windows downloads.
3. If it is not already installed on your system, select the **Microsoft Visual C++ 2008 Redistributables** link appropriate for your environment and click **Download**. Follow the instructions provided by the **Microsoft Visual C++ 2008 Redistributable Setup Wizard**.

   Note
   If you are not sure if the Microsoft Visual C++ 2008 Redistributables is already installed on your system, you can try installing OpenSSL first. The OpenSSL installer will display an error if the Microsoft Visual C++ 2008 Redistributables is not yet installed. Make sure you install the architecture (32-bit or 64-bit) that matches the version of OpenSSL that you install.
4. After you have installed the Microsoft Visual C++ 2008 Redistributables, select the appropriate version of the OpenSSL binaries for your environment and save the file locally. Launch the **OpenSSL Setup Wizard**.
5. Follow the instructions described in the **OpenSSL Setup Wizard**.

Configure OpenSSL

Before you use OpenSSL commands, you must configure the operating system so that it has information about the location where OpenSSL is installed.

To configure OpenSSL on Linux or Unix

1. At the command line, set the `OpenSSL_HOME` variable to the location of the OpenSSL installation:

   ```
   export OpenSSL_HOME=path_to_your_OpenSSL_installation
   ```

2. Set the path to include the OpenSSL installation:

```
export PATH=$PATH:$OpenSSL_HOME/bin
```

Note
Any changes you make to environment variables using the `export` command are valid only for the current session. You can make persistent changes to the environment variables by setting them using your shell configuration file. For more information, see the documentation for your operating system.

To configure OpenSSL on Windows

1. Open a **Command Prompt** window.
2. Set the `OpenSSL_HOME` variable to the location of the OpenSSL installation:

```
set OpenSSL_HOME=path_to_your_OpenSSL_installation
```

3. Set the `OpenSSL_CONF` variable to the location of the configuration file in your OpenSSL installation:

```
set OpenSSL_CONF=path_to_your_OpenSSL_installation\bin\openssl.cfg
```

4. Set the path to include the OpenSSL installation:

```
set Path=%Path%;%OpenSSL_HOME%\bin
```

Note
Any changes you make to Windows environment variables in a **Command Prompt** window are valid only for the current command line session. You can make persistent changes to the environment variables by setting them as system properties. The exact procedures depends on what version of Windows you're using. (For example, in Windows 7, open **Control Panel** > **System and Security** > **System**. Then choose **Advanced system settings** > **Advanced** tab > **Environment Variables**.) For more information, see the Windows documentation.

Actions on Server Certificates

The following table describes actions you can use to manage server certificates in IAM.

Action	API	Command Line Interface
Delete a server certificate	DeleteServerCertificate	aws iam delete-server-certificate
Get server certificate information	GetServerCertificate	aws iam get-server-certificate
List server certificates	ListServerCertificates	aws iam list-server-certificates
Update (rename) server certificates	UpdateServerCertificate	aws iam update-server-certificate
Upload server certificates	UploadServerCertificate	aws iam upload-server-certificate

For more information about these actions, see the AWS Identity and Access Management API Reference or the AWS Command Line Interface User Guide.

Renaming Server Certificates

When you rename a server certificate (using the aws iam update-server-certificate CLI command or the UpdateServerCertificate API), the unique ID for the server certificate remains the same (for more information about ID, see *Unique IDs* in IAM Identifiers (p. 8)). However, IAM does not automatically update policies that refer to the server certificate as a resource to use the new name. You must manually do that. For example, Bob is a developer in the company ABC and has a policy attached to him that lets him manage the company's build server certificate, `arn:aws:iam::123456789012:server-certificate/abc/certs/build`. If an admin changes the name of the build server certificate to `build_01` or changes the path for the server certificate, the admin also needs to update the policy attached to Bob to use the new name or path so that Bob can continue to manage that server certificate.

Permissions and Policies

When you use your account's root credentials, you can access all the resources in your AWS account. In contrast, when you create IAM users, IAM groups, or IAM roles, you must explicitly give permissions to these entities so that users can access your AWS resources.

This section describes *permissions*, which are rights that you grant to a user, group, or role that define what tasks users are allowed to perform in your AWS account. To define permissions, you use *policies*, which are documents in JSON format.

To learn more, we recommend you read the following sections:

- Overview of AWS IAM Permissions (p. 171) — This section discusses types of permissions, how to grant them, and how to manage permissions.
- Overview of IAM Policies (p. 174) — This section discusses how to specify what actions are allowed, which resources to allow the actions on, and what the effect will be when the user requests access to the resources.
- Managing Policies (p. 192) — This section describes how to create and manage policies by using the AWS Management Console, the AWS CLI, and the IAM API.
- Testing IAM Policies (p. 200) — This section describes how to test user and group policies using the IAM Policy Simulator.
- Using Policy Validator (p. 201) — This section describes how to use the Policy Validator when you have policies that do not comply with the AWS policy grammar.
- AWS IAM Policy Reference (p. 208) — This section describes the elements, variables, and evaluation logic used in policies.
- Example Policies for Administering IAM Resources (p. 97) — This section lists policies that let users perform tasks specific to IAM, like administer users, groups, and credentials.
- Example Policies for Administering AWS Resources (p. 203) — This section lists policies that let users perform tasks with other AWS services, like Amazon S3, Amazon EC2, and DynamoDB.

Overview of AWS IAM Permissions

Permissions let you specify who has access to AWS resources, and what actions they can perform on those resources. Every IAM user starts with no permissions. In other words, by default, users can do nothing, not even view their own access keys. To give a user permission to do something, you can add the permission to the user (that is, attach a policy to the user) or add the user to a group that has the desired permission.

For example, you might grant a user permission to list his or her own access keys. You might also expand that permission and also let each user create, update, and delete their own keys.

When you give permissions to a group, all users in that group get those permissions. For example, you can give the Admins group permission to perform any of the IAM actions on any of the AWS account resources. Another example: You can give the Managers group permission to describe the AWS account's Amazon EC2 instances.

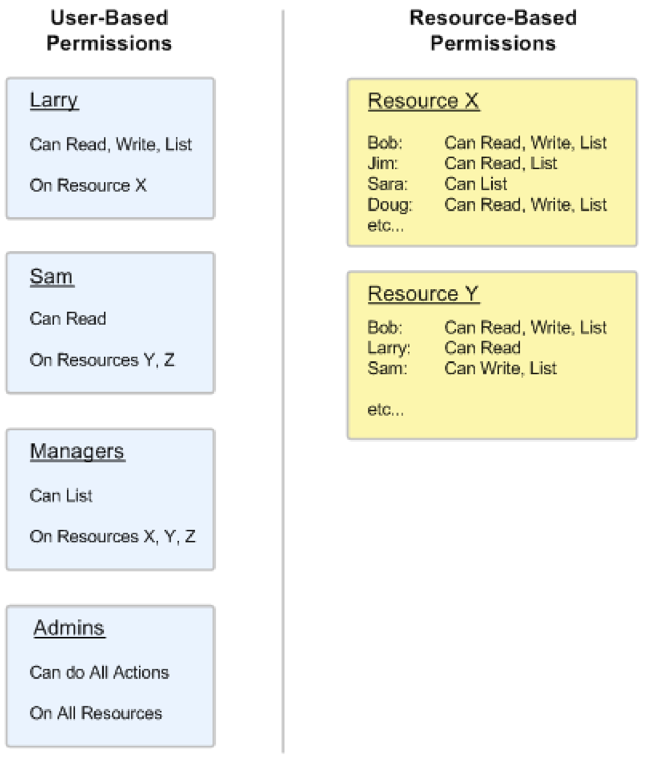
User-Based and Resource-Based Permissions

Permissions can be assigned in two ways: as *user-based permissions* or as *resource-based permissions*. User-based permissions are attached to an IAM user, group, or role and let you specify what that user, group, or role can do. For example, you can assign permissions to the IAM user named Bob, stating that he has permission to use the Amazon Elastic Compute Cloud (Amazon EC2) `RunInstances` action and that he has permission to get items from an Amazon DynamoDB table named `MyCompany`. The user Bob might also be granted access to manage his own IAM security credentials.

Resource-based permissions are attached to a resource. You can specify resource-based permissions for Amazon S3 buckets, Amazon SNS topics, Amazon SQS queues, and AWS Key Management Service encryption keys. Resource-based permissions let you specify who has access to the resource and what actions they can perform on it.

> **Note**
> There's a difference between *resource-based* permissions and *resource-level* permissions. Resource-based permissions are permissions you can attach directly to a resource, as described in this topic. Resource-level permissions refers to the ability to specify not just what actions users can perform, but which resources they're allowed to perform those actions on. Some AWS services let you specify permissions for actions, but don't let you specify the individual resources for those actions. Other services let you specify permissions for a combination of actions and individual resources.
> Resource-based permissions are supported only by Amazon S3, Amazon SNS, Amazon SQS, and AWS Key Management Service. For a list of which services support resource-level permissions, see AWS Services That Support IAM (p. 265).

The following figure illustrates both types of permissions.



A user who has specific permissions might request a resource that also has permissions attached to it. In that case, both sets of permissions are evaluated when AWS determines whether to grant access to the resource. For information about how policies are evaluated, see IAM Policy Evaluation Logic (p. 243).

> **Note**
> Amazon S3 supports policies both for IAM users and for resources (referred to in Amazon S3 as *bucket policies*). In addition, Amazon S3 supports a permission mechanism known as an *ACL* that's independent of IAM policies and permissions. You can use IAM policies in combination with Amazon S3 ACLs. For more information, see Access Control in the *Amazon Simple Storage Service Developer Guide*.

Resource Creators Do Not Automatically Have Permissions

Someone using the AWS account's security credentials has permission to perform any action on resources that belong to the account. However, this isn't true for IAM users. An IAM user might be granted access to create a resource, but the user's permissions, even for that resource, are limited to what's been explicitly granted. The user's permission can be revoked at any time by the account owner or by another user who has been granted access to manage user permissions.

Granting Permissions Across AWS Accounts

You can directly grant IAM users in your own account access to your resources. If users from another account need access to your resources, you can create an IAM role, which is an entity that includes permissions but that isn't associated with a specific user. Users from other accounts can then use the role and access resources according to the permissions you've assigned to the role. For more information, see IAM Roles (Delegation and Federation) (p. 116).

Note
For services that support resource-based policies as described in User-Based and Resource-Based Permissions (p. 172) (such as Amazon S3, Amazon SNS, and Amazon SQS), an alternative to using roles is to attach a policy to the resource (bucket, topic, or queue) that you want to share. The resource-based policy can specify the AWS account that has permissions to access the resource.

Permissions For One Service to Access Another

Many AWS services access other AWS services. For example, several AWS services—including Amazon Elastic MapReduce, Elastic Load Balancing, and Auto Scaling—manage Amazon EC2 instances. Other AWS services make use of Amazon S3 buckets, Amazon SNS topics, Amazon SQS queues, and so on.

The scenario for managing permissions in these cases varies by service. Here are some examples of how permissions are handled for different services:

- In Auto Scaling, users must have permission to use Auto Scaling, but don't need to be explicitly granted permission to manage Amazon EC2 instances.
- In AWS Data Pipeline, an IAM role determines what a pipeline can do; users need permission to assume the role. (For details, see Granting Permissions to Pipelines with IAM in the *AWS Data Pipeline Developer Guide*.)

For details about how to configure permissions properly so that an AWS service is able to accomplish the tasks you intend, refer to the documentation for the service you are calling.

Overview of IAM Policies

This section provides an overview of IAM policies. A policy is a document that formally states one or more permissions.

Topics

Introduction

To assign permissions to a user, group, role, or resource, you create a *policy*, which is a document that explicitly lists permissions. In its most basic sense, a policy lets you specify the following:

- **Actions**: what actions you will allow. Each AWS service has its own set of actions. For example, you might allow a user to use the Amazon S3 `ListBucket` action, which returns information about the items in a bucket. Any actions that you don't explicitly allow are denied.
- **Resources**: which resources you allow the action on. For example, what specific Amazon S3 buckets will you allow the user to perform the `ListBucket` action on? Users cannot access any resources that you have not explicitly granted permissions to.
- **Effect**: what the effect will be when the user requests access—either allow or deny. Because the default is that resources are denied to users, you typically specify that you will allow users access to resource.

Policies are documents that are created using JSON. A policy consists of one or more *statements*, each of which describes one set of permissions. Here's an example of a simple policy.

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

You can attach this policy to an IAM user or group. If that's the only policy for the user or group, the user or group is allowed to perform only this one action (`ListBucket`) on one Amazon S3 bucket (`example_bucket`).

To specify resource-based permissions, you can attach a policy to the resource, such as an Amazon SNS topic or Amazon S3 bucket. In that case, the policy has to include information about who is allowed to access the resource, known as the *principal*. (For user-based policies, the principal is the IAM user that the policy is attached to, or the user who gets the policy from a group.)

The following example shows a policy that might be attached to an Amazon S3 bucket and that grants permission to a specific AWS account to perform any Amazon S3 actions in `mybucket`. This includes both working with the bucket and with the objects in it. (Because the policy grants trust only to the account, individual users in the account must still be granted permissions for the specified Amazon S3 actions.)

```
{
  "Version": "2012-10-17",
  "Id": "S3-Account-Permissions",
  "Statement": [{
    "Sid": "1",
    "Effect": "Allow",
    "Principal": {"AWS": ["arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:root"]},
    "Action": "s3:*",
    "Resource": [
      "arn:aws:s3:::mybucket",
      "arn:aws:s3:::mybucket/*"
    ]
  }]
}
```

IAM policies control access regardless of the interface. For example, you could provide a user with a password to access the AWS Management Console, and the policies for that user (or any groups the user belongs to) would control what the user can do in the AWS Management Console. Or, you could provide the user with AWS access keys for making API calls to AWS, and the policies would control what actions the user could call through a library or client that uses those access keys for authentication.

For basic example policies that cover common scenarios, see Example Policies for Administering AWS Resources (p. 203), AWS Services That Support IAM (p. 265), and the AWS Policy Examples page in the *AWS Sample Code & Libraries* section of the AWS website.

AWS managed policies and the Policy Generator are available from the IAM console in the AWS Management Console. For more information about creating policies in the console, see Managing Policies (p. 192). Also, you can use the AWS Policy Generator online to create policies for AWS products without accessing the console.

Important
AWS is making improvements to the policy rules engine that enforce a stricter syntax. Starting in March 2015, you cannot save any policy that does not comply with the stricter rules.

To help you correct invalid policies, AWS created the Policy Validator that informs you whenever it detects an invalid policy. One click takes you to an editor that shows both the existing policy and a copy with the recommended changes. You can accept the changes or make further modifications. For more information, see Using Policy Validator (p. 201).

Note
When you apply a custom policy, IAM checks its syntax. However, because IAM evaluates policies at run time using a specific request context (in which multiple policies might be in effect), it cannot check the validity of all resources, actions, and permissions in a custom policy at the time that you apply the policy. If you need help in creating a policy, we recommend using an AWS managed policy or the Policy Generator. For help testing the effects of your IAM policies, see Testing IAM Policies (p. 200).

Multiple Statements and Multiple Policies

You can attach more than one policy to an entity. If you have multiple permissions to grant to an entity, you can put them in separate policies, or you can put them all in one policy.

Generally, each statement in a policy includes information about a single permission. If your policy includes multiple statements, a logical OR is applied across the statements at evaluation time. Similarly, if multiple policies are applicable to a request, a logical OR is applied across the policies at evaluation time.

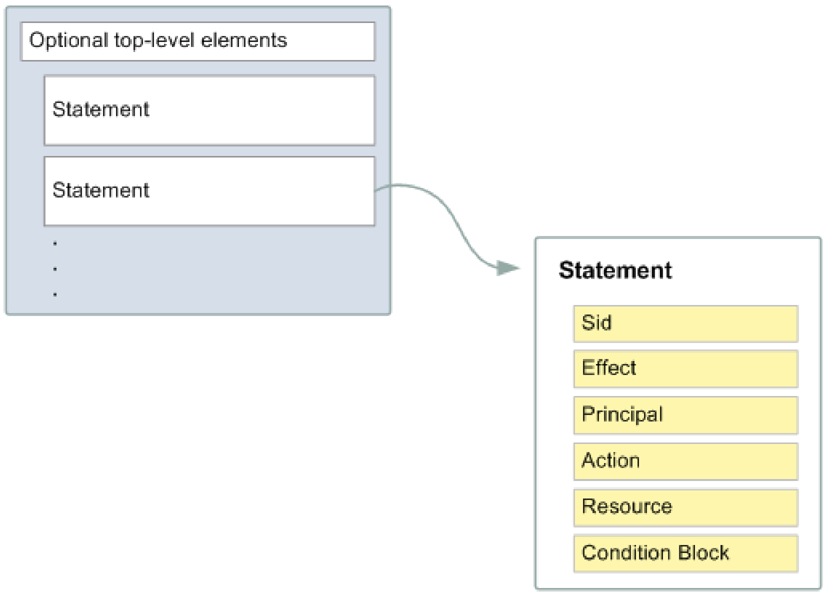
Users often have multiple policies that apply to them (but aren't necessarily *attached* to them). For example, IAM user Bob could have policies attached to him, and other policies attached to the groups he's in. In addition, he might be accessing an Amazon S3 bucket that has its own bucket policy (resource-based policy). All applicable policies are evaluated and the result is always that access is either granted or denied. For more information about the evaluation logic we use, see IAM Policy Evaluation Logic (p. 243).

Policy Structure

Each policy is a JSON document. As illustrated in the following figure, a policy includes:

- Optional policy-wide information (at the top of the document)
- One or more individual *statements*

Each statement includes the core information about a single permission. If a policy includes multiple statements, AWS applies a logical `OR` across the statements at evaluation time. If multiple policies are applicable to a request, AWS applies a logical `OR` across the policies at evaluation time.



The information in a statement is contained within a series of *elements*. For information about these elements, see IAM Policy Elements Reference (p. 209).

Example Policy with Multiple Statements

Policies often include multiple statements, where each statement grants permissions to a different set of resources or grants permissions under a specific condition. For example, the following policy has three statements, each of which grants a separate set of permissions. Assume that the user or group that the policy is attached to is in AWS account `123456789012`. (Policies can't reference resources in other accounts.) The statements do the following:

- The first statement, with a `Sid` (Statement ID) element set to `FirstStatement`, lets users manage their own passwords. The `Resource` element in this statement is "`*`" (which means "all resources"), but in practice, the `ChangePassword` API (or equivalent `change-password` CLI command) affects only the password for the user who makes the request.
- The second statement (`"Sid": "SecondStatement"`) lets the user list all the Amazon S3 buckets in their AWS account. The `Resource` element in this statement is `"*"` (which means "all resources"), but because policies don't grant access to resources in other accounts, the user can list only the buckets in their own AWS account. (This permission is necessary for the user to access a bucket from the AWS Management Console.)
- The third statement (`"Sid": "ThirdStatement"`) lets the user list and retrieve any object that is in a bucket called `confidential-data`, but only when the user has authenticated with a multi-factor authentication (MFA) device. The `Condition` element in the policy checks whether the user is MFA-authenticated, and if so, the user can list and retrieve objects in the bucket.

  When a policy statement contains a `Condition` element, the statement is only in effect when the `Condition` element evaluates to true. In this case, the `Condition` evaluates to true when the user is MFA-authenticated. If the user is not MFA-authenticated, this `Condition` evaluates to false. In that case, the third statement in this policy will not take effect, so the user will not have access to the `confidential-data` bucket.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "FirstStatement",
      "Effect": "Allow",
      "Action": ["iam:ChangePassword"],
      "Resource": "*"
    },
    {
      "Sid": "SecondStatement",
      "Effect": "Allow",
      "Action": "s3:ListAllMyBuckets",
      "Resource": "*"
    },
    {
      "Sid": "ThirdStatement",
      "Effect": "Allow",
      "Action": [
        "s3:List*",
        "s3:Get*"
      ],
      "Resource": [
        "arn:aws:s3:::confidential-data",
        "arn:aws:s3:::confidential-data/*"
      ],
      "Condition": {"Bool": {"aws:MultiFactorAuthPresent": "true"}}
    }
  ]
}
```

Managed Policies and Inline Policies

Using IAM, you apply permissions to IAM users, groups, and roles (which we refer to as *principal entities*) by creating policies. You can create two types of policies in IAM:

- **Managed policies** – Standalone policies that you can attach to multiple users, groups, and roles in your AWS account. You can use two types of managed policies:
  - **AWS managed policies** – Managed policies that are created and managed by AWS. If you are new to using policies, we recommend that you start by using AWS managed policies.
  - **Customer managed policies** – Managed policies that you create and manage in your AWS account. Using customer managed policies, you have more precise control over your policies than when using AWS managed policies.
- **Inline policies** – Policies that you create and manage, and that are *embedded* directly in a single user, group, or role.

Note
Another type of policy, known as a *resource-based policy*, is not discussed here. For more information about resource-based policies, see User-Based and Resource-Based Permissions (p. 172).

Generally speaking, the content of the policies is the same in all cases—each kind of policy defines a set of permissions using a common structure and a common syntax.

Note
For AWS managed policies and customer managed policies, the policy's `Version` element must be set to `2012-10-17`. For inline policies, the policy's `Version` element can be set to `2012-10-17` or to `2008-10-17`. We recommend that you set the `Version` element to `2012-10-17` for all policies.
For more information about the `Version` element, see Version (p. 210) in this guide's *Policy Element Reference*.

You can use the different types of policies together. You are not limited to using only one type.

The following sections provide more information about each of the types of policies and when to use them.

Topics

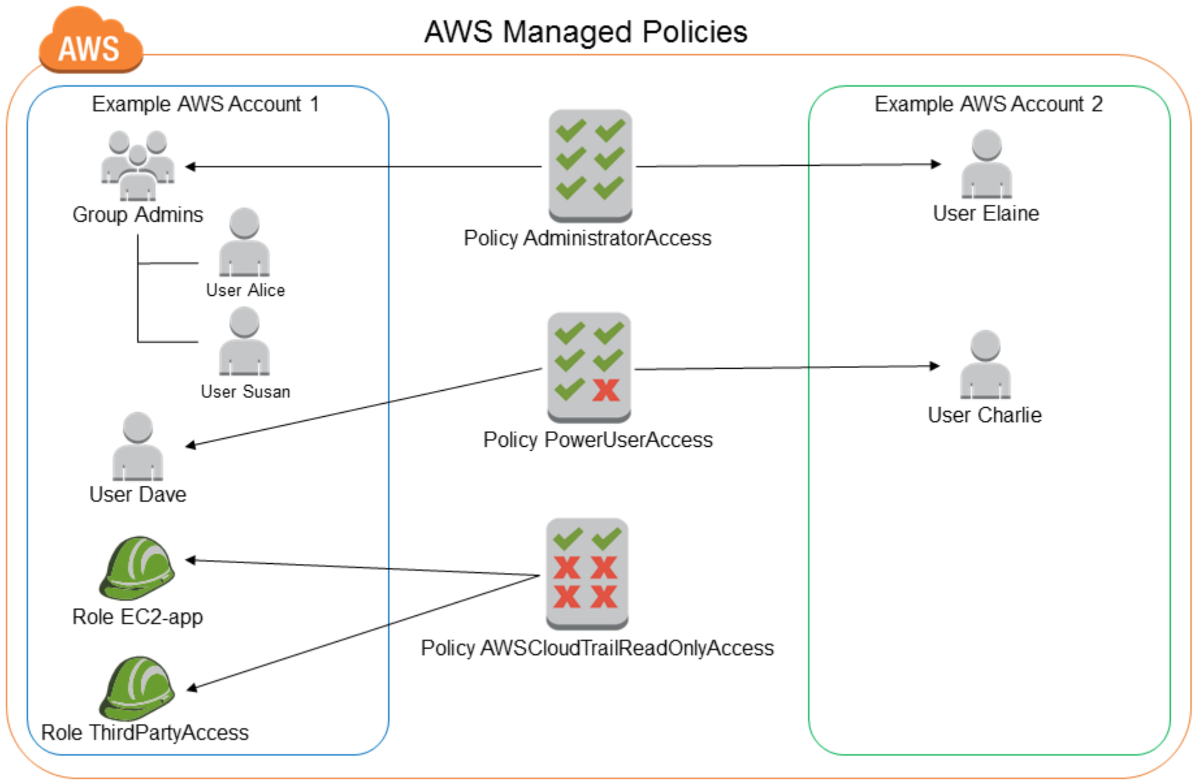
AWS Managed Policies

An *AWS managed policy* is a standalone policy that is created and administered by AWS. *Standalone policy* means that the policy has its own Amazon Resource Name (ARN) that includes the policy name.

AWS managed policies are designed to provide permissions for many common use cases. For example, there are AWS managed policies that define typical permissions for administrators (all access), for power users (all access except IAM), and for other various levels of access to AWS services. AWS managed policies make it easier for you to assign appropriate permissions to users, groups, and roles than if you had to write the policies yourself.

You cannot change the permissions defined in AWS managed policies. AWS will occasionally update the permissions defined in an AWS managed policy. When AWS does this, the update affects all principal entities (users, groups, and roles) that the policy is attached to. AWS is most likely to update an AWS managed policy when a new AWS service is launched or new APIs become available for existing services, and the policy needs to include permissions for the new service(s) or APIs. For example, the AWS managed policy called **ReadOnlyAccess** provides read-only access to all AWS services and resources. When AWS launches a new service, AWS will update the **ReadOnlyAccess** policy to add read-only permissions for the new service. The updated permissions are applied to all principal entities that the policy is attached to.

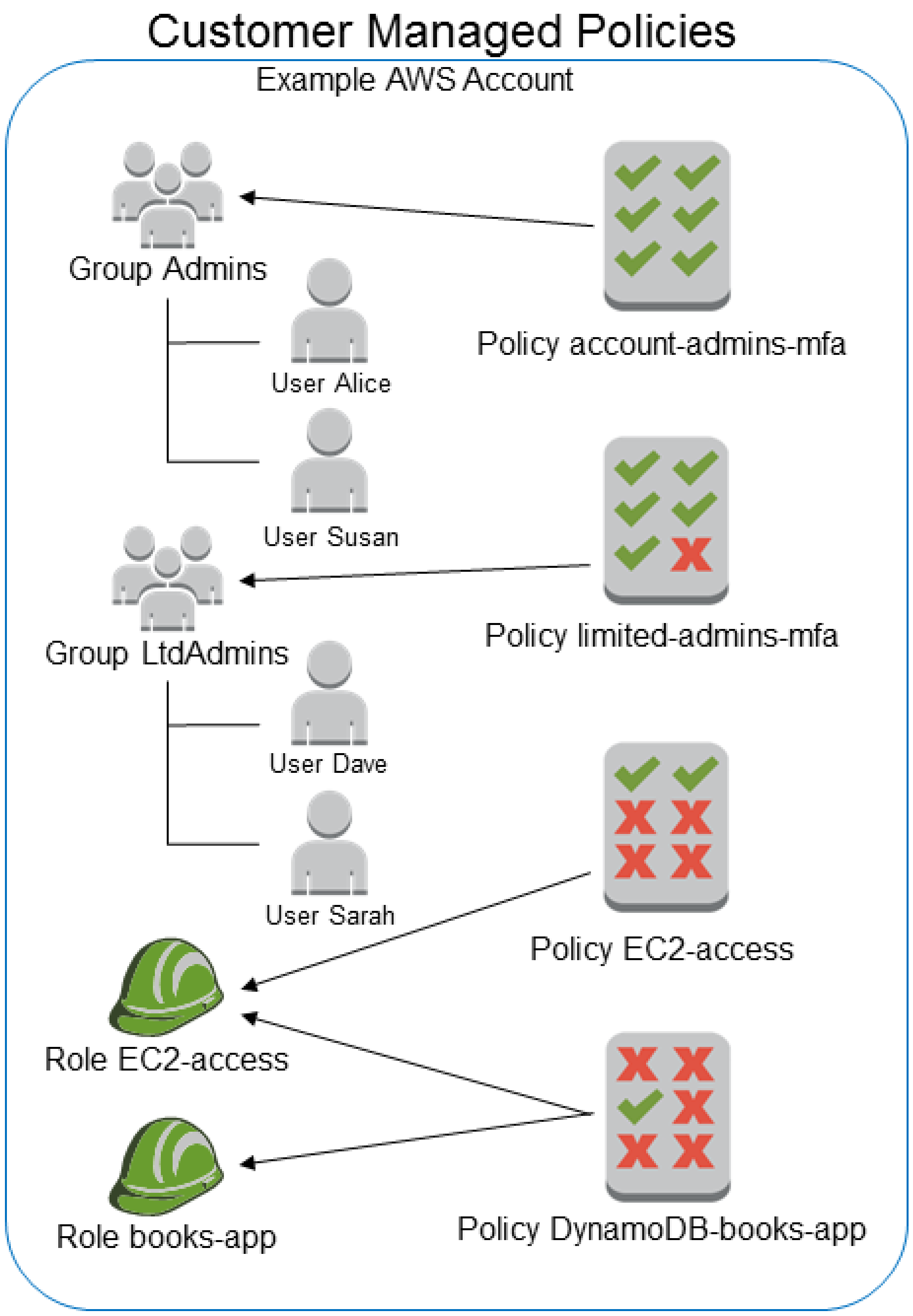
The following diagram illustrates AWS managed policies. The diagram shows three AWS managed policies: **AdministratorAccess**, **PowerUserAccess**, and **AWSCloudTrailReadOnlyAccess**. Notice that a single AWS managed policy can be attached to principal entities in different AWS accounts, and to different principal entities in a single AWS account.



Customer Managed Policies

You can create standalone policies that you administer in your own AWS account, which we refer to as a *customer managed policies*. You can then attach the policies to multiple principal entities in your AWS account. When you attach a policy to a principal entity, you give the entity the permissions that are defined in the policy.

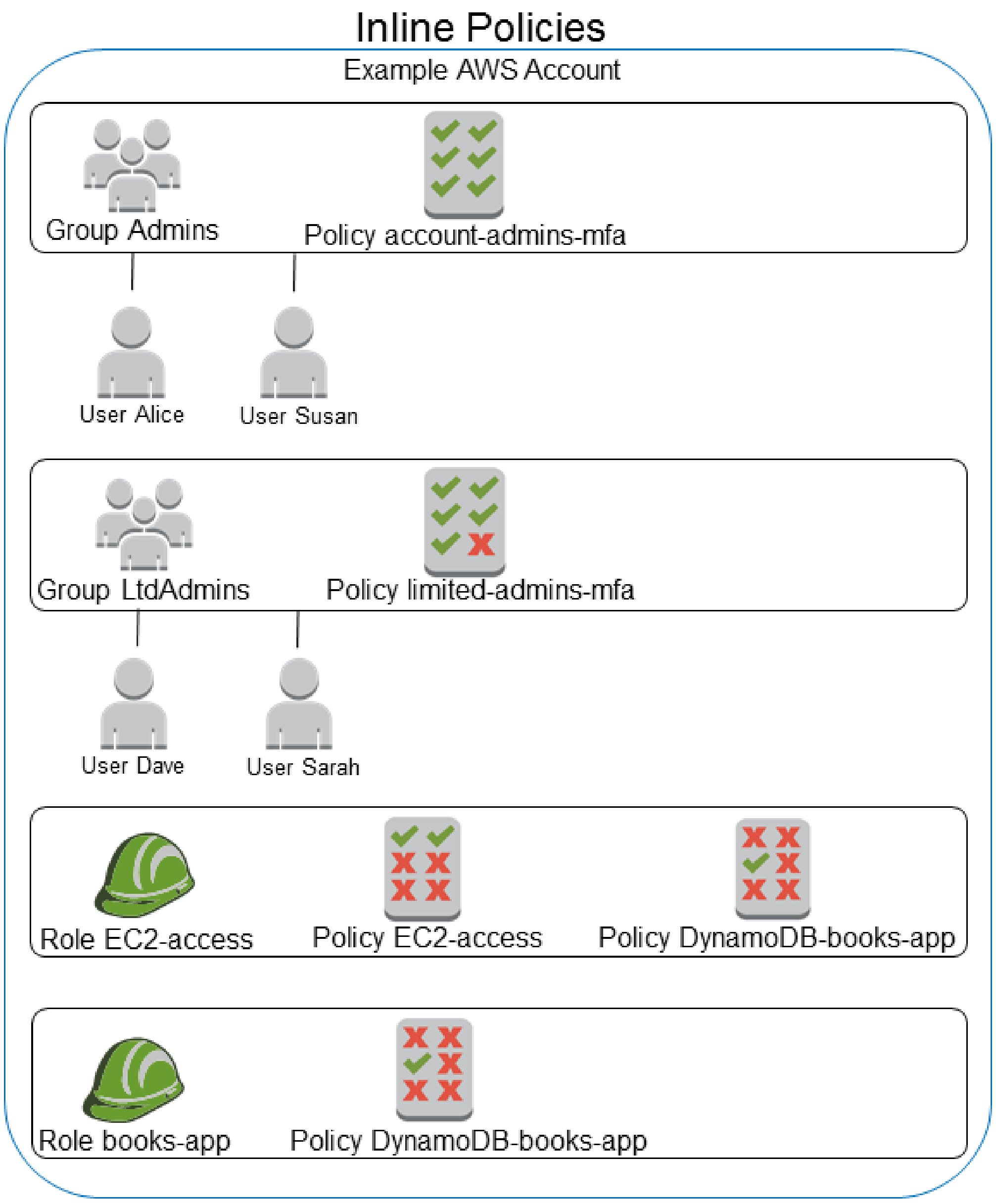
The following diagram illustrates customer managed policies. Each policy is an entity in IAM with its own Amazon Resource Name (ARN) that includes the policy name. Notice that the same policy can be attached to multiple principal entities—for example, the same **DynamoDB-books-app** policy is attached to two different IAM roles.



Inline Policies

An inline policy is a policy that's embedded in a principal entity (a user, group, or role)—that is, the policy is an inherent part of the principal entity. You can create a policy and embed it in a principal entity, either when you create the principal entity or later.

The following diagram illustrates inline policies. Each policy is an inherent part of the user, group, or role. Notice that two roles include the same policy (the **DynamoDB-books-app** policy), but they are not sharing a single policy; each role has its own copy of the policy.



Choosing Between Managed Policies and Inline Policies

The different types of policies are for different use cases. In most cases, we recommend that you use managed policies instead of inline policies.

Managed policies provide the following features:

Reusability
: A single managed policies can be attached to multiple principal entities (users, groups, and roles). In effect, you can create a library of policies that define permissions that are useful for your AWS account, and then attach these policies to principal entities as needed.

Central change management
: When you change a managed policy, the change is applied to all principal entities that the policy is attached to. For example, if you want to add permission for a new AWS API, you can update the

managed policy to add the permission. (If you're using an AWS managed policy, AWS updates to the policy.) When the policy is updated, the changes are applied to all principal entities that the policy is attached to. In contrast, to change an inline policy you must individually edit each principal entity that contains the policy—for example, if a group and a role both contain the same inline policy, you must individually edit both principal entities in order to change that policy.

Versioning and rolling back

Changes to managed policies are implemented as *versions*—making a change to a managed policy creates a new version of the policy with a new version identifier. You can use policy versions to revert a policy to an earlier version if you need to.

For more information about policy versions, see Versioning for Managed Policies (p. 183).

Delegating permissions management

You can allow users in your AWS account to attach and detach policies while maintaining control over the permissions defined in those policies. In effect, you can designate some users as full admins—that is, admins that can create, update, and delete policies. You can then designate other users as limited admins—that is, admins that can attach policies to other principal entities, but only the policies that you have allowed them to attach.

For more information about delegating permissions management, see Controlling Access to Managed Policies (p. 185).

Automatic updates for AWS managed policies

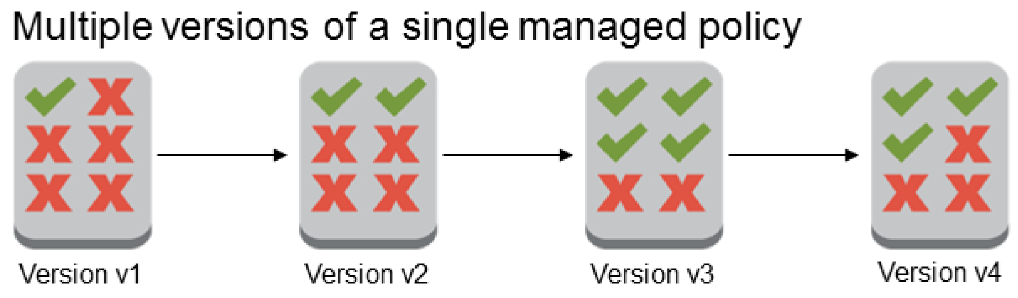
AWS maintains AWS managed policies and updates them when necessary (for example, to add permissions for new AWS services), without you having to make changes. The updates are automatically applied to the principal entities that you have attached the AWS managed policy to.

Using Inline Policies

Inline policies are useful if you want to maintain a strict one-to-one relationship between a policy and the principal entity that it's applied to. For example, you want to be sure that the permissions in a policy are not inadvertently assigned to a principal entity other than the one they're intended for. When you use an inline policy, the permissions in the policy cannot be inadvertently attached to the wrong principal entity. In addition, when you delete that principal entity using the AWS Management Console, the policies embedded in the principal entity are deleted as well, because they are part of the principal entity.

Versioning for Managed Policies

When you make changes to a customer managed policy, and when AWS makes changes to an AWS managed policy, the changed policy doesn't overwrite the existing policy. Instead, IAM creates a new *version* of the managed policy. The following diagram illustrates this.



You can use versions to track changes to a managed policy. For example, you might make a change to a managed policy and then discover that the change had unintended effects. In this case, you can roll back to a previous version of the managed policy by setting the previous version as the *default* version.

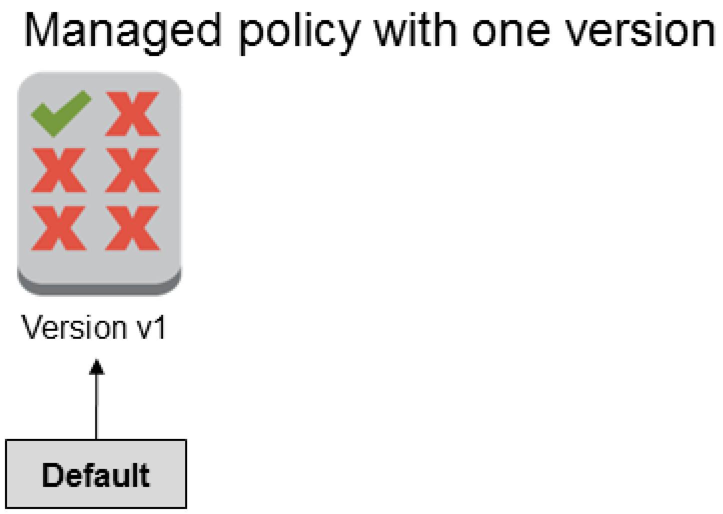
The following sections explain how you can use versioning for managed policies.

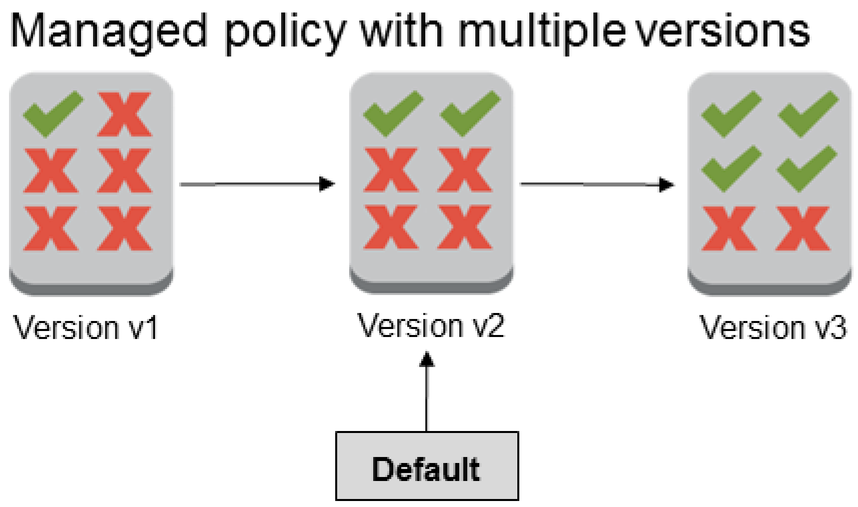
Topics

Setting the Default Version

One of the versions of a managed policy is set as the *default* version. The policy's default version is the operative version—that is, it's the version that is in effect for all of the principal entities (users, groups, and roles) that the managed policy is attached to.

When you create a customer managed policy, the policy begins with a single version identified as v1. For managed policies with only a single version, that version is automatically set as the default. For customer managed policies with more than one version, you choose which version to set as the default. For AWS managed policies, the default version is set by AWS. The following diagrams illustrate this concept.





You can set the default version of your customer managed policies, but AWS sets the default version of AWS managed policies. You set the default version of a customer managed policy using the AWS Management Console, the AWS Command Line Interface, or the IAM API.

Using Versions to Roll Back Changes

When you make changes to a customer managed policy, your changes are stored as policy versions. In some cases, you may want to roll back your changes. For example, consider the following scenario.

You create a customer managed policy that allows users to administer a particular Amazon S3 bucket using the AWS Management Console. Upon creation, your customer managed policy has only one version, identified as v1, so that version is automatically set as the default. The policy works as intended.

Later, you update the policy to add permission to administer a second Amazon S3 bucket. IAM creates a new version of the policy, identified as v2, that contains your changes. You set version v2 as the default, and a short time later your users report that they are unable to use the Amazon S3 console at all due to a permissions error. In this case, you can roll back to version v1 of the policy, which you know works as intended. To do this, you set version v1 as the default version. Your users are now able to use the Amazon S3 console to administer the original bucket.

Later, after you determine the error in version v2 of the policy, you update the policy again to add permission to administer the second Amazon S3 bucket. IAM creates another new version of the policy, identified as v3. You set version v3 as the default, and this version works as intended. At this point, you delete version v2 of the policy.

Version Limits

A managed policy can have up to five versions. If you need to make changes to a managed policy beyond five versions, you must first delete one or more existing versions. You can delete any version of the managed policy that you want, except for the default version.

When you delete a version, the version identifiers for the remaining versions do not change. As a result, version identifiers might not be sequential. For example, if you delete versions v2 and v4 of a managed policy and add two new versions, the remaining version identifiers might be v1, v3, v5, v6, and v7.

Controlling Access to Managed Policies

Managed policies give you precise control over how your users can manage policies and manage permissions for others. You can separately control who can create, update, and delete policies, and who can attach and detach policies to and from principal entities (users, groups, and roles). You can also control which policies a user can attach or detach, and to and from which entities.

A typical scenario is that you give permissions to an account administrator to create, update, and delete policies. Then, you give permissions to a team leader or other limited administrator to attach and detach these policies to and from principal entities that the limited administrator manages.

Topics

Controlling Permissions for Creating, Updating, and Deleting Customer Managed Policies

You can use IAM policies (p. 174) to control who is allowed to create, update, and delete customer managed policies in your AWS account. The following list contains APIs that pertain directly to creating, updating, and deleting policies or policy versions:

- CreatePolicy
- CreatePolicyVersion
- DeletePolicy
- DeletePolicyVersion
- SetDefaultPolicyVersion

The APIs in the preceding list correspond to actions that you can allow or deny—that is, permissions that you can grant—using an IAM policy.

The following example shows a policy that allows a user to create, update (that is, create a new policy version), delete, and set a default version for all customer managed policies in the AWS account. The example policy also allows the user to list policies and get policies.

Example policy that allows creating, updating, deleting, listing, getting, and setting the default version for all policies

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "iam:CreatePolicy",
      "iam:CreatePolicyVersion",
      "iam:DeletePolicy",
      "iam:DeletePolicyVersion",
      "iam:GetPolicy",
      "iam:GetPolicyVersion",
      "iam:ListPolicies",
      "iam:ListPolicyVersions",
      "iam:SetDefaultPolicyVersion"
    ],
    "Resource": "*"
  }
}
```

You can create policies that limit the use of these APIs to affect only the managed policies that you specify. For example, you might want to allow a user to set the default version and delete policy versions, but only for specific customer managed policies. You do this by specifying the policy ARN in the `Resource` element of the policy that grants these permissions.

The following example shows a policy that allows a user to delete policy versions and set the default version, but only for the customer managed policies that include the path /TEAM-A/. The customer managed policy ARN is specified in the `Resource` element of the policy (in this example the ARN includes a path and a wildcard and thus matches all customer managed policies that include the path /TEAM-A/).

For more information about using paths in the names of customer managed policies, see Friendly Names and Paths (p. 9).

Example policy that allows deleting policy versions and setting the default version for only specific policies

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "iam:DeletePolicyVersion",
      "iam:SetDefaultPolicyVersion"
    ],
    "Resource": "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:policy/TEAM-A/*"
  }
}
```

Controlling Permissions for Attaching and Detaching Managed Policies

You can also use IAM policies to allow users to work with only specific managed policies—in effect, you can control which permissions a user is allowed to grant to other principal entities.

The following list shows APIs that pertain directly to attaching and detaching managed policies to and from principal entities:

- AttachGroupPolicy
- AttachRolePolicy
- AttachUserPolicy
- DetachGroupPolicy
- DetachRolePolicy
- DetachUserPolicy

You can create policies that limit the use of these APIs to affect only the specific managed policies and/or principal entities that you specify. For example, you might want to allow a user to attach managed policies, but only the managed policies that you specify. Or, you might want to allow a user to attach managed policies, but only to the principal entities that you specify.

The following example policy allows a user to attach managed policies to only the groups and roles that include the path /TEAM-A/. The group and role ARNs are specified in the `Resource` element of the policy (in this example the ARNs include a path and a wildcard and thus match all groups and roles that include the path /TEAM-A/).

Example policy that allows attaching managed policies to only specific groups or roles

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "iam:AttachGroupPolicy",
      "iam:AttachRolePolicy"
    ],
    "Resource": [
      "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:group/TEAM-A/*",
      "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:role/TEAM-A/*"
    ]
  }
}
```

You can further limit the actions in the preceding example to affect only specific policies—that is, you can control which permissions a user is allowed to attach to other principal entities—by adding a condition to the policy.

In the following example, the condition ensures that the `AttachGroupPolicy` and `AttachRolePolicy` permissions are allowed only when the policy being attached matches one of the specified policies. The condition uses the `iam:PolicyArn` condition key (p. 219) to determine which policy or policies are allowed to be attached. The following example policy expands on the previous example by allowing a user to attach only the managed policies that include the path /TEAM-A/ to only the groups and roles that include the path /TEAM-A/.

Example policy that allows attaching only specific managed policies to only specific groups or roles

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "iam:AttachGroupPolicy",
      "iam:AttachRolePolicy"
    ],
    "Resource": [
      "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:group/TEAM-A/*",
      "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:role/TEAM-A/*"
    ],
    "Condition": {"ArnLike":
      {"iam:PolicyArn": "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:policy/TEAM-
A/*"}
    }
  }
}
```

Specifying the Amazon Resource Name (ARN) for Managed Policies

To control access to specific managed policies, you use the Amazon Resource Name (ARN) (p. 9) of the managed policy. In some cases you use the ARN of the managed policy in the `Resource` element of a policy, and in other cases you use the ARN of the managed policy in the `Condition` element of a policy.

The following sections explain when to use each.

Using the `Resource` Element to Control Access to Actions That Affect the Managed Policy

To control access to specific managed policies for actions that affect the managed policy, you specify the ARN of the managed policy in the `Resource` element of a policy.

The following list contains IAM actions (APIs) that affect a managed policy:

- CreatePolicy
- CreatePolicyVersion
- DeletePolicy
- DeletePolicyVersion
- GetPolicy
- GetPolicyVersion
- ListEntitiesForPolicy
- ListPolicyVersions
- SetDefaultPolicyVersion

You can limit the use of these actions to affect only the managed policies that you specify. You do this by specifying the policy ARN in the `Resource` element of the policy that grants these permissions. For example, to specify the ARN of a customer managed policy:

```
"Resource": "arn:aws:iam::123456789012:policy/POLICY-NAME"
```

You can also specify the ARN of an AWS managed policy in a policy's `Resource` element. The ARN of an AWS managed policy uses the special alias `aws` in the policy ARN instead of an account ID, as in this example:

```
"Resource": "arn:aws:iam::aws:policy/AmazonEC2FullAccess"
```

Using the `Condition` Element to Control Access to Actions That Affect the Principal Entity (User, Group, or Role)

To control access to specific managed policies for actions that involve a managed policy but that affect a principal entity (user, group, or role), you specify the ARN of the managed policy in the `Condition` element of a policy. In this case, the `Resource` element of a policy is used to specify the ARN of the principal entity that is affected.

The following list contains IAM actions (APIs) that involve a managed policy but that affect a principal entity:

- AttachGroupPolicy
- AttachRolePolicy
- AttachUserPolicy
- DetachGroupPolicy
- DetachRolePolicy
- DetachUserPolicy

- ListAttachedGroupPolicies
- ListAttachedRolePolicies
- ListAttachedUserPolicies

You can limit the use of these actions to involve only the managed policies that you specify. You do this by specifying the policy ARN in the `Condition` element of the policy that grants these permissions. For example, to specify the ARN of a customer managed policy:

```
"Condition": {"ArnEquals":
  {"iam:PolicyArn": "arn:aws:iam::123456789012:policy/POLICY-NAME"}
}
```

You can also specify the ARN of an AWS managed policy in a policy's `Condition` element. The ARN of an AWS managed policy uses the special alias `aws` in the policy ARN instead of an account ID, as in this example:

```
"Condition": {"ArnEquals":
  {"iam:PolicyArn": "arn:aws:iam::aws:policy/AmazonEC2FullAccess"}
}
```

You can use the `ArnLike` or `ArnEquals` condition types. For more information about `ArnLike` and `ArnEquals`, see Amazon Resource Name (ARN) (p. 229) in the *Condition Types* section of the *Policy Element Reference*.

Tutorial: Create and Attach Your First Customer Managed Policy

This tutorial shows you how to get started with managed policies (p. 179). You'll create a customer managed policy (p. 180) and attach it to IAM users in your AWS account using the AWS Management Console. To complete this tutorial, you must already have created the following:

- An AWS account with one or more IAM users.
- The user name and password for an IAM user with administrative permissions—that is, permissions to perform all IAM actions.

  To learn how to create an IAM user with administrative permissions, see Creating an Administrators Group Using the Console (p. 20).

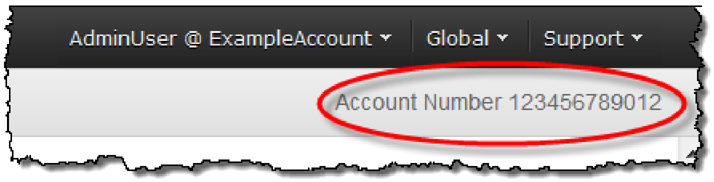
In this tutorial you use the AWS Management Console to create a customer managed policy and then attach that policy to all IAM users in your AWS account. The policy you create allows IAM users to change (but not delete) their own passwords and manage (create, update, or delete) their own access keys using the AWS Management Console.

Topics

Part 1: Locate Your AWS Account ID

In this part of the tutorial you locate and note the 12-digit account ID for your AWS account. If you already know your 12-digit account ID, you can move on to Part 2 (p. 191) of this tutorial.

1. Sign in to the AWS Management Console.
2. In the navigation bar, click **Support** and then click **Support Center**.
3. In the content pane, in the Support Center title bar, find your account ID. Your account ID is the 12-digit number following the text **Account Number**.



Copy your account ID and save it where you can easily find it again for the next part of the tutorial. After you have saved your account ID, you are ready to move on to Part 2 (p. 191).

Part 2: Create the Policy

Next, you create a customer managed policy that allows IAM users to change their own passwords and manage their own access keys using the AWS Management Console.

1. Open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, click **Policies**.
3. In the content pane, click **Create Policy**.
4. Next to **Create Your Own Policy**, click **Select**.
5. In the **Policy Name** field, enter **`UsersManageOwnCredentials`**.
6. In the **Policy Document** field, paste the following policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iam:ChangePassword",
        "iam:CreateAccessKey",
        "iam:DeleteAccessKey",
        "iam:ListAccessKeys",
        "iam:UpdateAccessKey",
        "iam:GetLoginProfile",
        "iam:UpdateLoginProfile"
      ],
      "Resource": "arn:aws:iam::YOUR-ACCOUNT-ID-HERE:user/${aws:username}"
    },
    {
      "Effect": "Allow",
      "Action": "iam:ListUsers",
      "Resource": "*"
    }
  ]
}
```

After pasting the policy, replace `YOUR-ACCOUNT-ID-HERE` with your own 12-digit AWS account ID. This is the account ID you noted in Part 1 (p. 191) of this tutorial.

7. Click **Validate Policy** and ensure that no errors display in a red box at the top of the screen. Correct any that are reported.

   Note
   If **Use autoformatting** is selected, the policy is reformatted whenever you open a policy or click **Validate Policy**.

8. After you paste your own account ID into the policy, click **Create Policy**.

You have successfully created a policy that allows IAM users to change their own passwords and manage their own access keys using the AWS Management Console. Next, you attach this policy to all of your IAM users to give them the permissions defined in this policy.

Part 3: Attach the Policy

Next you attach the policy named UsersManageOwnCredentials to every IAM user in your AWS account.

1. In the IAM console, in the navigation pane, click **Policies**.
2. At the top of the policy list, in the search box, type **`UsersManageOwnCredentials`**, then click the **UsersManageOwnCredentials** policy in the list.
3. Scroll down to the **Attached Entities** section, then click **Attach**.
4. Click **All Types** to display the filter menu, then click **Users**.
5. In the top row of the list, select the check box to select all IAM users in your AWS account.
6. Click **Attach Policy**.

You have attached the policy to all IAM users in your account, which means they now have permission to change their own passwords and manage their own access keys using the AWS Management Console.

Managing Policies

This section describes how to create and manage all types of IAM policies (managed policies and inline policies).

For more information about the different types of IAM policies, see Managed Policies and Inline Policies (p. 179).

For general information about IAM policies, see Overview of IAM Policies (p. 174).

To add permissions to an IAM principal entity—that is, an IAM user, group, or role—you create a policy and then attach the policy to the principal entity. You can attach multiple policies to a principal entity, and each policy can contain multiple permissions.

For information about how permissions are evaluated when multiple policies are in effect for a given IAM principal entity, see IAM Policy Evaluation Logic (p. 243).

For information about policy size limitations and other quotas, see Limitations on IAM Entities (p. 13).

Topics

Managing Managed Policies

This section describes how to work with AWS managed policies, and how to create and work with customer managed policies, that is, managed policies that you create yourself. You can manage and create managed policies using the AWS Management Console, the AWS Command Line Interface (AWS CLI), or the IAM API.

For more information about the different types of IAM policies, see Managed Policies and Inline Policies (p. 179).

For general information about IAM policies, see Overview of IAM Policies (p. 174).

For information about policy size limitations and other quotas, see Limitations on IAM Entities (p. 13).

Topics

Managing Managed Policies Using the AWS Management Console

This section describes how to manage managed policies (p. 179) using the AWS Management Console.

For information about managing managed policies using the AWS Command Line Interface (AWS CLI) or the IAM API, see Managing Managed Policies Using the AWS CLI or the IAM API (p. 196).

Topics

Attaching Managed Policies

You can attach a managed policy to a principal entity (a user, group, or role) to apply the permissions in the policy to the principal entity. You can attach up to two managed policies to each principal entity.

To attach a managed policy using the AWS Management Console

1. Sign in to the AWS Management Console and open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, click **Policies**.
3. In the list of policies, select the check box next to the name of the customer managed policy to attach. You can use the **Filter** menu and the **Search** box to filter the list of policies.
4. Click **Policy Actions**, then click **Attach**.
5. Select the principal entities to attach the policy to. You can use the **Filter** menu and the **Search** box to filter the list of principal entities. After selecting the principal entities to attach the policy to, click **Attach Policy**.

Detaching Managed Policies

You can detach a managed policy from a principal entity (a user, group, or role) to remove the permissions in the policy from the principal entity.

To detach a managed policy using the AWS Management Console

1. Sign in to the AWS Management Console and open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, click **Policies**.
3. In the list of policies, select the check box next to the name of the customer managed policy to detach. You can use the **Filter** menu and the **Search** box to filter the list of policies.
4. Click **Policy Actions**, then click **Detach**.
5. Select the principal entities to detach the policy from. You can use the **Filter** menu and the **Search** box to filter the list of principal entities. After selecting the principal entities to detach the policy from, click **Detach Policy**.

Creating Customer Managed Policies

You can create customer managed policies to define sets of permissions to attach to principal entities (users, groups, and roles) in your AWS account. For more information about customer managed policies, see Managed Policies and Inline Policies (p. 179)

To create a managed policy using the AWS Management Console

1. Sign in to the AWS Management Console and open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, click **Policies**, then click **Create Policy**.
3. Click the **Select** button that corresponds to the way you want to create your policy.
   - **Copy an AWS Managed Policy**. See a list of all existing policies and then click **Select** next to the one you want to copy.
   - **Policy Generator**. Build a policy by selecting elements from lists of available options. Select the appropriate **Effect**, **AWS Service**, and **Actions** options, enter the Amazon Resource Name ARN (if applicable), and add any conditions you want to include. Then click **Add Statement**. You can add as many statements as you want to the policy. When you are finished adding statements, click **Next Step**.
   - **Create Your Own Policy**. Type a **Policy Name** in the space provided. For **Policy Document**, type or paste a policy document into the editor.
4. In the editor, make any customizations that you need to tailor the policy to your environment.
5. After you complete your changes, click **Validate Policy** and ensure that no errors display in a red box at the top of the screen. Correct any errors that are reported.

   Note
   If **Use autoformatting** is selected, the policy is reformatted whenever you open a policy or click **Validate Policy**.
6. Click **Create Policy** to save your new policy.

Editing Customer Managed Policies

You edit customer managed policies to change the permissions that are defined in the policy. You cannot edit AWS managed policies.

To edit a customer managed policy using the AWS Management Console

1. Sign in to the AWS Management Console and open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, click **Policies**.
3. In the list of policies, click the policy name of the policy to edit. You can use the **Filter** menu and the **Search** box to filter the list of policies.
4. On the right side of the content pane, on the **Policy Document** row, click **Edit**, then edit the policy document.
5. After you complete your changes, click **Validate Policy** and ensure that no errors display in a red box at the top of the screen. Correct any that are reported.

   Note
   If **Use autoformatting** is selected, then the policy is reformatted whenever you open a policy or click **Validate Policy**.
6. When you are finished editing the policy, decide whether you want to immediately apply your changes to all principal entities (users, groups, and roles) that this policy is attached to:
   - To immediately apply your changes to all attached entities, select **Save as default version**.
   - To save your changes without affecting the currently attached entities, clear **Save as default version**.
7. Click **Save**.

Setting the Default Version of Customer Managed Policies

You can specify the default version of a customer managed policy to make that version the one that is in effect for every principal entity (user, group, and role) that the policy is attached to. You cannot set the default version for an AWS managed policy.

You can set the default version of a customer managed policy when you edit the policy. To set the default version while editing the policy, see Editing Customer Managed Policies (p. 194). To set the default version of a customer managed policy independently of editing the policy, see the following procedure.

To set the default version of a customer managed policy using the AWS Management Console

1. Sign in to the AWS Management Console and open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, click **Policies**.
3. In the list of policies, click the policy name of the policy to set the default version of. You can use the **Filter** menu and the **Search** box to filter the list of policies.
4. Scroll down to the **Policy Versions** section. Select the check box next to the version that you want to set as the default version, then click **Set as Default**.

Deleting Versions of Customer Managed Policies

You can delete a version of a customer managed policy to ensure that version can never be set as the default version of the policy—that is, to ensure that version can never be attached to any entities (users, groups, and roles) in your AWS account. You cannot delete versions of AWS managed policies.

To delete a version of a customer managed policy using the AWS Management Console

1. Sign in to the AWS Management Console and open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, click **Policies**.
3. Click the name of the customer managed policy to delete a version of. You can use the **Filter** menu and the **Search** box to filter the list of policies.
4. Scroll down to the **Policy Versions** section. Select the check box next to the version to delete, then click **Delete**.
5. Confirm that you want to delete the version, then click **Delete**.

Deleting Customer Managed Policies

You can delete a customer managed policy to remove it from your AWS account. You cannot delete AWS managed policies.

To delete a customer managed policy using the AWS Management Console

1. Sign in to the AWS Management Console and open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, click **Policies**.
3. Select the check box next to the customer managed policy to delete. You can use the **Filter** menu and the **Search** box to filter the list of policies.
4. Click **Policy Actions**, then click **Delete**.
5. Confirm that you want to delete the policy, then click **Delete**.

Managing Managed Policies Using the AWS CLI or the IAM API

This section describes how to manage managed policies (p. 179) using the AWS Command Line Interface (AWS CLI) or the IAM API. Information in this section applies to both AWS managed policies and customer managed policies, that is, managed policies that you create.

For information about managing managed policies using the AWS Management Console, see Managing Managed Policies Using the AWS Management Console (p. 193).

To list managed policies

- AWS CLI: list-policies
- IAM API: ListPolicies

To retrieve detailed information about a managed policy

- AWS CLI: get-policy
- IAM API: GetPolicy

To list the versions of a managed policy

- AWS CLI: list-policy-versions
- IAM API: ListPolicyVersions

To retrieve detailed information about a version of a managed policy, including the policy document

- AWS CLI: get-policy-version

- IAM API: GetPolicyVersion

To list the principal entities (users, groups, and roles) attached to a managed policy

- AWS CLI: list-entities-for-policy
- IAM API: ListEntitiesForPolicy

To list the managed policies attached to a principal entity (a user, group, or role)

- AWS CLI:
  - list-attached-group-policies
  - list-attached-role-policies
  - list-attached-user-policies
- IAM API:
  - ListAttachedGroupPolicies
  - ListAttachedRolePolicies
  - ListAttachedUserPolicies

To attach a managed policy to a group, role, or user

- AWS CLI:
  - attach-group-policy
  - attach-role-policy
  - attach-user-policy
- IAM API:
  - AttachGroupPolicy
  - AttachRolePolicy
  - AttachUserPolicy

To detach a managed policy from a group, role, or user

- AWS CLI:
  - detach-group-policy
  - detach-role-policy
  - detach-user-policy
- IAM API:
  - DetachGroupPolicy
  - DetachRolePolicy
  - DetachUserPolicy

To create a customer managed policy

- AWS CLI: create-policy
- IAM API: CreatePolicy

To edit a customer managed policy

- AWS CLI: create-policy-version
- IAM API: CreatePolicyVersion

To set the default version of a customer managed policy

- AWS CLI: set-default-policy-version
- IAM API: SetDefaultPolicyVersion

To delete a version of a customer managed policy

- AWS CLI: delete-policy-version
- IAM API: DeletePolicyVersion

To delete a customer managed policy

- AWS CLI: delete-policy
- IAM API: DeletePolicy

Managing Inline Policies

This section describes how to create and manage inline policies (p. 179).

For information about managing *managed policies*, see Managing Managed Policies (p. 193).

Topics

Managing Inline Policies using the AWS Management Console

To create an inline policy and embed it in a group, user, or role

1. Sign in to the AWS Management Console and open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, click **Groups**, **Users**, or **Roles**.
3. In the list, click the name of the group, user, or role to embed a policy in.
4. Scroll down to the **Permissions** section and, if necessary, expand the **Inline Policies** section.
5. Click **Create Another Policy** if in Groups, **Create User Policy** if in Users, or **Create Role Policy** if in Roles.
6. Click **Policy Generator** or **Custom Policy**, and then click **Select**.
7. Do one of the following:

   - If you chose **Custom Policy**, specify a name for the policy and create your policy document.
   - If you are using the policy generator to create your policy, select the appropriate **Effect**, **AWS Service**, and **Actions** options, enter the Amazon Resource Name ARN (if applicable), and add any conditions you want to include. Then click **Add Statement**. You can add as many statements as you want to the policy. When you are finished adding statements, click **Next Step**.

8. Click **Validate Policy** and ensure that no errors display in a red box at the top of the screen. Correct any that are reported.

   Note
   If **Use autoformatting** is selected, the policy is reformatted whenever you open a policy or click **Validate Policy**.

9. When you are satisfied with the policy, click **Apply Policy**.

To view a policy or a list of all policies associated with a user, group, or role

- In the navigation pane, click **Users**, **Groups**, or **Roles**, click the name of the entity to view, then scroll down to the **Permissions** section.

To edit or delete an inline policy for a group, user, or role

1. In the navigation pane, click **Groups**, **Users**, or **Roles**.
2. Click the name of the group, user, or role with the policy you want to modify, and then scroll down to the **Permissions** section.
3. To edit an inline policy, click **Edit Policy**.
4. To delete an inline policy, click **Remove Policy**.

Managing Inline Policies using the AWS CLI or the IAM API

To list all inline policies that are embedded in a principal entity (user, group, or role)

- AWS CLI:
  - list-group-policies
  - list-role-policies
  - list-user-policies
- IAM API:
  - ListGroupPolicies
  - ListRolePolicies
  - ListUserPolicies

To retrieve an inline policy document that is embedded in a principal entity (user, group, or role)

- AWS CLI:
  - get-group-policy
  - get-role-policy
  - get-user-policy
- IAM API:
  - GetGroupPolicy
  - GetRolePolicy
  - GetUserPolicy

To embed an inline policy in a principal entity (user, group, or role)

- AWS CLI:
  - put-group-policy
  - put-role-policy
  - put-user-policy
- IAM API:
  - PutGroupPolicy
  - PutRolePolicy
  - PutUserPolicy

To delete an inline policy that is embedded in a principal entity (user, group, or role)

- AWS CLI:
  - delete-group-policy
  - delete-role-policy
  - delete-user-policy
- IAM API:
  - DeleteGroupPolicy
  - DeleteRolePolicy
  - DeleteUserPolicy

Testing IAM Policies

This section describes how to test IAM policies that are attached to users and groups in your AWS account. You can test policies to ensure that they have the intended effect, and to identify which specific statement in a policy results in allowing or denying access to a particular resource or action.

To test a policy, you use the IAM Policy Simulator. When you work with an IAM user or group in the IAM console, you can launch the IAM Policy Simulator directly from the IAM console to test the policies that are attached to the user or group.

> **Note**
> The IAM Policy Simulator does not currently support resource-based policies or policies attached to roles.

To test a policy attached to a user or group

1. Sign in to the AWS Management Console and open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, click **Groups** or **Users**.
3. Click the name of the group or user that you want to test a policy for, then scroll down to the **Permissions** section.
4. In the **Inline Policies** section, locate the policy you want to test, then click **Simulate Policy**.

   > **Note**
   > You cannot test a managed policy by clicking a link in the IAM console. To test a managed policy, first open the IAM Policy Simulator directly, then select the user or group that the managed policy is attached to.

   The IAM Policy Simulator opens in a new window and displays the selected policy in the **Policies** pane.
5. In the content pane, select the service to test the policy for, then select the action or actions to test.
6. *(Optional)* If you want to simulate a request against a specific resource, click to expand **Simulation Settings** and then enter the Amazon Resource Name (ARN) for a resource. For help formatting an ARN, select a service from the **Resource Name Format** list.

   > **Note**
   > The IAM Policy Simulator currently supports only one resource per simulation.
7. If the policy you are testing contains condition keys or variables, click to expand **Simulation Settings** and then enter a value for each key or variable.

   > **Note**
   > If you leave the field empty for a condition key or variable, that key or variable will be ignored during simulation. In some cases, this will result in an error and the simulation will fail to run.

8. Click **Run Simulation**.

   The policy simulator displays the results.
9. *(Optional)* To see which statement in a policy explicitly allowed or denied an action:

   a. Click **List** next to the result to show the policy name.
   b. Click **Show statement** to show the relevant statement in the policy.

   Note
   If an action is *implicitly* denied—that is, if the action is denied because it is not explicitly allowed—the **List** and **Show statement** options are not displayed.

For more information about the IAM Policy Simulator, see IAM Policy Simulator in the *Using IAM Policy Simulator Guide*.

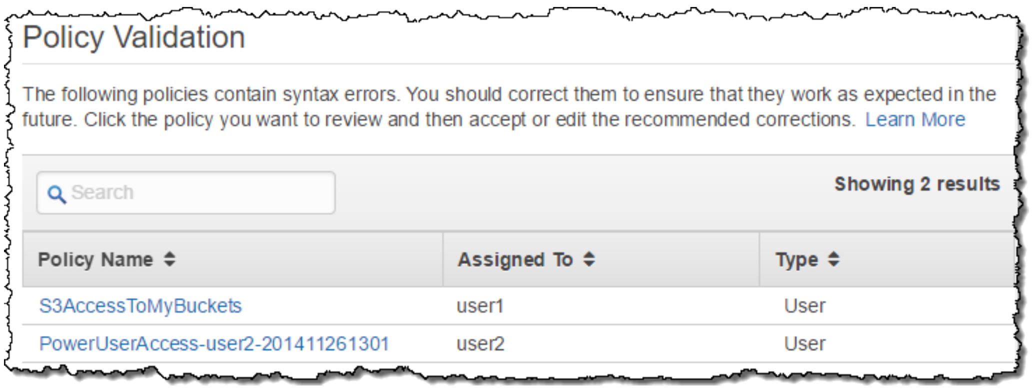
Using Policy Validator

Policy Validator automatically examines your existing IAM access control policies to ensure that they comply with the IAM policy grammar. A policy is a JSON document written using the IAM policy grammar. It defines access permissions for the AWS user, group, or role you attach the policy to. If the Policy Validator determines that a policy is not in compliance with the grammar, it prompts you to fix the policy. Policy Validator is only available if you have non-compliant policies.

Important
Starting in March 2015, you cannot save any new or updated policies that do not comply with the policy syntax. If a policy fails validation, it cannot be saved until the error is corrected. Although existing policies with these errors continue to function as they did, you cannot edit and save them without fixing any errors in the syntax.

Identifying and fixing non-compliant access control policies to comply with the policy grammar

1. Sign in to the IAM console. If you have any non-compliant policies, a yellow banner titled **Fix policy syntax** appears at the top of the console screen. If this banner does not appear, then all of your policies are in good shape and you can stop right now.
2. Click the **Fix Now** link.
3. A list of the non-compliant policies appears. Select the policy that you want to correct by clicking the policy name.


Policy Validation

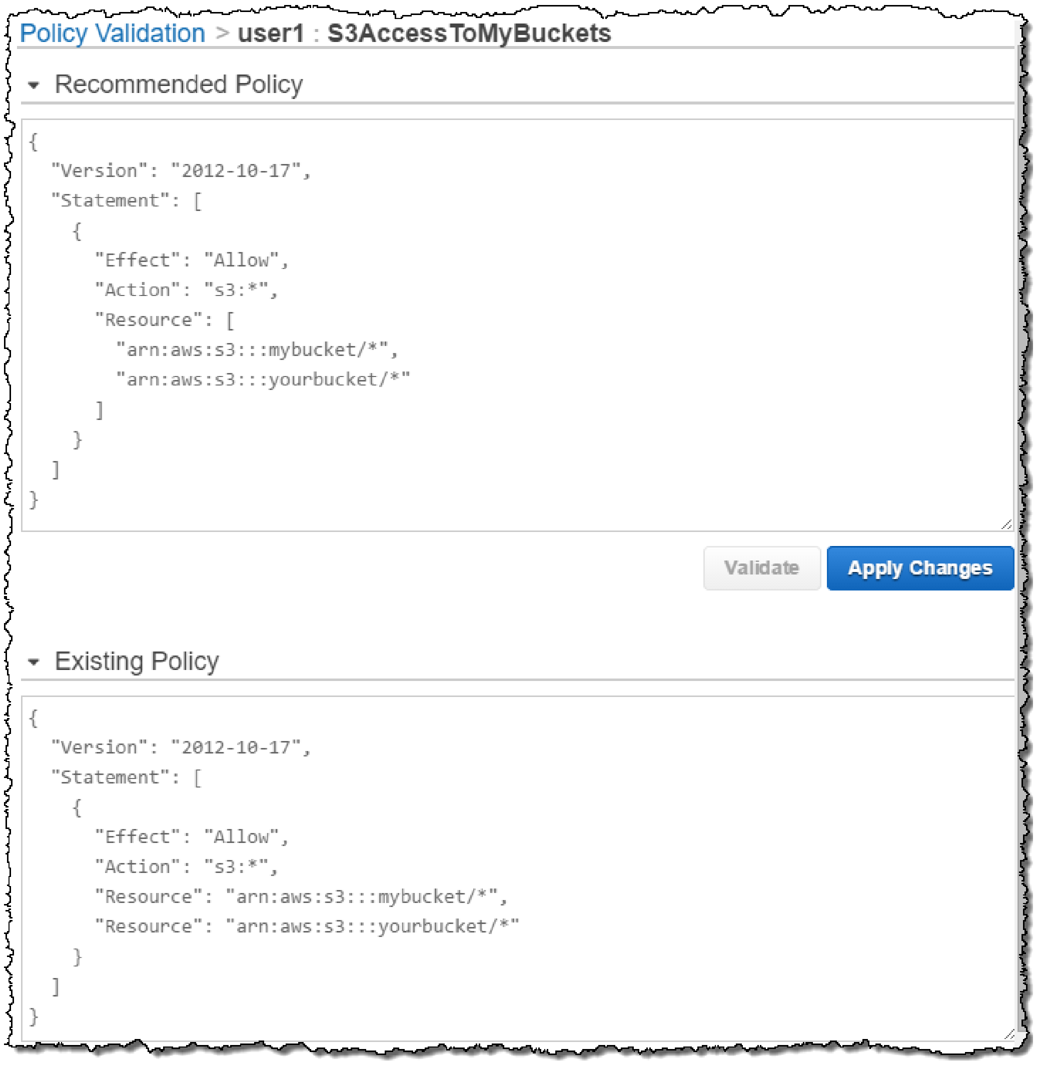
The following policies contain syntax errors. You should correct them to ensure that they work as expected in the future. Click the policy you want to review and then accept or edit the recommended corrections. Learn More

Showing 2 results

Policy Name	Assigned To	Type
S3AccessToMyBuckets	user1	User
PowerUserAccess-user2-201411261301	user2	User

4. A screen similar to the following appears, showing the recommended changes to your policy at the top of the page in an editing window and the original version at the bottom. In the following example,

the policy engine recommends changing two separate Resource elements (not valid) into a single Resource array element with two items in it (valid). For more information about the policy rules, see the AWS IAM Policy Reference.



```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:*",
      "Resource": [
        "arn:aws:s3:::mybucket/*",
        "arn:aws:s3:::yourbucket/*"
      ]
    }
  ]
}
```

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:*",
      "Resource": "arn:aws:s3:::mybucket/*",
      "Resource": "arn:aws:s3:::yourbucket/*"
    }
  ]
}
```

5. Do one of the following:

   - If you want to accept the recommended changes, click **Apply Changes**.
   - If you want to alter the policy further, you can edit directly in the top edit box. If you make any changes, the **Validate** button is enabled. When you are done, check the syntax against the rules by clicking **Validate**. If Policy Validator confirms that your edited policy is OK, click **Apply Changes**. If errors are reported, continue to edit the policy until it passes validation and then click **Apply Changes**.

6. You are returned to the list of non-compliant policies, if any. Repeat the procedure for each until you have fixed all of your policies.

You can edit any of your policies on your own at any time, without using the Policy Validator. If you fix any compliance issues then they are automatically removed from the list of non-compliant policies.

Example Policies for Administering AWS Resources

This section shows some examples of policies that control access to resources in AWS services. For examples of policies that show how to let IAM users administer IAM resources—for example, to allow users to change their own access keys—see Example Policies for Administering IAM Resources (p. 97).

Topics

Allow Users to Access a Specific Bucket in Amazon S3

The following policy can be attached to an IAM user or an IAM group. It gives the user or group members access to a specific bucket and to all the objects in it. Users can also list all the buckets in the account (the `s3:ListAllMyBuckets` permission); this permission lets the user perform the other actions using the Amazon S3 console.

Note
In the following policy, you need to replace `EXAMPLE-BUCKET-NAME` with the name of your bucket.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:ListAllMyBuckets",
      "Resource": "arn:aws:s3:::*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket",
        "s3:GetBucketLocation"
      ],
      "Resource": "arn:aws:s3:::EXAMPLE-BUCKET-NAME"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:DeleteObject"
      ],
      "Resource": "arn:aws:s3:::EXAMPLE-BUCKET-NAME/*"
    }
  ]
}
```

Allow Users to Access a Personal "Home Directory" in Amazon S3

The following policy can be attached to an IAM group. It lets an IAM user in that group use the AWS Management Console access a "home directory" in Amazon S3 that matches their user name. For example, user `Bob` can access `s3://BUCKET-NAME/home/Bob/`, but he cannot access `s3://BUCKET-NAME/home/Alice/`. Inside his or her "home directory," each user can perform all Amazon S3 actions, such as `GetObject`, `ListBucket`, `PutObject`, and `DeleteObject`.

> **Note**
> In the following policy, you need to replace BUCKET-NAME with the name of a bucket under which you have created a `home` folder and folders for individual users.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListAllMyBuckets",
        "s3:GetBucketLocation"
      ],
      "Resource": "arn:aws:s3:::*"
    },
    {
      "Effect": "Allow",
      "Action": "s3:ListBucket",
      "Resource": "arn:aws:s3:::BUCKET-NAME",
      "Condition": {"StringLike": {"s3:prefix": [
        "",
        "home/",
        "home/${aws:username}/"
      ]}}
    },
    {
      "Effect": "Allow",
      "Action": "s3:*",
      "Resource": [
        "arn:aws:s3:::BUCKET-NAME/home/${aws:username}",
        "arn:aws:s3:::BUCKET-NAME/home/${aws:username}/*"
      ]
    }
  ]
}
```

The previous example policy uses a policy variable (p. 232) (`${aws:username}`) that is evaluated at run time and contains the friendly name (p. 9) of the IAM user who made the request.

Console-based access is granted by the statements that include the `ListAllMyBuckets`, `GetBucketLocation`, and `ListBucket` actions; these actions are required in order to be able to get to the bucket listings in the AWS Management Console. When the preceding policy is attached to a group, each user in the group can read, write, and list objects only in their home directory. The policy also lets the user see that other user directories exist in the bucket, but users cannot list, read, nor write the contents of the other users' directories.

Allow Users Signed In Using Amazon Cognito to Access their Own Amazon S3 Folder

Amazon Cognito is an easy way to use web identity federation in your mobile app. Using Amazon Cognito, you can provide access to AWS resources for users who have signed in to your app using a third-party identity provider like Login with Amazon, Facebook, Google, or any Open-ID Connect (OIDC) compatible identity provider instead of using an IAM user. To use Amazon Cognito for web identity federation, you create a role that determines what permissions the federated user will have. You can create one role for authenticated users. If your app allows unauthenticated (guest) users, you can create a second role that defines the permissions for those users.

For more information about Amazon Cognito, see the following:

- Amazon Cognito Identity in the *AWS SDK for Android Developer Guide*
- Amazon Cognito Identity in the *AWS SDK for iOS Developer Guide*

The following example shows a policy that might be used for a mobile app that uses Amazon Cognito. The condition makes sure that the user has access to objects in the Amazon S3 bucket represented by EXAMPLE-BUCKET-NAME only if the object's name includes a provider name (here, `cognito`), the friendly name of the application (here, `mynumbersgame`), and the federated user's ID.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["s3:ListBucket"],
      "Resource": ["arn:aws:s3:::EXAMPLE-BUCKET-NAME"],
      "Condition": {"StringLike": {"s3:prefix": ["cognito/mynumbersgame/"]}}
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObject"
      ],
      "Resource": [
        "arn:aws:s3:::EXAMPLE-BUCKET-NAME/cognito/mynumbersgame/${cognito-
identity.amazonaws.com:sub}",
        "arn:aws:s3:::EXAMPLE-BUCKET-NAME/cognito/mynumbersgame/${cognito-
identity.amazonaws.com:sub}/*"
      ]
    }
  ]
}
```

Deny All Access Except to a Specific Set of AWS Products and Resources

The following policy gives users access to only the following:

- The DynamoDB table whose name is represented by EXAMPLE-TABLE-NAME.
- The AWS account's corporate Amazon S3 bucket whose name is represented by EXAMPLE-BUCKET-NAME and all the objects it contains.

The policy includes an explicit deny (p. 244) (`"Effect":"Deny"`). In conjunction with the `NotAction` and `NotResource` elements, this helps to ensure that the users can't use any AWS actions or resources except those specified in the policy, even if permissions have been granted in another policy. (An explicit deny statement takes precedence over an allow statement.) For more information, see IAM Policy Evaluation Logic (p. 243).

Note
The following policy works only for API access. For more information about granting bucket access through the AWS Management Console, see An Example: Using IAM Policies to Control Access to your Bucket in the *Amazon Simple Storage Service Developer Guide*.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:*",
        "s3:*"
      ],
      "Resource": [
        "arn:aws:dynamodb:AWS-REGION-IDENTIFIER:ACCOUNT-ID-WITHOUT-HY
PHENS:table/EXAMPLE-TABLE-NAME",
        "arn:aws:s3:::EXAMPLE-BUCKET-NAME",
        "arn:aws:s3:::EXAMPLE-BUCKET-NAME/*"
      ]
    },
    {
      "Effect": "Deny",
      "NotAction": [
        "dynamodb:*",
        "s3:*"
      ],
      "NotResource": [
        "arn:aws:dynamodb:AWS-REGION-IDENTIFIER:ACCOUNT-ID-WITHOUT-HY
PHENS:table/EXAMPLE-TABLE-NAME",
        "arn:aws:s3:::EXAMPLE-BUCKET-NAME",
        "arn:aws:s3:::EXAMPLE-BUCKET-NAME/*"
      ]
    }
  ]
}
```

Allow a User All Actions on an DynamoDB Table Whose Name Matches the User Name

The following policy can be attached to an IAM group and gives a user permission to programmatically access an DynamoDB table whose name matches the user's name. For example, user `Bob` can perform any DynamoDB actions in the table named `Bob`. The policy can be attached to a group that contains users who are allowed to each manage their own DynamoDB table.

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": "dynamodb:*",
    "Resource": "arn:aws:dynamodb:AWS-REGION-IDENTIFIER:ACCOUNT-ID-WITHOUT-HY
PHENS:table/${aws:username}"
  }]
}
```

The policy uses a policy variable (p. 232) (`${aws:username}`) that is evaluated at run time and contains the friendly name (p. 9) of the IAM user who made the request.

Allow Users to Manage Amazon EC2 Resources with a Specific Tag and in a Specified Date Range

The following example policy allows users to perform any Amazon EC2 action on resources (such as instances) that have the tag `Department=Test`, as long as the request occurs before the beginning of the year 2016. This policy shows an example of multiple conditions, which are evaluated using a logical AND.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Action": "ec2:*",
    "Effect": "Allow",
    "Resource": "*",
    "Condition": {
      "StringEquals": {"ec2:ResourceTag/Department": "Test"},
      "DateLessThan": {"aws:CurrentTime": "2016-01-01T00:00:00Z"}
    }
  }
}
```

For more information about tagging in Amazon EC2, see Tagging Your Amazon EC2 Resources in the *Amazon EC2 User Guide for Linux Instances*. For more information about adding conditions to policies, see Condition (p. 219).

Allow Users to Manage Amazon EBS Volumes for Particular Amazon EC2 Instances

The following example policy allows users to attach Amazon EBS volumes that have the tag `volume_user=`*`IAM user name`* to Amazon EC2 instances that have the tag `department=dev`, and to detach the volumes from those instances. When you attach this policy to an IAM group, the `${aws:username}` policy variable (p. 232) resolves to the user name of the IAM user, and thus the policy grants each IAM user in the group permission to attach or detach volumes that have a tag named `volume_user` that has his or her IAM user name as a value.

For more information about creating IAM policies to control access to Amazon EC2 resources, see Controlling Access to Amazon EC2 Resources in the *Amazon EC2 User Guide for Linux Instances*.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:AttachVolume",
        "ec2:DetachVolume"
      ],
      "Resource": "arn:aws:ec2:AWS-REGION-IDENTIFIER:ACCOUNT-ID-WITHOUT-HY
PHENS:instance/*",
      "Condition": {"StringEquals": {"ec2:ResourceTag/department": "dev"}}
    },
    {
      "Effect": "Allow",
      "Action": [
        "ec2:AttachVolume",
        "ec2:DetachVolume"
      ],
      "Resource": "arn:aws:ec2:AWS-REGION-IDENTIFIER:ACCOUNT-ID-WITHOUT-HY
PHENS:volume/*",
      "Condition": {"StringEquals": {"ec2:ResourceTag/volume_user":
"${aws:username}"}}
    }
  ]
}
```

Block Requests That Don't Come From An Approved IP Address or Range

You might find this policy useful to apply to an IAM group that all the users in your company belong to. The policy denies access to all actions in the account when the request comes from outside the IP range 192.0.2.0 to 192.0.2.255 *or* 203.0.113.0 to 203.0.113.255. (The policy assumes the IP addresses for your company are within the specified ranges.) Use this policy in combination with other policies that allow specific actions. Regardless of which actions are allowed to a user or group via another policy, this policy ensures that all actions will be denied if the request originates from outside your company's IP address ranges.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Deny",
    "Action": "*",
    "Resource": "*",
    "Condition": {"NotIpAddress": {"aws:SourceIp": [
      "192.0.2.0/24",
      "203.0.113.0/24"
    ]}}
  }
}
```

AWS IAM Policy Reference

This section presents detailed syntax, descriptions, and examples of the elements, variables, and evaluation logic of IAM policies. It includes the following sections.

- IAM Policy Elements Reference (p. 209) — This section describes each of the elements that you can use when you create a policy. It includes additional policy examples and describes conditions, supported data types, and how they are used in various services.
- IAM Policy Variables Overview (p. 232) — This section describes placeholders that you can specify in a policy that are replaced during policy evaluation with values from the request.
- Creating a Condition That Tests Multiple Key Values (Set Operations) (p. 238) — This section describes how to create policies for requests in which a request key includes multiple items that you need to test against a set of values.
- IAM Policy Evaluation Logic (p. 243) — This section describes AWS requests, how they are authenticated, and how AWS uses policies to determine access to resources.
- Grammar of the IAM Policy Language (p. 248) — This section presents a formal grammar for the language used to create policies in IAM.
- Common Errors in IAM Policies (p. 253) — This section describes the most common policy errors that can prevent you from saving changes to a policy and how to correct them.

Important
AWS is making improvements to the policy rules engine that enforce a stricter syntax. Starting in March 2015, you cannot save any policy that does not comply with the stricter rules.
To help you correct invalid policies, AWS created the Policy Validator that informs you whenever it detects an invalid policy. One click takes you to an editor that shows both the existing policy and a copy with the recommended changes. You can accept the changes or make further modifications. For more information, see Using Policy Validator (p. 201).

IAM Policy Elements Reference

This section describes the elements that you can use in an IAM policy. The elements are listed here in the general order you use them in a policy. The order of the elements doesn't matter—for example, the `Resource` element can come before the `Action` element. You're not required to specify any `Condition` elements in the policy.

Topics

Note
The details of what goes into a policy vary for each service, depending on what actions the service makes available, what types of resources it contains, and so on. When you're writing policies for a specific service, it's helpful to see examples of policies for that service. For a list of all the services that support IAM, and for links to the documentation in those services that discusses IAM and policies, see AWS Services That Support IAM (p. 265).

Version

The `Version` element specifies the policy language version. The only allowed values are these:

- `2012-10-17`. This is the current version of the policy language, and you should use this version number for all policies.
- `2008-10-17`. This was an earlier version of the policy language. You might see this version on existing policies. Do not use this version for any new policies or any existing policies that you are updating.

If you do not include a `Version` element, the value defaults to `2008-10-17`. However, it is a good practice to always include a `Version` element and set it to `2012-10-17`.

Note
If your policy includes policy variables (p. 232), you *must* include a `Version` element and set it to `2012-10-17`. If you don't include a `Version` element set to `2012-10-17`, variables such as `${aws:username}` won't be recognized as variables and will instead be treated as literal strings in the policy.

```
"Version":"2012-10-17"
```

Id

The `Id` element specifies an optional identifier for the policy. The ID is used differently in different services.

For services that let you set an `ID` element, we recommend you use a UUID (GUID) for the value, or incorporate a UUID as part of the ID to ensure uniqueness.

```
"Id":"cd3ad3d9-2776-4ef1-a904-4c229d1642ee"
```

Note
Some AWS services (for example, Amazon SQS or Amazon SNS) might require this element and have uniqueness requirements for it. For service-specific information about writing policies, refer to the documentation for the service you're working with.

Statement

The `Statement` element is the main element for a policy. This element is required. It can include multiple elements (see the subsequent sections in this page). The `Statement` element contains an array of individual statements. Each individual statement is a JSON block enclosed in braces `{ }`.

```
"Statement":[{...},{...},{...}]
```

The following example shows a policy that contains an array of three statements inside a single `Statement` element. (The policy allows you to access your own "home folder" in the Amazon S3 console.) The policy includes the `aws:username` variable, which is replaced during policy evaluation with the user name from the request. For more information, see Introduction (p. 232).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListAllMyBuckets",
        "s3:GetBucketLocation"
      ],
      "Resource": "arn:aws:s3:::*"
    },
    {
      "Effect": "Allow",
      "Action": "s3:ListBucket",
      "Resource": "arn:aws:s3:::BUCKET-NAME",
      "Condition": {"StringLike": {"s3:prefix": [
        "",
        "home/",
        "home/${aws:username}/"
      ]}}
    },
    {
      "Effect": "Allow",
      "Action": "s3:*",
      "Resource": [
        "arn:aws:s3:::BUCKET-NAME/home/${aws:username}",
        "arn:aws:s3:::BUCKET-NAME/home/${aws:username}/*"
      ]
    }
  ]
}
```

Sid

The `Sid` (statement ID) is an optional identifier that you provide for the policy statement. You can assign a `Sid` value to each statement in a statement array. In services that let you specify an `ID` element, such as SQS and SNS, the `Sid` value is just a sub-ID of the policy document's ID. In IAM, the `Sid` value must be unique within a policy.

```
"Sid":"1"
```

In IAM, the `Sid` is not exposed in the IAM API. You can't retrieve a particular statement based on this ID.

Note
Some AWS services (for example, Amazon SQS or Amazon SNS) might require this element and have uniqueness requirements for it. For service-specific information about writing policies, refer to the documentation for the service you're working with.

Effect

The `Effect` element is required and specifies whether the statement will result in an allow or an explicit deny. Valid values for `Effect` are `Allow` and `Deny`.

```
"Effect":"Allow"
```

By default, access to resources is denied. To allow access to a resource, you must set the `Effect` element to `Allow`. To override an allow (for example, to override an allow that is otherwise in force), you set the `Effect` element to `Deny`. For more information, see IAM Policy Evaluation Logic (p. 243).

Principal

Use the `Principal` element to specify the user (IAM user, federated user, or assumed-role user), AWS account, AWS service, or other principal entity that is allowed or denied access to a resource. You use the `Principal` element in the trust policies for IAM roles and in resource-based policies—that is, in policies that you embed directly in a resource. For example, you can embed such policies in an Amazon S3 bucket, an Amazon SNS topic, an Amazon SQS queue, or an AWS KMS encryption key.

Use the `Principal` element in these ways:

- In IAM roles, use the `Principal` element in the role's trust policy to specify who can assume the role. For cross-account access, you typically specify the identifier of the trusted account.
- In resource-based policies, use the `Principal` element to specify the accounts or users who are allowed to access the resource.

Do not use the `Principal` element in policies that you attach to IAM users and groups. Similarly, you do not specify a principal in the access policy for an IAM role. In those cases, the principal is implicitly the user that the policy is attached to (for IAM users) or the user who assumes the role (for role access policies). When the policy is attached to an IAM group, the principal is the IAM user in that group who is making the request.

Specifying a Principal

You specify a principal using the *Amazon Resource Name* (ARN) (p. 9) of the AWS account, IAM user, IAM role, federated user, or assumed-role user. You cannot specify IAM groups as principals. When you specify an AWS account, you can use a shortened form that consists of the `AWS:` prefix followed by the account ID, instead of using the account's full ARN.

The following examples show various ways in which principals can be specified.

Everyone (anonymous users)

```
"Principal": "*"
```

Specific AWS accounts

When you use an AWS account identifier as the principal in a policy, the permissions in the policy statement can be granted to all identities contained in that account, including IAM users and roles in that account. The following examples show different ways to specify an AWS account as a principal.

```
"Principal": {"AWS": "arn:aws:iam::AWS-account-ID:root"}
```

```
"Principal": {"AWS": "AWS-account-ID"}
```

You can specify more than one AWS account as a principal, as shown in the following example.

```
"Principal": {"AWS": [
  "arn:aws:iam::AWS-account-ID:root",
  "arn:aws:iam::AWS-account-ID:root"
]}
```

Individual IAM user or users

You can specify an individual IAM user (or array of users) as the principal, as in the following examples.

Note
In a `Principal` element, the user name is case sensitive.

```
"Principal": {"AWS": "arn:aws:iam::AWS-account-ID:user/user-name"}
```

```
"Principal": {"AWS": [
  "arn:aws:iam::AWS-account-ID:user/user-name-1",
  "arn:aws:iam::AWS-account-ID:user/UserName2"
]}
```

When you specify users in a `Principal` element, you cannot use a wildcard (`*`) to mean "all users". Principals must always name a specific user or users.

Federated users (using web identity federation)

```
"Principal": {"Federated": "cognito-identity.amazonaws.com"}
```

```
"Principal": {"Federated": "www.amazon.com"}
```

```
"Principal": {"Federated": "graph.facebook.com"}
```

```
"Principal": {"Federated": "accounts.google.com"}
```

Federated users (using a SAML identity provider)

```
"Principal": {"Federated": "arn:aws:iam::AWS-account-ID:saml-provider/provider-
name"}
```

IAM role

```
"Principal": {"AWS": "arn:aws:iam::AWS-account-ID:role/role-name"}
```

Specific assumed-role user

```
"Principal" {"AWS": "arn:aws:sts::AWS-account-ID:assumed-role/role-name/role-
session-name"}
```

AWS service

When you create a trust policy for an IAM role that will be assumed by an AWS service, you typically specify the principal using a friendly name for that service, as in the following example.

```
"Principal": {"Service": "ec2.amazonaws.com"}
```

Some AWS services support additional options for specifying a principal. For example, Amazon S3 lets you use a canonical user in a format like this:

```
"Principal": {"CanonicalUser": "79a59df900b949e55d96a1e698fbaced
fd6e09d98eacf8f8d5218e7cd47ef2be"}
```

More Information

For more information, see the following:

- Bucket Policy Examples in the *Amazon Simple Storage Service Developer Guide*
- Example Policies for Amazon SNS in the *Amazon Simple Notification Service Developer Guide*
- Amazon SQS Policy Examples in the *Amazon Simple Queue Service Developer Guide*
- Key Policies in the *AWS Key Management Service Developer Guide*
- Account Identifiers in the *AWS General Reference*
- Creating Temporary Security Credentials for Mobile Apps Using Mobile Apps Using Identity Providers in *Using Temporary Security Credentials*

NotPrincipal

Use the `NotPrincipal` element to specify an exception to a list of principals. For example, you can deny access to all principals *except* the one named in the `NotPrincipal` element. The syntax for specifying `NotPrincipal` is the same as for specifying Principal (p. 211).

Note that you can also use `NotPrincipal` to allow all principals *except* the one named in the `NotPrincipal` element; however, we do not recommend this.

Warning
When you use `NotPrincipal` in the same policy statement as `"Effect": "Allow"`, the permissions specified in the policy statement will be granted to *all* principals except for the one(s) specified, including anonymous (unauthenticated) users. We strongly recommend you do not use `NotPrincipal` in the same policy statement as `"Effect": "Allow"`.

When you use `NotPrincipal` in the same policy statement as `"Effect": "Deny"`, the permissions specified in the policy statement are explicitly denied to all principals *except* for the one(s) specified. When you explicitly deny access to an AWS account, you deny access to all users contained in that account.

You cannot explicitly deny access to your own AWS account. That is, a resource-based policy cannot explicitly deny access to the AWS account that contains the resource. However, you can explicitly deny access to other AWS accounts. When a resource-based policy combines `"Effect": "Deny"` with a `NotPrincipal` element that specifies a principal in another account, it is possible that the specified principal will be unable to access the resource. To understand how this can happen, see the examples in the next section.

Caution
Very few scenarios require the use of `NotPrincipal`, and we recommend that you explore other authorization options before you decide to use `NotPrincipal`.

Specifying `NotPrincipal` in the same policy statement as `"Effect": "Deny"`

You specify principals in the `NotPrincipal` element using the same syntax that you use for specifying principals in the Principal (p. 211) element. However, it can be difficult to achieve the intended effect, particularly when you combine `NotPrincipal` with `"Effect": "Deny"` in the same policy statement and you work across AWS account boundaries. The following examples show ways to use `NotPrincipal` and `"Effect": "Deny"` in the same policy statement effectively.

Example 1: IAM user in the same account

In the following example, all principals *except* the user named Alice in AWS account 111122223333 are explicitly denied access to a resource. This example works as intended when it is part of a policy statement in a resource-based policy that is attached to a resource in AWS account 111122223333. This example by itself does not grant access to Alice, it only omits Alice from the list of principals that are explicitly denied. To give Alice access to the resource, another policy statement must explicitly allow access using `"Effect": "Allow"`.

```
"Effect": "Deny",
"NotPrincipal": {"AWS": "arn:aws:iam::111122223333:user/Alice"}
```

Example 2: IAM user in a different account

In the following example, all principals *except* the user named Bob in AWS account 444455556666 are explicitly denied access to a resource. Note that to achieve the intended effect, the `NotPrincipal` element contains the ARN of both the user Bob and the AWS account that Bob belongs to (`arn:aws:iam::444455556666:root`). If the `NotPrincipal` element contained only Bob's ARN, the effect of the policy would be to explicitly deny access to Bob's AWS account. Users cannot have more permissions than their parent account, so when Bob's account is explicitly denied access, Bob is unable to access the resource.

This example works as intended when it is part of a policy statement in a resource-based policy that is attached to a resource in a different AWS account (not 444455556666). This example by itself does not grant access to Bob, it only omits Bob from the list of principals that are explicitly denied. To give Bob access to the resource, another policy statement must explicitly allow access using `"Effect": "Allow"`.

```
"Effect": "Deny",
"NotPrincipal": {"AWS": [
  "arn:aws:iam::444455556666:user/Bob",
  "arn:aws:iam::444455556666:root"
]}
```

Example 3: IAM role in the same account

In the following example, all principals *except* the assumed-role user named HR-database-app in AWS account 111122223333 are explicitly denied access to a resource. Note that to achieve the intended effect, the `NotPrincipal` element contains the ARN of both the role (HR-database-access-role) and the assumed-role user (HR-database-app). If the `NotPrincipal` element contained only the ARN of the assumed-role user, the effect of the policy would be to explicitly deny access to the role. Assumed-role users cannot have more permissions than their parent role, so when the role is explicitly denied access, the assumed role user is unable to access the resource.

This example works as intended when it is part of a policy statement in a resource-based policy that is attached to a resource in AWS account 111122223333. This example by itself does not grant access to the assumed-role user HR-database-app, it only omits HR-database-app from the list of principals that are explicitly denied. To give HR-database-app access to the resource, another policy statement must explicitly allow access using `"Effect": "Allow"`.

```
"Effect": "Deny",
"NotPrincipal": {"AWS": [
  "arn:aws:sts::111122223333:assumed-role/HR-database-access-role/HR-database-
app",
  "arn:aws:iam::111122223333:role/HR-database-access-role"
]}
```

Example 4: IAM role in a different account

In the following example, all principals *except* the assumed-role user named cross-account-audit-app in AWS account 444455556666 are explicitly denied access to a resource. Note that to achieve the intended effect, the `NotPrincipal` element contains the ARN of the assumed-role user (cross-account-audit-app), the role (cross-account-read-only-role), and the AWS account that the role belongs to (444455556666). If the `NotPrincipal` element was missing the ARN of the role, the effect of the policy would be to explicitly deny access to the role. Similarly, if the `NotPrincipal` element was missing the ARN of the AWS account that the role belongs to, the effect of the policy would be to explicitly deny access to the AWS account. Assumed-role users cannot have more permissions than their parent role, and roles cannot have more permissions than their parent AWS account, so when the role or the account is explicitly denied access, the assumed role user is unable to access the resource.

This example works as intended when it is part of a policy statement in a resource-based policy that is attached to a resource in a different AWS account (not 444455556666). This example by itself does not grant access to the assumed-role user cross-account-audit-app, it only omits cross-account-audit-app from the list of principals that are explicitly denied. To give cross-account-audit-app access to the resource, another policy statement must explicitly allow access using `"Effect": "Allow"`.

```
"Effect": "Deny",
"NotPrincipal": {"AWS": [
  "arn:aws:sts::444455556666:assumed-role/cross-account-read-only-role/cross-
account-audit-app",
  "arn:aws:iam::444455556666:role/cross-account-read-only-role",
  "arn:aws:iam::444455556666:root"
]}
```

Action

The `Action` element describes the specific action or actions that will be allowed or denied. Statements must include either an `Action` or `NotAction` element. Each AWS service has its own set of actions that describe tasks that you can perform with that service. For example, the list of actions for Amazon S3 can be found at Specifying Permissions in a Policy in the *Amazon Simple Storage Service Developer Guide*, the list of actions for Amazon EC2 can be found in the Amazon EC2 API Reference, and the list of actions for AWS Identity and Access Management can be found in the IAM API Reference. To find the list of actions for other services, consult the API reference documentation for the service.

You specify a value using a namespace that identifies a service (`iam`, `ec2` `sqs`, `sns`, `s3`, etc.) followed by the name of the action to allow or deny. The name must match an action that is supported by the service. The prefix and the action name are case insensitive. For example, `iam:ListAccessKeys` is the same as `IAM:listaccesskeys`. The following examples show `Action` elements for different services.

Amazon SQS action

```
"Action":"sqs:SendMessage"
```

Amazon EC2 action

```
"Action":"ec2:StartInstances"
```

IAM action

```
"Action":"iam:ChangePassword"
```

Amazon S3 action

```
"Action":"s3:GetObject"
```

You can specify multiple values for the `Action` element.

```
"Action":["sqs:SendMessage","sqs:ReceiveMessage"]
```

You can use a wildcard (*) to give access to all the actions the specific AWS product offers. For example, the following `Action` element applies to all IAM actions.

```
"Action":"iam:*"
```

You can also use wildcards (*) as part of the action name. For example, the following `Action` element applies to all IAM actions that include the string `AccessKey`, including `CreateAccessKey`, `DeleteAccessKey`, `ListAccessKeys`, and `UpdateAccessKey`.

```
"Action":"iam:*AccessKey*"
```

Some services let you limit the actions that are available. For example, Amazon SQS lets you make available just a subset of all the possible Amazon SQS actions. In that case, the `*` wildcard doesn't allow complete control of the queue; it allows only the subset of actions that you've shared. For more information, see Understanding Permissions in the *Amazon Simple Queue Service Developer Guide*.

NotAction

The `NotAction` element lets you specify an exception to a list of actions. For example, you can use `NotAction` to let users use only the Amazon SQS `SendMessage` action, without having to list all the actions that the user is not allowed to perform. Using `NotAction` can sometimes result in shorter policies than using an `Action` element and listing many actions.

The following example refers to all actions *other* than the Amazon SQS `SendMessage` action. You might use this in a policy with `"Effect":"Deny"` to keep users from accessing any other actions except `SendMessage` Note, however, that this would not grant the user access to any actions, it would only explicitly deny all other actions except `SendMessage`.

```
"NotAction":"sqs:SendMessage"
```

The following example matches any action except `Publish`.

```
"NotAction":"sns:Publish"
```

The following example shows how to reference all Amazon S3 actions except `GetObject`.

```
"NotAction":"s3:GetObject"
```

For an example of how to use the `NotAction` element in a policy that controls access to an Amazon S3 bucket, see Example Policies for Amazon S3 in the *Amazon Simple Storage Service Developer Guide*.

Resource

The `Resource` element specifies the object or objects that the statement covers. Statements must include either a `Resource` or a `NotResource` element. You specify a resource using an ARN. (For more information about the format of ARNs, see Amazon Resource Names (ARNs) and AWS Service Namespaces.)

Each service has its own set of resources. Although you always use an ARN to specify a resource, the details of the ARN for a resource depend on the service and the resource. For information about how to specify a resource, refer to the documentation for the service whose resources you're writing a statement for.

Note
Some services do not let you specify actions for individual resources; instead, any actions that you list in the `Action` or `NotAction` element apply to all resources in that service. In these cases, you use the wildcard `*` in the `Resource` element.

The following example refers to a specific Amazon SQS queue.

```
"Resource":"arn:aws:sqs:us-west-2:account-ID-without-hyphens:queue1"
```

The following example refers to the IAM user named Bob in an AWS account.

```
"Resource":"arn:aws:iam::account-ID-without-hyphens:user/Bob"
```

You can use wildcards as part of the resource ARN. The following example refers to all IAM users whose path is `/accounting`.

```
"Resource":"arn:aws:iam::account-ID-without-hyphens:user/accounting/*"
```

The following example refers to all items within a specific Amazon S3 bucket.

```
"Resource":"arn:aws:s3:::my_corporate_bucket/*"
```

You can specify multiple resources. The following example refers to two DynamoDB tables.

```
"Resource":[
    "arn:aws:dynamodb:us-west-2:account-ID-without-hyphens:table/books_table",

   "arn:aws:dynamodb:us-west-2:account-ID-without-hyphens:table/magazines_table"
]
```

In the `Resource` element, you can use policy variables (p. 232) in the part of the ARN that identifies the specific resource (that is, in the trailing part of the ARN). For example, you can use the key `{aws:username}` as part of a resource ARN to indicate that the current user's name should be included as part of the resource's name. The following example shows how you can use the `{aws:username}` key in a `Resource` element. The policy allows access to a Amazon DynamoDB table that matches the current user's name.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "dynamodb:*",
    "Resource": "arn:aws:dynamodb:us-east-1:ACCOUNT-ID-WITHOUT-HY
PHENS:table/${aws:username}"
  }
}
```

For more information about policy variables, see IAM Policy Variables Overview (p. 232).

NotResource

The `NotResource` element lets you grant or deny access to all but a few of your resources, by allowing you to specify only those resources to which your policy should *not* be applied.

For example, imagine you have a group named `Developers`. Members of `Developers` should have access to all of your Amazon S3 resources except the `CompanySecretInfo` folder in the `mybucket` bucket. The following example shows what the policy to establish these permissions might look like.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "s3:*",
    "NotResource": [
      "arn:aws:s3:::mybucket/CompanySecretInfo",
      "arn:aws:s3:::mybucket/CompanySecretInfo/*"
    ]
  }
}
```

Normally you would write a policy that uses `"Effect":"Allow"` and that includes a `Resources` element that lists each folder individually that the `Developers` group has access to. However, in that case, each time you added a folder to `mybucket` that users should have access to, you would have to add its name to the list in `Resource`. If you use a `NotResource` element instead, users will automatically have access to new folders unless you add the folder names to the `NotResource` element.

Condition

The `Condition` element (or `Condition` *block*) lets you specify conditions for when a policy is in effect. The `Condition` element is optional. In the `Condition` element, you build expressions in which you use Boolean operators (equal, less than, etc.) to match your condition against values in the request. Condition values can include date, time, the IP address of the requester, the ARN of the request source, the user name, user ID, and the user agent of the requester. Some services let you specify additional values in conditions; for example, Amazon S3 lets you write a condition using the `s3:VersionId` key, which is unique to that service. (For more information about writing conditions in policies for Amazon S3, see Using IAM Policies in the Amazon Simple Storage Service Developer Guide.)

The Condition Block

The following example shows the basic format of a `Condition` element:

```
"Condition": {
  "DateGreaterThan" : {
     "aws:CurrentTime" : "2013-12-15T12:00:00Z"
   }
}
```

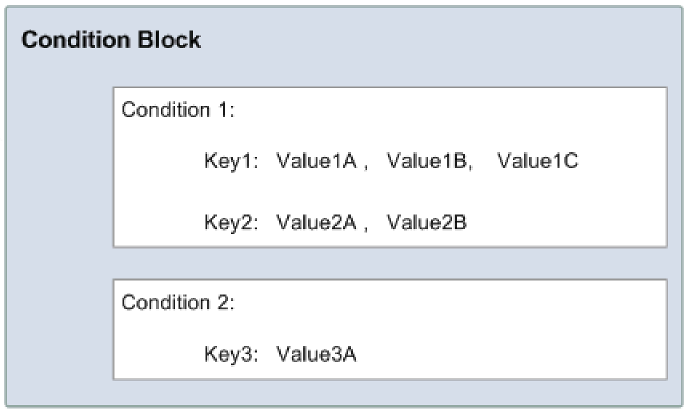
A value from the request is represented by a key, in this case `aws:CurrentTime`. The key value is compared to a value that you specify either as a literal value (`2013-08-16T12:00:00Z`) or as a policy variable, as explained later. The type of comparison to make is specified by the condition (here, `DateGreaterThan`). You can create string conditions, date conditions, numeric conditions, and so on, and you can use typical Boolean comparisons like equals, greater than, and less than.

Under some circumstances, keys can contains multiple values. For example, a request to DynamoDB might ask to return or update multiple attributes from a table. A policy for access to DynamoDB tables can include the `dynamodb:Attributes` key, which contains all the attributes listed in the request. You can test the multiple attributes in the request against a list of allowed attributes in a policy by using set operators in a condition. For more information, see Creating a Condition That Tests Multiple Key Values (Set Operations) (p. 238).

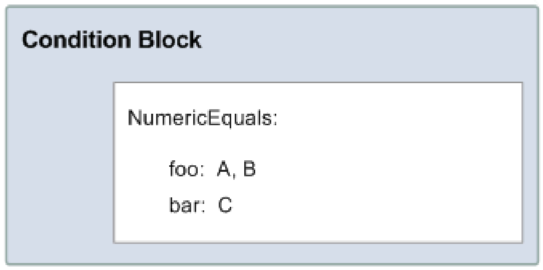
When the policy is evaluated during a request, AWS replaces the key with the corresponding value from the request. (In this example, AWS would use the date and time of the request.) The condition is evaluated to return true or false, which is then factored into whether the policy as a whole allows or denies the request.

Multiple Values in a Condition

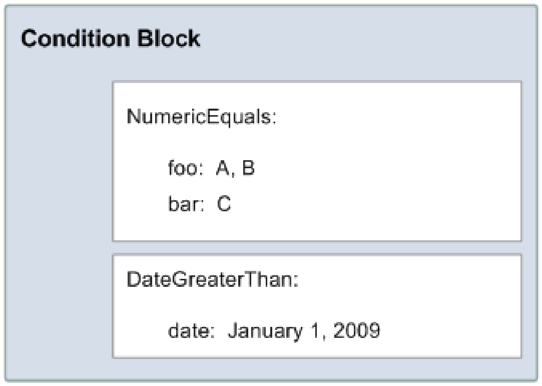
A `Condition` element can contain multiple conditions, and each condition can contain multiple key-value pairs. The following figure illustrates this. Unless otherwise specified, all keys can have multiple values.



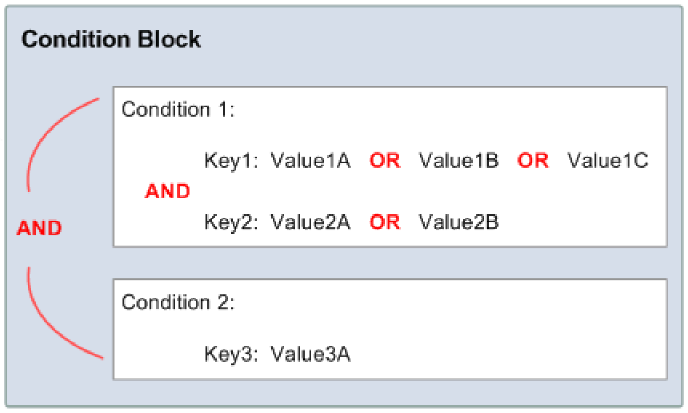
Let's say you want to let John use a resource only if a numeric value *foo* equals either A or B, and another numeric value *bar* equals C. You would create a condition block that looks like the following figure.



Let's say you also want to restrict John's access to after January 1, 2009. You would add another condition, `DateGreaterThan`, with a date equal to January 1, 2009. The condition block would then look like the following figure.



If there are multiple conditions, or if there are multiple keys in a single condition, the conditions are evaluated using a logical `AND`. If a single condition includes multiple values for one key, the condition is evaluated using a logical `OR`. All conditions must be met for an allow or an explicit deny decision. If a condition isn't met, the result is a deny.



As noted, AWS has predefined conditions and keys (like `aws:CurrentTime`). Individual AWS services also define service-specific keys.

As an example, let's say you want to let user John access your Amazon SQS queue under the following conditions:

- The time is after 12:00 noon on 8/16/2013
- The time is before 3:00 p.m. on 8/16/2013
- The request (IAM or SQS) or message (SNS) comes from an IP address within the range 192.0.2.0 to 192.0.2.255 *or* 203.0.113.0 to 203.0.113.255.

Your condition block has three separate conditions, and all three of them must be met for John to have access to your queue, topic, or resource.

The following shows what the condition block looks like in your policy. The two values for `aws:SourceIp` are evaluated using OR. The three separate conditions are evaluated using AND.

```
"Condition" :  {
      "DateGreaterThan" : {
         "aws:CurrentTime" : "2013-08-16T12:00:00Z"
       },
      "DateLessThan": {
         "aws:CurrentTime" : "2013-08-16T15:00:00Z"
       },
       "IpAddress" : {
          "aws:SourceIp" : ["192.0.2.0/24", "203.0.113.0/24"]
      }
}
```

Finally, under some circumstances, individual keys in a policy can contain multiple values, and you can use set operators to test these multi-valued keys against one or more values listed in the policy. For more information, see Creating a Condition That Tests Multiple Key Values (Set Operations) (p. 238).

Available Keys for Conditions

Context keys in a condition represent values that are part of the request sent by a user. AWS provides the following predefined context keys for all AWS services that support IAM for access control:

- `aws:CurrentTime` – To check for date/time conditions (see Date Conditions (p. 228)).
- `aws:EpochTime` – To check for date/time conditions using a date in epoch or UNIX time (see Date Conditions (p. 228)).
- `aws:TokenIssueTime` – To check the date/time that temporary security credentials were issued (see Date Conditions (p. 228)). This key is only present in requests that are signed using temporary security credentials. For more information about temporary security credentials, see *Using Temporary Security Credentials*.
- `aws:MultiFactorAuthPresent` – To check whether multi-factor authentication (MFA) was used to validate the security credentials making the current request (see Boolean Conditions (p. 228)).
- `aws:MultiFactorAuthAge` – To check how long ago (in seconds) the MFA-validated security credentials making the request were issued using multi-factor authentication (MFA). Unlike other keys, if MFA is not used, this key is not present (see Existence of Condition Keys (p. 231), Numeric Conditions (p. 227) and Using Multi-Factor Authentication (MFA) Devices with AWS (p. 69).
- `aws:principaltype` – To check the type of principal (user, account, federated user, etc.) for the current request (see String Conditions (p. 225)).
- `aws:SecureTransport` – To check whether the request was sent using SSL (see Boolean Conditions (p. 228)).
- `aws:SourceArn` – To check the source of the request, using the Amazon Resource Name (ARN) of the source. (This value is available for only some services. For more information, see Amazon Resource Name (ARN) under "Element Descriptions" in the *Amazon Simple Queue Service Developer Guide*.)
- `aws:SourceIp` – To check the requester's IP address (see IP Address (p. 229)).
- `aws:UserAgent` – To check the requester's client application (see String Conditions (p. 225)).
- `aws:userid` – To check the requester's user ID (see String Conditions (p. 225)).
- `aws:username` – To check the requester's user name (see String Conditions (p. 225)).

Some AWS services also provide service-specific keys. For example, for information about keys that you can use in policies for Amazon S3 resources, see Amazon S3 Policy Keys in the *Amazon Simple Storage Service Developer Guide*. For information about keys that you can use in policies for IAM resources, see the following section. For information about keys that you can use in policies for other services, see the documentation for the individual services.

Available Keys for IAM

You can use the following keys in policies that control access to IAM resources:

- `iam:PolicyArn` – To check the Amazon Resource Name (ARN) of a managed policy in requests that involve a managed policy. For more information, see Specifying the Amazon Resource Name (ARN) for Managed Policies (p. 188).

Available Keys for Web Identity Federation

If you are using web identity federation to give temporary security credentials to users who have been authenticated using an identity provider (IdP) such as Login with Amazon, Amazon Cognito, Google, or Facebook, additional condition keys are available when the temporary security credentials are used to make a request. These keys let you write policies that make sure that federated users can get access only to resources that are associated with a specific provider, app, or user.

Federated Provider

The `aws:FederatedProvider` key identifies which of the IdPs was used to authenticate the user. For example, if the user authenticated using Amazon Cognito, the key would contain `cognito-identity.amazonaws.com`. Similarly, if the user authenticated using Login with Amazon, the key would contain the value `www.amazon.com`. You might use the key in a resource policy like the following, which uses the `aws:FederatedProvider` key as a policy variable in the ARN of a resource. The policy allows any user who has been authenticated using an IdP to get objects out of a folder in an Amazon S3 bucket that's specific to the provider they used to authenticate with.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Principal": "*",
    "Action": "s3:GetObject",
    "Resource": "arn:aws:s3:::BUCKET-NAME/${aws:FederatedProvider}/*"
  }
}
```

Application ID and User ID

You can also use two keys that provide a unique identifier for the user and an identifier for the application or site that the user authenticated with. These keys have the following IdP-specific names:

- For Amazon Cognito users, the keys are `cognito-identity.amazonaws.com:aud` (for the identity pool ID) and `cognito-identity.amazonaws.com:sub` (for the user ID).
- For Login With Amazon users, the keys are `www.amazon.com:app_id` and `www.amazon.com:user_id`
- For Facebook users, the keys are `graph.facebook.com:app_id` and `graph.facebook.com:id`
- For Google users, the keys are `accounts.google.com:aud` (for the app ID) and `accounts.google.com:sub` (for the user ID).

The amr Key in Amazon Cognito

If you are using Amazon Cognito for web identity federation, the `cognito-identity.amazonaws.com:amr` key (Authenticated Methods Reference) in a trust policy includes login information about the user. The key is multivalued, meaning that you test it in a policy using condition set operators (p. 238). The key can contain the following values:

- If the user is unauthenticated, the key contains only `unauthenticated`.
- If the user is authenticated, the key contains the value `authenticated` and the name of the login provider used in the call (`graph.facebook.com`, `accounts.google.com`, or `www.amazon.com`).

As an example, the following condition in the trust policy for an Amazon Cognito role tests whether the user is unauthenticated:

```
"Condition": {
  "StringEquals":
    {"cognito-identity.amazonaws.com:aud": "us-east-1:identity-pool-id"},
  "ForAnyValue:StringLike":
    {"cognito-identity.amazonaws.com:amr": "unauthenticated"}
}
```

More Information About Web Identity Federation

For more information about web identity federation, see the following:

- Amazon Cognito Overview in the *AWS SDK for Android Developer Guide* guide
- Amazon Cognito Overview in the *AWS SDK for iOS Developer Guide* guide
- Creating Temporary Security Credentials for Mobile Apps Using Third-Party Identity Providers in the *Using Temporary Security Credentials* guide

Available Keys for SAML-Based Federation

If you are working with SAML-based federation, you can include additional condition keys in the policy.

Trust Policies

In the trust policy of a role, you can include the following keys, which help you establish whether the caller is allowed to assume the role. Except for `saml:doc`, all the values are derived from the SAML assertion. Items in the list that are marked with an asterisk (*) are available in the console UI to create conditions.

- `saml:aud` (URL). An endpoint to which SAML assertions are presented.
- `saml:cn` (list). This is an `eduOrg` attribute.
- `saml:doc` (string).* This represents the principal that was used to assume the role. The format is *`account-ID`*/*`document-friendly-name`*, such as `123456789012/Metadata_Document`. The *account-ID* value refers to the account that owns the SAML provider (p. 105).
- `saml:edupersonaffiliation` (list). This is an `eduPerson` attribute.
- `saml:edupersonassurance` (list). This is an `eduPerson` attribute.
- `saml:edupersonentitlement` (list).* This is an `eduPerson` attribute.
- `saml:edupersonnickname` (list). This is an `eduPerson` attribute.
- `saml:edupersonorgdn` (string).* This is an `eduPerson` attribute.
- `saml:edupersonorgunitdn` (list). This is an `eduPerson` attribute.
- `saml:edupersonprimaryaffiliation` (string). This is an `eduPerson` attribute.
- `saml:edupersonprimaryorgunitdn` (string). This is an `eduPerson` attribute.
- `saml:edupersonprincipalname` (string). This is an `eduPerson` attribute.
- `saml:edupersonscopedaffiliation` (list). This is an `eduPerson` attribute.
- `saml:edupersontargetedid` (list). This is an `eduPerson` attribute.
- `saml:eduorghomepageuri` (list). This is an `eduOrg` attribute.
- `saml:eduorgidentityauthnpolicyuri` (list). This is an `eduOrg` attribute.
- `saml:eduorglegalname` (list). This is an `eduOrg` attribute.
- `saml:eduorgsuperioruri` (list). This is an `eduOrg` attribute.
- `saml:eduorgwhitepagesuri` (list). This is an `eduOrg` attribute.
- `saml:iss` (string).* The issuer, which is represented by a URN.

- `saml:sub` (string).* This is the subject of the claim, which includes a value that uniquely identifies an individual user within an organization (for example, `_cbb88bf52c2510eabe00c1642d4643f41430fe25e3`).
- `saml:sub_type` (string).* This key can be "persistent" or "transient". A value of "persistent" indicates that the value in `saml:sub` is the same for a user between sessions. If the value is "transient", the user has a different `saml:sub` value for each session.

For general information about `eduPerson` and `eduOrg` attributes, see the Internet2 website. For a list of `eduPerson` attributes, see eduPerson Object Class Specification (201203).

Keys whose type is a list can include multiple values. To create conditions in the policy for list values, you can use set operators (p. 238) (`ForAllValues`, `ForAnyValue`). For example, to allow any user whose affiliation is "faculty", "staff", or "employee" (but not "student", "alum", or other possible affiliations), you might use a condition like the following:

```
"Condition": {
   "ForAllValues:StringLike":{
     "saml:edupersonaffiliation":["faculty","staff","employee"]}
   }
}
```

Access (Permissions) Policies

In the access policy of a role for SAML federation that defines what users are allowed to access in AWS, you can include the following keys:

- `saml:namequalifier`. This contains a hash value that represents the combination of the `saml:doc` and `saml:iss` values. It is used as a namespace qualifier; the combination of `saml:namequalifier` and `saml:sub` uniquely identifies a user.
- `saml:sub`. As described in the previous list.
- `saml:sub_type`. As described in the previous list.

For more information about using these keys, see Creating Temporary Security Credentials for SAML Federation in the *Using Temporary Security Credentials* guide.

Condition Types

These are the types of conditions you can specify:

- String
- Numeric
- Date and time
- Boolean
- Binary
- IP address
- Amazon Resource Name (ARN). (This value is available for only some services.)
- ...IfExists
- Existence of condition keys

String Conditions

String conditions let you restrict access based on comparing a key to a string value.

Condition	Description
`StringEquals`	Exact matching, case sensitive
`StringNotEquals`	Negated matching
`StringEqualsIgnoreCase`	Exact matching, ignoring case
`StringNotEqualsIgnoreCase`	Negated matching, ignoring case
`StringLike`	Case-sensitive matching. The values can include a multi-character match wildcard (*) or a single-character match wildcard (?) anywhere in the string. **Note** If a key contains multiple values, `StringLike` can be qualified with set operators—`ForAllValues:StringLike` and `ForAnyValues:StringLike`. For more information, see Creating a Condition That Tests Multiple Key Values (Set Operations) (p. 238).
`StringNotLike`	Negated case-sensitive matching. The values can include a multi-character match wildcard (*) or a single-character match wildcard (?) anywhere in the string.

For example, the following statement uses the `StringEquals` condition with the `aws:UserAgent` key to specify that the request must include a specific value in its user agent header.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "iam:*AccessKey*",
    "Resource": "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:user/*",
   "Condition": {"StringEquals": {"aws:UserAgent": "Example Corp Java Client"}}

  }
}
```

The following example uses string matching with a policy variable (p. 232) to create a policy that lets an IAM user use the Amazon S3 console to manage his or her own "home directory" in an Amazon S3 bucket.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListAllMyBuckets",
        "s3:GetBucketLocation"
      ],
      "Resource": "arn:aws:s3:::*"
    },
    {
      "Effect": "Allow",
      "Action": "s3:ListBucket",
      "Resource": "arn:aws:s3:::BUCKET-NAME",
      "Condition": {"StringLike": {"s3:prefix": [
        "",
        "home/",
        "home/${aws:username}/"
      ]}}
    },
    {
      "Effect": "Allow",
      "Action": "s3:*",
      "Resource": [
        "arn:aws:s3:::BUCKET-NAME/home/${aws:username}",
        "arn:aws:s3:::BUCKET-NAME/home/${aws:username}/*"
      ]
    }
  ]
}
```

For an example of a policy that shows how to use the `Condition` element to restrict access to resources based on an application ID and a user ID for web identity federation, see Allow Users Signed In Using Amazon Cognito to Access their Own Amazon S3 Folder (p. 205).

Numeric Conditions

Numeric conditions let you restrict access based on comparing a key to an integer or decimal value.

Condition	Description
`NumericEquals`	Matching
`NumericNotEquals`	Negated matching
`NumericLessThan`	"Less than" matching
`NumericLessThanEquals`	"Less than or equals" matching
`NumericGreaterThan`	"Greater than" matching
`NumericGreaterThanEquals`	"Greater than or equals" matching

For example, the following statement uses the `NumericLessThanEquals` condition with the `s3:max-keys` key to specify that the requester can list up to 10 objects in `example_bucket` at a time.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "s3:ListBucket",
    "Resource": "arn:aws:s3:::example_bucket",
    "Condition": {"NumericLessThanEquals": {"s3:max-keys": "10"}}
  }
}
```

Date Conditions

Date conditions let you restrict access based on comparing a key to a date/time value. You use these conditions with the `aws:CurrentTime` key or `aws:EpochTime` keys. You must specify date/time values with one of the W3C implementations of the ISO 8601 date formats or in epoch (UNIX) time.

> **Note**
> Wildcards are not permitted for date conditions.

Condition	Description
`DateEquals`	Matching a specific date
`DateNotEquals`	Negated matching
`DateLessThan`	Matching before a specific date and time
`DateLessThanEquals`	Matching at or before a specific date and time
`DateGreaterThan`	Matching after a specific a date and time
`DateGreaterThanEquals`	Matching at or after a specific date and time

For example, the following statement uses the `DateLessThan` condition with the `aws:CurrentTime` key to specify that the request must be received before June 30, 2013.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "iam:*AccessKey*",
    "Resource": "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:user/*",
    "Condition": {"DateLessThan": {"aws:CurrentTime": "2013-06-30T00:00:00Z"}}

  }
}
```

Boolean Conditions

Boolean conditions let you restrict access based on comparing a key to "true" or "false."

Condition	Description
`Bool`	Boolean matching

For example, the following statement uses the `Bool` condition with the `aws:SecureTransport` key to specify that the request must use SSL.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "iam:*AccessKey*",
    "Resource": "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:user/*",
    "Condition": {"Bool": {"aws:SecureTransport": "true"}}
  }
}
```

Binary Comparison

The `BinaryEquals` condition can be useful for testing key values that are in binary format. It compares the value of the specified key byte for byte to a value in the policy that uses base-64 to represent the binary value.

```
"Condition" : {
  "BinaryEquals": {
    "key" : "QmluYXJ5VmFsdWVJbkJhc2U2NA=="
  }
}
```

IP Address

IP address conditions let you restrict access based on comparing a key to an IP address or range of IP addresses. You use these with the `aws:SourceIp` key. The value must be in the standard CIDR format (for example, 203.0.113.0/24). For more information, see RFC 4632.

Condition	Description
`IpAddress`	The specified IP address or range
`NotIpAddress`	All IP addresses except the specified IP address or range

For example, the following statement uses the `IpAddress` condition with the `aws:SourceIp` key to specify that the request must come from the IP range 203.0.113.0 to 203.0.113.255.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "iam:*AccessKey*",
    "Resource": "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:user/*",
    "Condition": {"IpAddress": {"aws:SourceIp": "203.0.113.0/24"}}
  }
}
```

The `aws:SourceIp` condition key resolves to the IP address that the request originates from. If the requests originates from an Amazon EC2 instance, `aws:SourceIp` evaluates to the instance's public IP address. If the request originates from an AWS service that makes calls to AWS on your behalf, like Amazon Elastic MapReduce, AWS Elastic Beanstalk, AWS CloudFormation, and Tag Editor, `aws:SourceIp` resolves to the IP address of that service. For these types of services, we recommend that you don't use the `aws:SourceIp` condition.

Amazon Resource Name (ARN)

Amazon Resource Name (ARN) conditions let you restrict access based on comparing a key to an ARN. The ARN is considered a string. This value is available for only some services; not all services support request values that can be compared as ARNs.

Condition	Description
`ArnEquals`	Matching for ARN
`ArnNotEquals`	Negated matching for ARN
`ArnLike`	Case-insensitive matching of the ARN. Each of the six colon-delimited components of the ARN is checked separately and each can include a multi-character match wildcard (*) or a single-character match wildcard (?).
`ArnNotLike`	Negated case-insensitive matching of the ARN. The values can include a multi-character match wildcard (*) or a single-character match wildcard (?) anywhere in the string.

The following example shows a policy you need to attach to any queue that you want Amazon SNS to send messages to. It gives Amazon SNS permission to send messages to the queue (or queues) of your choice, but only if the service is sending the messages on behalf of a particular Amazon SNS topic (or topics). You specify the queue in the `Resource` field, and the Amazon SNS topic as the value for the `SourceArn` key.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Principal": {"AWS": "ACCOUNT-ID-WITHOUT-HYPHENS"},
    "Action": "SQS:SendMessage",
    "Resource": "arn:aws:sqs:REGION:ACCOUNT-ID-WITHOUT-HYPHENS:QUEUE-ID",
    "Condition": {"ArnEquals": {"aws:SourceArn": "arn:aws:sns:REGION:ACCOUNT-
ID-WITHOUT-HYPHENS:TOPIC-ID"}}
  }
}
```

...IfExists Conditions

You can add `IfExists` at the end of any condition except the `Null` condition—for example, `StringEqualsIfExists`. You do this if you mean "If the policy key is present in the context of the request, process the key as specified in the policy. If the key is not present, I don't care."

Behavior of Key Checks When the Key Does Not Exist

Many context keys describe information about a certain type of resource and only exist when that type of resource is being accessed. These context keys are not present on other types of resources. This doesn't cause an issue when the policy statement applies to only one type of resource. However, there are cases where a single statement can apply to multiple types of resources, such as when the policy statement references actions from multiple services or when a given action within a service accesses several different resource types within the same service. In such cases, including a context key that applies to only one of the resources in the policy statement can cause the condition in the policy statement to fail such that the statement's `"Effect"` does not apply. For example, consider the following policy example:

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "ec2:RunInstances",
    "Resource": "*",
    "Condition": {"StringLike": {"ec2:InstanceType": [
      "t1.*",
      "t2.*",
      "m3.*"
    ]}}
  }
}
```

The intent of the preceding policy is to enable the user to launch any instance that is type t1, t2 or m3. However, launching an instance actually requires accessing many resources in addition to the instance itself; for example, images, key pairs, security groups, etc. The entire statement is evaluated against every resource that is required to launch the instance. These additional resources do not have the `ec2:InstanceType` context key, so the `StringLike` check fails, and the user is not granted the ability to launch *any* instance type. To address this, use the `StringLikeIfExists` condition instead. This way, the test only happens if the context key exists. If the key does not exist in the context of the current request, the condition check does not occur and returns a `true` value as if it had passed the test.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "ec2:RunInstances",
    "Resource": "*",
    "Condition": {"StringLikeIfExists": {"ec2:InstanceType": [
      "t1.*",
      "t2.*",
      "m3.*"
    ]}}
  }
}
```

Existence of Condition Keys

Use a `Null` condition to check if a condition key is present at the time of authorization. In the policy statement, use either `true` (the key doesn't exist) or `false` (the key exists and its value is not null).

For example, you can use this condition to determine whether a user is using their own credentials for the operation or temporary credentials. If the user is using temporary credentials, then the key `aws:TokenIssueTime` exists and has a value. The following example shows a condition that states that the user must not be using temporary credentials (the key must not exist) for the user to use the Amazon EC2 API.

```
{
  "Version": "2012-10-17",
  "Statement":{
      "Action":"ec2:*",
      "Effect":"Allow",
      "Resource":"*",
      "Condition":{"Null":{"aws:TokenIssueTime":"true"}}
  }
}
```

Supported Data Types

This section lists the data types that are supported when you specify values in IAM policies. The policy language doesn't support all types for each policy element; for information about each element, see the preceding sections.

- Strings
- Numbers (Ints and Floats)
- Boolean
- Null
- Lists
- Maps
- Structs (which are just nested Maps)

The following table maps each data type to the serialization. Note that all policies must be in UTF-8. For information about the JSON data types, go to RFC 4627.

Type	JSON
String	String
Integer	Number
Float	Number
Boolean	true false
Null	null
Date	String adhering to the W3C Profile of ISO 8601
IpAddress	String adhering to RFC 4632
List	Array
Object	Object

IAM Policy Variables Overview

Topics

Introduction

In IAM policies, you can provide a name for the specific resources that want to control access to. The following policy allows the user who gets these permissions to programmatically list, read, and write objects with a prefix `David` in the Amazon S3 bucket `mybucket`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": ["s3:ListBucket"],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::mybucket"],
      "Condition": {"StringLike": {"s3:prefix": ["David/*"]}}
    },
    {
      "Action": [
        "s3:GetObject",
        "s3:PutObject"
      ],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::mybucket/David/*"]
    }
  ]
}
```

In some cases, you might not know the exact name of the resource when you write the policy. For example, you might want to allow each user to have his or her own objects in an Amazon S3 bucket, as in the previous example. However, instead of creating a separate policy for each user that specifies the user's name as part of the resource, you want to create a single group policy that works for any user in that group.

You can do this by using *policy variables*, a feature that lets you specify placeholders in a policy. When the policy is evaluated, the policy variables are replaced with values that come from the request itself.

The following example shows a policy for an Amazon S3 bucket that uses a policy variable.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": ["s3:ListBucket"],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::mybucket"],
      "Condition": {"StringLike": {"s3:prefix": ["${aws:username}/*"]}}
    },
    {
      "Action": [
        "s3:GetObject",
        "s3:PutObject"
      ],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::mybucket/${aws:username}/*"]
    }
  ]
}
```

When this policy is evaluated, IAM uses the friendly name (p. 9) of the authenticated user in place of the variable `${aws:username}`. This means that a single policy applied to a group of users can be used to control access to a bucket by using the user name as part of the resource.

The variable is marked using a $ prefix followed by a pair of curly braces ({ }). Inside the ${ } characters, you can include the name of the value from the request that you want to use in the policy. The values you can use are discussed later on this page.

Note
In order to use policy variables, you must include the `Version` element in a statement, and the version must be set to the value `2012-10-17`. Earlier versions of the policy language don't support policy variables. If you don't include the `Version` element and set it to this version date, variables like `${aws:username}` are treated as literal strings in the policy.

You can use policy variables in a similar way to allow each user to manage his or her own access keys. A policy that allows a user to programmatically change the access key for user `David` looks like this:

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Action": ["iam:*AccessKey*"],
    "Effect": "Allow",
    "Resource": ["arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:user/David"]
  }]
}
```

If this policy is attached to user `David`, that user can change his own access key. As with the policies for accessing user-specific Amazon S3 objects, you'd have to create a separate policy for each user that includes the user's name, and then attach each policy to the individual users.

By using a policy variable, you can create a policy like this:

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Action": ["iam:*AccessKey*"],
    "Effect": "Allow",
   "Resource": ["arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:user/${aws:username}"]

  }]
}
```

When you use a policy variable for the user name like this, you don't have to have a separate policy for each individual user. Instead, you can attach this new policy to an IAM group that includes everyone who should be allowed to manage their own access keys. When a user makes a request to modify his or her access key, IAM substitutes the user name from the current request for the `${aws:username}` variable and evaluates the policy.

Where You Can Use Policy Variables

You can use policy variables in the `Resource` element and in string comparisons in the `Condition` element.

Resource Element

A policy variable can appear as the last part of the ARN that identifies a resource. The following policy might be attached to a group. It gives each of the users in the group full programmatic access to a user-specific object (their own "home directory") in Amazon S3.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": ["s3:ListBucket"],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::mybucket"],
      "Condition": {"StringLike": {"s3:prefix": ["${aws:username}/*"]}}
    },
    {
      "Action": [
        "s3:GetObject",
        "s3:PutObject"
      ],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::mybucket/${aws:username}/*"]
    }
  ]
}
```

Note
This example uses the `aws:username` key, which returns the user's friendly name (like "Adele" or "David"). Under some circumstances, you might want to use the `aws:userid` key instead, which is a globally unique value. For more information, see Unique IDs (p. 12).

The following policy might be used for an IAM group. It gives users in that group the ability to create, use, and delete queues that have their names and that are in the us-west-2 region.

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": "sqs:*",
    "Resource": "arn:aws:sqs:us-west-2:*:${aws:username}-queue"
  }]
}
```

Condition Element

A policy variable can also be used for `Condition` values in any condition that involves the string operators (`StringEquals`, `StringLike`, `StringNotLike`, etc.) or the ARN operators (`ArnEquals`, `ArnLike`, etc.). The following Amazon SNS topic policy gives users in AWS account `999999999999` the ability to manage (perform all actions for) the topic only if the URL matches their AWS user name.

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Principal": {"AWS": "999999999999"},
    "Effect": "Allow",
    "Action": "sns:*",
   "Condition": {"StringLike": {"sns:endpoint": "https://example.com/${aws:user
name}/*"}}
  }]
}
```

Request Information That You Can Use for Policy Variables

The values that can be substituted for policy variables must come from the current request context (p. 243).

Information Available in All Requests

Policies contain keys whose values you can use as policy variables. (Under some circumstances, the keys do not contain a value—see the information that follows this list.)

- `aws:CurrentTime` (for date/time conditions)
- `aws:EpochTime` (the date in epoch or UNIX time, for use with date/time conditions)
- `aws:TokenIssueTime` (the date/time that temporary security credentials were issued, for use with date/time conditions)

  Note
  This key is only available in requests that are signed using temporary security credentials. For more information about temporary security credentials, see *Using Temporary Security Credentials*.
- `aws:principaltype` (a value that indicates whether the principal is an account, user, federated, or assumed role—see the explanation that follows)
- `aws:SecureTransport` (Boolean representing whether the request was sent using SSL)
- `aws:SourceIp` (the requester's IP address, for use with IP address conditions)
- `aws:UserAgent` (information about the requester's client application, for use with string conditions)
- `aws:userid` (the unique ID for the current user—see the following chart)
- `aws:username` (the friendly name (p. 9) of the current user—see the following chart)

Important
The names of condition keys are case sensitive.

The values for `aws:username`, `aws:userid`, and `aws:principaltype` depend on what type of principal initiated the request—whether the request was made using the credentials of an AWS account, an IAM user, an IAM role, and so on. The following list shows values for these keys for different types of principal.

- **AWS Account**
  - `aws:username`: (not present)
  - `aws:userid`: AWS account ID
  - `aws:principaltype`: `Account`
- **IAM user**
  - `aws:username`: *`IAM-user-name`*
  - `aws:userid`: unique ID (p. 12)
  - `aws:principaltype`: `User`
- **Federated user**
  - `aws:username`: (not present)
  - `aws:userid`: *`account`*:*`caller-specified-name`*
  - `aws:principaltype`: `FederatedUser`
- **Web federated user and SAML federated user**

  Note
  For information about policy keys that are available when you use web identity federation, see Identifying Apps and Users with Web Identity Federation in the *Using Temporary Security Credentials* guide.
  - `aws:username`: (not present)

  - `aws:userid`: (not present)
  - `aws:principaltype`: `AssumedRole`
- **Assumed role**
  - `aws:username`: (not present)
  - `aws:userid`: *`role-id`*:*`caller-specified-role-name`*
  - `aws:principaltype`: `Assumed role`
- **Role assigned to Amazon EC2 instance**
  - `aws:username`: (not present)
  - `aws:userid`: *`role-id`*:*`ec2-instance-id`*
  - `aws:principaltype`: `Assumed role`
- **Anonymous caller** (Amazon SQS Amazon SNS and Amazon S3 only)
  - `aws:username`: (not present)
  - `aws:userid`: (not present)
  - `aws:principaltype`: `Anonymous`

In this list:

- *not present* means that the value is not in the current request information, and any attempt to match it causes the request to be denied.
- *`role-id`* is a unique identifier assigned to each role at creation. You can display the role ID with the AWS CLI command: `aws iam get-role --role-name` *`rolename`*
- *`caller-specified-name`* and *`caller-specified-role-name`* are names that are passed by the calling process (e.g. application or service) when it makes a call to get temporary credentials.
- *`ec2-instance-id`* is a value assigned to the instance when it is launched and appears on the **Instances** page of the Amazon EC2 console. You can also display the instance ID by running the AWS CLI command: `aws ec2 describe-instances`

Information Available in Requests for Federated Users

Federated users are users who are authenticated using a system other than IAM. For example, a company might have an application for use in-house that makes calls to AWS. It might be impractical to give an IAM identity to every corporate user who uses the application. Instead, the company might use a proxy (middle-tier) application that has a single IAM identity, or the company might use a SAML identity provider (IdP). The proxy application or SAML IdP authenticates individual users using the corporate network. A proxy application can then use its IAM identity to get temporary security credentials for individual users; a SAML IdP can in effect exchange identity information for AWS temporary security credentials. The temporary credentials can then be used to access AWS resources.

Similarly, you might create an app for a mobile device in which the app needs to access AWS resources. In that case, you might use *web identity federation*, where the app authenticates the user using a well-known identity provider like Login with Amazon, Amazon Cognito, Facebook, or Google. The app can then use the user's authentication information from these providers to get temporary security credentials for accessing AWS resources.

The recommended way to use web identity federation is by taking advantage of Amazon Cognito and the AWS mobile SDKs. For more information, see the following:

- Amazon Cognito Overview in the *AWS SDK for Android Developer Guide*
- Amazon Cognito Overview in the *AWS SDK for iOS Developer Guide*
- Scenarios for Granting Temporary Access in the *Using Temporary Security Credentials* guide.

Service-Specific Information

Requests can also include service-specific keys and values in its request context. Examples include the following:

- `s3:prefix`
- `s3:max-keys`
- `s3:x-amz-acl`
- `sns:Endpoint`
- `sns:Protocol`

For information about service-specific keys that you can use to get values for policy variables, refer to the documentation for the individual services. For example, see the following topics:

- Bucket Keys in Amazon S3 Policies in the *Amazon Simple Storage Service Developer Guide*.
- Amazon SNS Keys in the *Amazon Simple Notification Service Developer Guide*.

Special characters

There are a few special predefined policy variables that have fixed values that enable you to represent characters that otherwise have special meaning. If these special characters are part of the string you are trying to match and you inserted them literally they would be misinterpreted. For example, inserting an * asterisk in the string would be interpreted as a wildcard, matching any characters, instead of as a literal *. In these cases, you can use the following predefined policy variables:

- **${*}** - use where you need an * asterisk character.
- **${?}** - use where you need a ? question mark character.
- **${$}** - use where you need a $ dollar sign character.

These predefined policy variables can be used in any string where you can use regular policy variables.

For More Information

For more information about policies, see the following:

- Overview of AWS IAM Permissions (p. 171)
- Example Policies for Administering AWS Resources (p. 203)
- IAM Policy Elements Reference (p. 209)
- IAM Policy Evaluation Logic (p. 243)
- Creating Temporary Security Credentials for Mobile Apps Using Third-Party Identity Providers in the *Using Temporary Security Credentials* guide.

Creating a Condition That Tests Multiple Key Values (Set Operations)

This topic discusses how to create a policy condition that lets you test multiple values for a single key in a request. In effect, you can create a condition that uses set operations. This type of condition is useful for creating policies that allow fine-grained control for DynamoDB tables (for example, allowing or denying access to particular attributes).

To create this type of condition, you can use the `ForAllValues` or `ForAnyValue` qualifier with the condition operator. These qualifiers add set-operation functionality to the condition operator so that you can test multiple request values against multiple condition values.

Contents

Introduction

In some situations, you need to create a policy in which you test multiple values in a request against one or more values that you specify in the policy. A good example is a request for attributes from an Amazon DynamoDB table. Imagine an Amazon DynamoDB table named `Thread` that is used to store information about threads in a technical support forum. The table might have attributes like `ID`, `UserName`, `PostDateTime`, `Message`, and `Tags` (among others).

```
{
  ID=101
  UserName=Bob
  PostDateTime=20130930T231548Z
  Message="A good resource for this question is http://aws.amazon.com/document
ation/"
  Tags=["AWS", "Database", "Security"]
}
```

You might want to create a policy that allows users to see only some attributes—for example, maybe you'll let these users see only `PostDateTime`, `Message`, and `Tags`. If the user's request contains any of these attributes, it is allowed; but if the request contains other attributes (for example, `ID`), the request is denied. Logically speaking, you want to create a list of allowed attributes (`PostDateTime`, `Message`, `Tags`) and indicate in the policy that all of the user's requested attributes must be within this list.

Or you might want to make sure that users are explicitly forbidden to include some attributes in a request, such as the `ID` and `UserName` attributes. For example, you might exclude attributes when the user is updating the DynamoDB table, because an update (`PUT` operation) should not change certain attributes. In that case, you create a list of forbidden attributes (`ID`, `UserName`), and if any of the user's requested attributes match any of the forbidden attributes, the request is denied.

To support these scenarios, you can use the following modifiers to a condition operator:

- `ForAnyValue` – The condition returns true if any one of the key values in the request matches any one of the condition values in the policy.
- `ForAllValues` – The condition returns true if there's a match between every one of the specified key values in the request and at least one condition value in the policy.

For information about how set operators are used in DynamoDB to implement fine-grained access to individual data items and attributes, see Fine-Grained Access Control for DynamoDB in the *Amazon DynamoDB Developer Guide* guide.

Examples of Condition Set Operators

The following example policy shows how to use the `ForAllValues` qualifier with the `StringLike` condition operator. The condition allows a user to request only the attributes `PostDateTime`, `Message`, or `Tags` from the DynamoDB table named `Thread`.

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": "GetItem",
    "Resource": "arn:aws:dynamodb:REGION:ACCOUNT-ID-WITHOUT-HY
PHENS:table/Thread",
    "Condition": {"ForAllValues:StringLike": {"dynamodb:requestedAttributes":
[
      "PostDateTime",
      "Message",
      "Tags"
    ]}}
  }]
}
```

If the user makes a request to DynamoDB to get the attributes `PostDateTime` and `Message` from the `Threads` table, the request is allowed, because the user's requested attributes all match values specified in the policy. However, if the user's request includes the `ID` attribute, the request fails, because `ID` is not within the list of allowed attributes, and the `ForAllValues` qualifier requires all requested values to be listed in the policy.

The following example shows how to use the `ForAnyValue` qualifier as part of a policy that denies access to the `ID` and `PostDateTime` attributes if the user tries to perform the `PutItem` action—that is, if the user tries to update either of those attributes. Notice that the `Effect` element is set to `Deny`.

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Deny",
    "Action": "PutItem",
    "Resource": "arn:aws:dynamodb:REGION:ACCOUNT-ID-WITHOUT-HY
PHENS:table/Thread",
    "Condition": {"ForAnyValue:StringLike": {"dynamodb:requestedAttributes": [

      "ID",
      "PostDateTime"
    ]}}
  }]
}
```

Imagine that the user makes a request to update the `PostDateTime` and `Message` attributes. The `ForAnyValue` qualifier determines whether any of the requested attributes appear in the list in the policy. In this case, there is one match (`PostDateTime`), so the condition is true. Assuming the other values in the request also match (for example, the resource), the overall policy evaluation returns true. Because the policy's effect is `Deny`, the request is denied.

Imagine the user instead makes a request to perform `PutItem` with just the `UserName` attribute. None of the attributes in the request (just `UserName`) match any of attributes listed in the policy (`ID`, `PostDateTime`). The condition returns false, so the effect of the policy (`Deny`) is also false, and the request is not denied by this policy. (For the request to succeed, it must be explicitly allowed by a different policy. It is not explicitly denied by this policy, but all requests are implicitly denied.)

Evaluation Logic for Condition Set Operators

This section discusses the specifics of the evaluation logic used with the `ForAllValues` and `ForAnyValue` qualifiers. The following table illustrates possible keys that might be included in a request (`PostDateTime` and `UserName`) and a policy condition that includes the values `PostDateTime`, `Message`, and `Tags`.

Key (in the request)	Condition Value (in the policy)
`PostDateTime`	`PostDateTime`
`UserName`	`Message`
	`Tags`

The evaluation for the combination is this:

`PostDateTime` matches `PostDateTime`?
`PostDateTime` matches `Message`?
`PostDateTime` matches `Tags`?
`UserName` matches `PostDateTime`?
`UserName` matches `Message`?
`UserName` matches `Tags`?

The result of the condition depends on which modifier is used with the policy condition:

- `ForAllValues`. If there is at least one match between every key in the request (`PostDateTime` or `UserName`) and at least one condition value in the policy (`PostDateTime`, `Message`, `Tags`), the condition returns true. Stated another way, in order for the condition to be true, (`PostDateTime` must equal `PostDateTime`, `Message`, or `Tags`) *and* (`UserName` must equal `PostDateTime`, `Message`, or `Tags`).
- `ForAnyValue`. If any one of the six possible combinations returns true, the condition returns true.

The following policy includes a `ForAllValues` qualifier:

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": "GetItem",
    "Resource": "arn:aws:dynamodb:REGION:ACCOUNT-ID-WITHOUT-HY
PHENS:table/Thread",
    "Condition": {"ForAllValues:StringLike": {"dynamodb:requestedAttributes":
[
      "PostDateTime",
      "Message",
      "Tags"
    ]}}
  }]
}
```

Suppose that the user makes a request to DynamoDB to get the attributes `PostDateTime` and `UserName`. The keys in the request and condition values in the policy are these:

Key (in the request)	Condition Value (in the policy)
`PostDateTime`	`PostDateTime`
`UserName`	`Message`
	`Tags`

The evaluation for the combination is this:

`PostDateTime` matches `PostDateTime`?	`True`
`PostDateTime` matches `Message`?	`False`
`PostDateTime` matches `Tags`?	`False`
`UserName` matches `PostDateTime`?	`False`
`UserName` matches `Message`?	`False`
`UserName` matches `Tags`?	`False`

The policy includes the `ForAllValues` condition, meaning that there must be at least one match for `PostDateTime` and one match for `UserName`. There's no match for `UserName`, so the condition returns false, and the policy does not allow the request.

The following policy includes a `ForAnyValue` qualifier:

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Deny",
    "Action": "PutItem",
    "Resource": "arn:aws:dynamodb:REGION:ACCOUNT-ID-WITHOUT-HY
PHENS:table/Thread",
    "Condition": {"ForAnyValue:StringLike": {"dynamodb:requestedAttributes": [

      "ID",
      "PostDateTime"
    ]}}
  }]
}
```

Notice that the policy includes `"Effect":"Deny"` and the action is `PutItem`. Imagine that the user makes a `PutItem` request that includes the attributes `UserName`, `Message`, and `PostDateTime`. The evaluation is this:

`UserName` matches `ID`?	`False`
`UserName` matches `PostDateTime`?	`False`
`Messages` matches `ID`?	`False`

`Message` matches `PostDateTime`?	`False`
`PostDateTime` matches `ID`?	`False`
`PostDateTime` matches `PostDateTime`?	`True`

With the modifier `ForAnyValue`, if any one of these tests returns true, the condition returns true. The last test returns true, so the condition is true; because the `Effect` element is set to `Deny`, the request is denied.

Note
If the key values in the request resolve to an empty data set (for example, an empty string), `ForAllValues` returns true, and `ForAnyValue` returns false.

IAM Policy Evaluation Logic

Topics

Policy Evaluation Basics

When an AWS service receives a request, the request is first authenticated using information about the access key ID and signature. (A few services, like Amazon S3, allow requests from anonymous users.) If the request passes authentication, AWS then determines whether the requester is authorized to perform the action represented by the request.

Requests that are made using the credentials of the AWS account owner (the root credentials) for resources in that account are allowed. However, if the request is made using the credentials of an IAM user, or if the request is signed using temporary credentials that are granted by AWS STS, AWS uses the permissions defined in one or more IAM policies to determine whether the user's request is authorized.

Note
Amazon S3 supports Access Control Lists (ACLs) and resource-level policies for buckets and objects. The permissions established using ACLs and bucket-level policies can affect what actions the root owner is allowed to perform on a bucket. For more information, see Guidelines for Using the Available Access Policy Options in the *Amazon Simple Storage Service Developer Guide*.

The Request Context

When AWS authorizes a request, information about the request is assembled from several sources:

- Principal (the requester), which is determined based on the secret access key. This might represent the root user, an IAM user, a federated user (via STS), or an assumed role, and includes the aggregate permissions that are associated with that principal.
- Environment data, such as the IP address, user agent, SSL enabled, the time of day, etc. This information is determined from the request.
- Resource data, which pertains to information that is part of the resource being requested. This can include information such as a DynamoDB table name, a tag on an Amazon EC2 instance, etc.

This information is gathered into a *request context*, which is a collection of information that's derived from the request. During evaluation, AWS uses values from the request context to determine whether to allow or deny the request. For example, does the action in the request context match an action in the `Action` element? If not, the request is denied. Similarly, does the resource in the request context match one of the resources in the `Resource` element? If not, the request is denied.

This is also how the keys work that you can use in the `Condition` element. For example, for the following policy fragment, AWS uses the date and time from the current request context for the `aws:CurrentTime` key and then performs the `DateGreaterThan` and `DateLessThan` comparisons.

```
"Condition" :  {
      "DateGreaterThan" : {
         "aws:CurrentTime" : "2013-08-16T12:00:00Z"
       },
      "DateLessThan": {
         "aws:CurrentTime" : "2013-08-16T15:00:00Z"
       }
}
```

Policy variables like `${aws:username}` also work like this. In the following policy fragment, AWS gets the user name from the request context and uses it in the policy at the place where the `${aws:username}` occurs.

```
  "Resource": [
        "arn:aws:s3:::mybucket/${aws:username}/*"
      ]
```

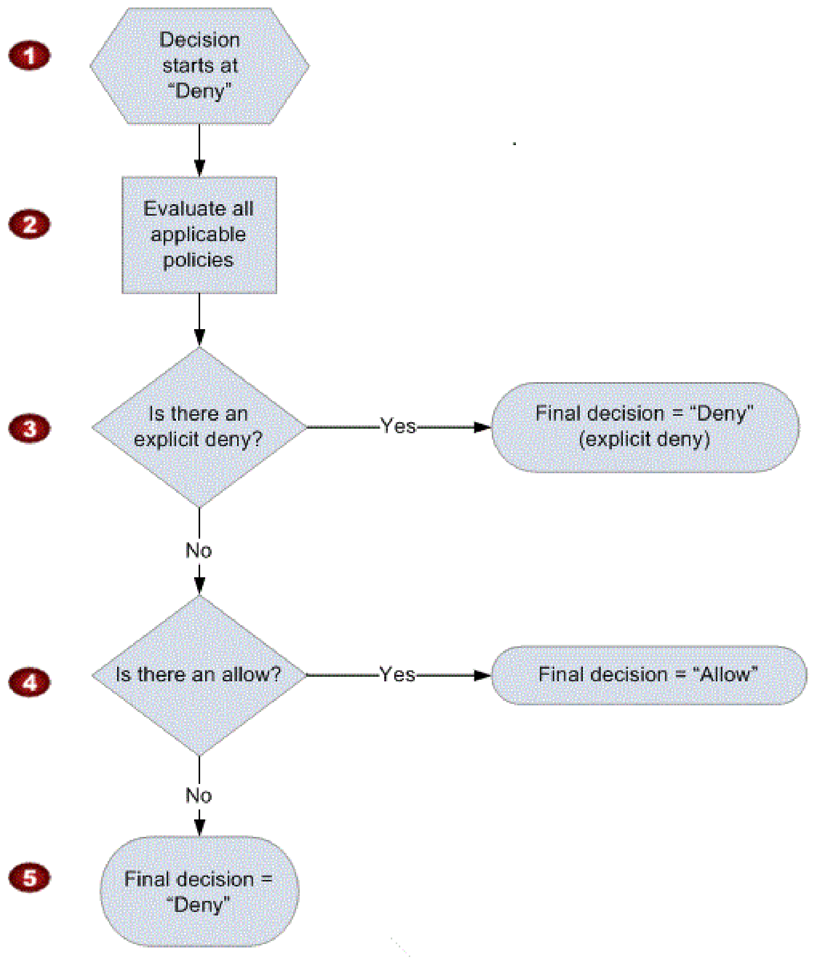
Determining Whether a Request is Allowed or Denied

When a request is made, the AWS service decides whether a given request should be allowed or denied. The evaluation logic follows these rules:

- By default, all requests are denied. (In general, requests made using the account credentials for resources in the account are always allowed.)
- An explicit allow overrides this default.
- An explicit deny overrides any allows.

The order in which the policies are evaluated has no effect on the outcome of the evaluation. All policies are evaluated, and the result is always that the request is either allowed or denied.

The following flow chart provides details about how the decision is made.


1
Decision starts at "Deny"
2
Evaluate all applicable policies
3
Is there an explicit deny?
Yes
Final decision = "Deny" (explicit deny)
No
4
Is there an allow?
Yes
Final decision = "Allow"
No
5
Final decision = "Deny"

1. The decision starts by assuming that the request will be denied.
2. The enforcement code evaluates all user-based and resource-based policies that are applicable to the request (based on the resource, principal, action, and conditions).

   The order in which the enforcement code evaluates the policies is not important.
3. In all those policies, the enforcement code looks for an explicit deny instruction that would apply to the request.

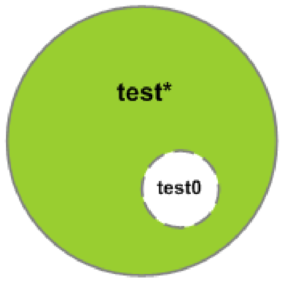
   If the code finds even one explicit deny that applies, the code returns a decision of "deny" and the process is finished.
4. If no explicit deny is found, the code looks for any "allow" instructions that would apply to the request.

   If it finds even one explicit allow, the code returns a decision of "allow" and the service continues to process the request.
5. If no allow is found, the final decision is "deny."

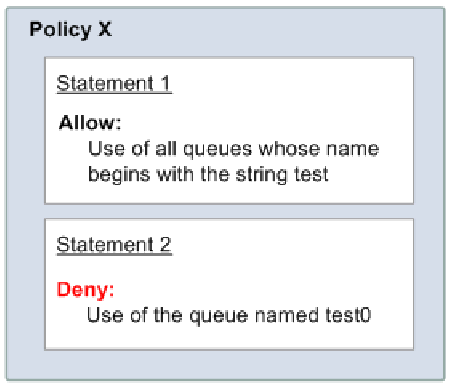
The following example illustrates how you can use an explicit deny to override a broad policy that allows access to a wide set of resources. Let's say that you give an IAM group permissions to use any Amazon SQS queues in your AWS account whose names begin with the string `test`.

The following diagram represents that set of queues.

Let's say that you have a queue called `test0` that you want to remove the group's access to. The following diagram represents that set of queues.



You could add another policy to the group, or another statement to the existing policy, that explicitly denies the group's access to the `test0` queue. The group would still have access to all other queues whose names begin with the string `test`. The following diagram illustrates those two statements in the policy.


Policy X
Statement 1
Allow:
Use of all queues whose name begins with the string test
Statement 2
Deny:
Use of the queue named test0

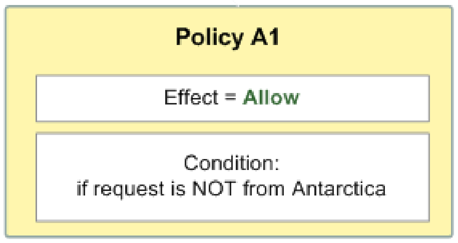
When any user in the group makes a request to use the queue `test0`, the explicit deny overrides the allow, and the user is denied access to the queue.

The Difference Between Denying by Default and Explicit Deny

A policy results in a deny if the policy doesn't directly apply to the request. For example, if a user makes a request to use Amazon SQS, but the only policy that applies to the user states that the user can use Amazon S3, the request is denied.

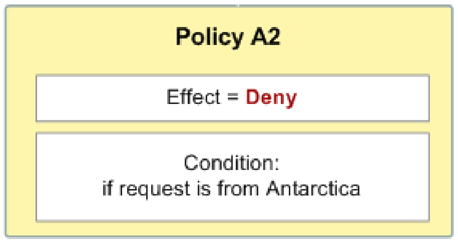
A policy also results in a deny if a condition in a statement isn't met. If all conditions in the statement are met, the policy results in either an allow or an explicit deny, based on the value of the `Effect` element in the policy. Policies don't specify what to do if a condition isn't met, so the result in that case is a deny.

For example, let's say you want to prevent requests that come from Antarctica. You write a policy called Policy A1 that allows a request only if it doesn't come from Antarctica. The following diagram illustrates the policy.



If someone sends a request from the U.S., the condition is met (the request is not from Antarctica). Therefore, the request is allowed. But if someone sends a request from Antarctica, the condition isn't met, and the policy's result is therefore a deny.

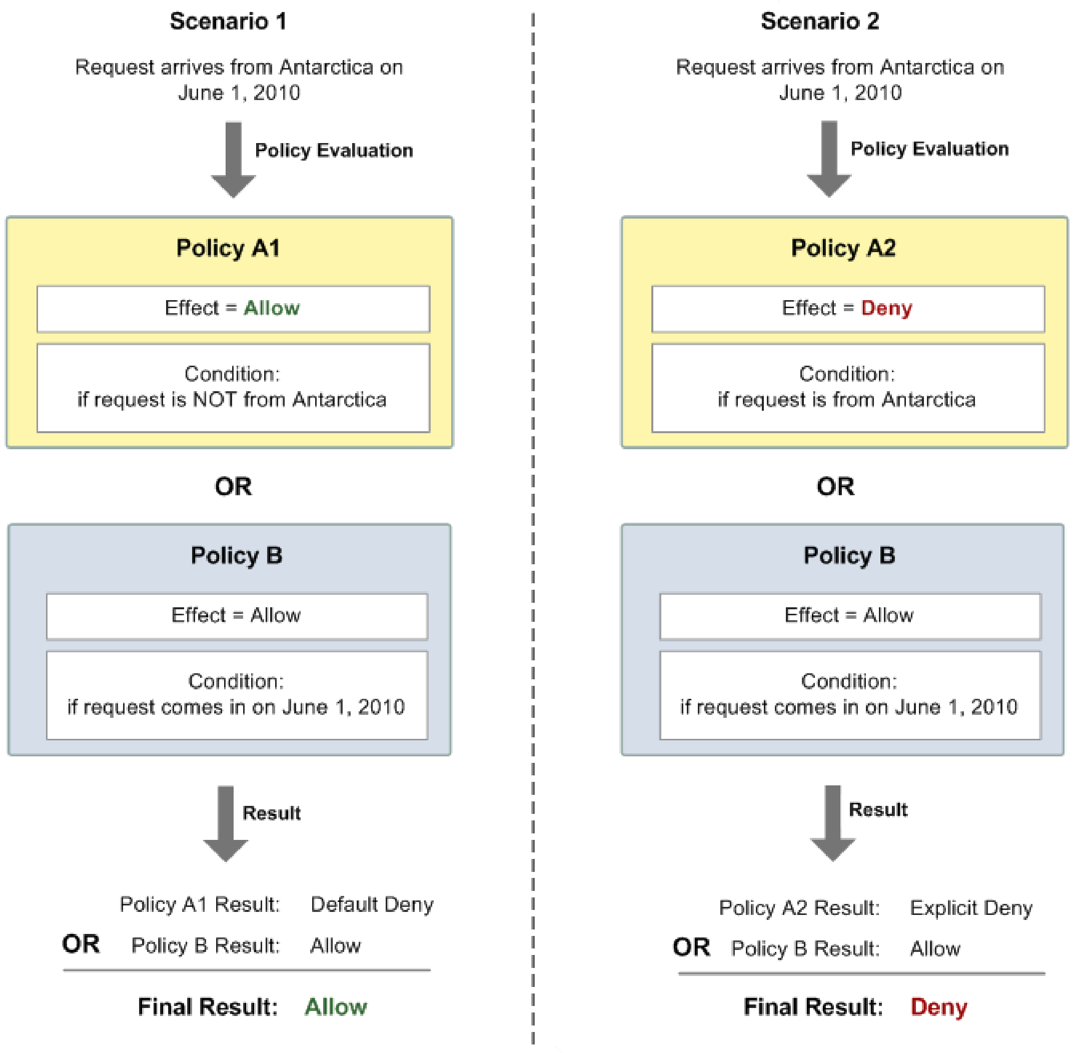
You could *explicitly* deny access from Antarctica by creating a different policy (named Policy A2) as in the following diagram.



If this policy applies to a user who sends a request from Antarctica, the condition is met, and the policy's result is therefore a deny.

The distinction between a request being denied by default and an explicit deny in a policy is important. By default, a request is denied, but this can be overridden by an allow. In contrast, if a policy explicitly denies a request, that deny can't be overridden.

For example, let's say there's another policy that allows requests if they arrive on June 1, 2010. How does this policy affect the outcome when evaluated together with the policy that restricts access from Antarctica? We'll compare the outcome when evaluating the date-based policy (we'll call it Policy B) with the preceding policies (A1 and A2). Scenario 1 evaluates Policy A1 with Policy B, and Scenario 2 evaluates Policy A2 with Policy B. The following figure show the results when a request comes in from Antarctica on June 1, 2010.


Scenario 1
Request arrives from Antarctica on June 1, 2010
Policy Evaluation
Policy A1
Effect = Allow
Condition: if request is NOT from Antarctica
OR
Policy B
Effect = Allow
Condition: if request comes in on June 1, 2010
Result
Policy A1 Result: Default Deny
OR Policy B Result: Allow
Final Result: Allow
Scenario 2
Request arrives from Antarctica on June 1, 2010
Policy Evaluation
Policy A2
Effect = Deny
Condition: if request is from Antarctica
OR
Policy B
Effect = Allow
Condition: if request comes in on June 1, 2010
Result
Policy A2 Result: Explicit Deny
OR Policy B Result: Allow
Final Result: Deny

In Scenario 1, Policy A1 returns a deny because requests are allowed only if they don't come from Antarctica; any other condition (requests that do come from Antarctica) are denied by default. Policy B returns an allow because the request arrives on June 1, 2010. The allow from Policy B overrides the default deny from Policy A1, and the request is therefore allowed.

In Scenario 2, Policy B2 returns an explicit deny, as described earlier in this section. Again, Policy B returns an allow. However, the explicit deny from Policy A2 overrides the allow from Policy B, and the request is therefore denied.

Grammar of the IAM Policy Language

This page presents a formal grammar for the language used to create policies in IAM. We present this grammar so that you can understand how to construct and validate policies.

For examples of policies, see the following topics:

- Overview of IAM Policies (p. 174)
- Example Policies for Administering AWS Resources (p. 203)

- Example Policies for Working in the Amazon EC2 Console and Example Policies for Working With the AWS CLI, the Amazon EC2 CLI, or an AWS SDK in the *Amazon EC2 User Guide for Linux Instances*.
- Bucket Policy Examples and User Policy Examples in the *Amazon Simple Storage Service Developer Guide*.

For examples of policies used in other AWS services, go to the documentation for those services.

Topics

The Policy Language and JSON

Policies are expressed in JSON. When a policy is submitted to IAM, it is first validated to make sure that the JSON syntax is correct. In this document, we do not provide a complete description of what constitutes valid JSON. However, here are some basic JSON rules:

- Whitespace between individual entities is allowed.
- Values are enclosed in quotation marks. Quotation marks are optional for numeric and boolean values.
- Many elements (for example, `action_string_list` and `resource_string_list`) can take a JSON array as a value. Arrays can take one or more values. If more than one value is included, the array is in square brackets (`[` and `]`) and comma-delimited, as in the following example:

  ```
  "Action" : ["ec2:Describe*","ec2:List*"]
  ```
- Basic JSON datatypes (boolean, number, and string) are defined in RFC 7159.

You can use a JSON validator to check the syntax of a policy. You can find a validator online, and many code editors and XML-editing tools include JSON validation features.

Conventions Used in This Grammar

The following conventions are used in this grammar:

- The following characters are JSON tokens and *are* included in policies:

  ```
  { } [ ] " , :
  ```
- The following characters are special characters in the grammar and are *not* included in policies:

  ```
  = < > ( ) |
  ```
- If an element allows multiple values, it is indicated using repeated values, a comma delimiter, and an ellipsis (`...`). Examples:

  ```
  [<action_string>, <action_string>, ...]
  ```

  ```
  <principal_map> = { <principal_map_entry>, <principal_map_entry>, ... }
  ```

  If multiple values are allowed, it is also valid to include only one value. For only one value, the trailing comma must be omitted. If the element takes an array (marked with [and]) but only one value is included, the brackets are optional. Examples:

  ```
  "Action": [<action_string>]
  ```

  ```
  "Action": <action_string>
  ```

- A question mark (?) following an element indicates that the element is optional. Example:

  ```
  <version_block?>
  ```

  However, be sure to refer to the notes that follow the grammar listing for details about optional elements.
- A vertical line (|) between elements indicates alternatives. In the grammar, parentheses define the scope of the alternatives. Example:

  ```
  ("Principal" | "NotPrincipal")
  ```
- Elements that must be literal strings are enclosed in double quotation marks ("). Example:

  ```
  <version_block> = "Version" : ("2008-10-17" | "2012-10-17")
  ```

For additional notes, see Policy Grammar Notes (p. 251) following the grammar listing.

Grammar

The following listing describes the policy language grammar. For conventions used in the listing, see the preceding section. For additional information, see the notes that follow.

Note
This grammar describes policies marked with a version of `2008-10-17` and `2012-10-17`.

```
policy  = {
     <version_block?>
     <id_block?>
     <statement_block>
}

<version_block> = "Version" : ("2008-10-17" | "2012-10-17")

<id_block> = "Id" : <policy_id_string>

<statement_block> = "Statement" : [ <statement>, <statement>, ... ]

<statement> = {
    <sid_block?>,
    <principal_block?>,
    <effect_block>,
    <action_block>,
    <resource_block>,
    <condition_block?>
}

<sid_block> = "Sid" : <sid_string>

<effect_block> = "Effect" : ("Allow" | "Deny")

<principal_block> = ("Principal" | "NotPrincipal") : ("*" | <principal_map>)

<principal_map> = { <principal_map_entry>, <principal_map_entry>, ... }

<principal_map_entry> = ("AWS" | "Federated" | "Service") :
    [<principal_id_string>, <principal_id_string>, ...]

<action_block> = ("Action" | "NotAction") :
    ("*" | [<action_string>, <action_string>, ...])
```

```
<resource_block> = ("Resource" | "NotResource") :
    ("*" | [<resource_string>, <resource_string>, ...])

<condition_block> = "Condition" : { <condition_map> }
<condition_map> {
  <condition_type_string> : { <condition_key_string> : <condition_value_list>
},
  <condition_type_string> : { <condition_key_string> : <condition_value_list>
}, ...
}
<condition_value_list> = [<condition_value>, <condition_value>, ...]
<condition_value> = ("string" | "number" | "Boolean")
```

Policy Grammar Notes

- A single policy can contain an array of statements.
- Policies have a maximum size between 2048 characters and 10,240 characters, depending on what entity the policy is attached to. For more information, see Limitations on IAM Entities (p. 13). Policy size calculations do not include whitespace characters.
- Individual elements must not contain multiple instances of the same key. For example, you cannot include the `Effect` block twice in the same statement.
- Blocks can appear in any order. For example, `version_block` can follow `id_block` in a policy. Similarly, `effect_block`, `principal_block`, `action_block` can appear in any order within a statement.
- The `id_block` is optional in resource-based policies. It must *not* be included in user-based policies.
- The `principal_block` element is required in resource-based policies (for example, in Amazon S3 bucket policies) and in trust policies for IAM roles. It must *not* be included in user-based policies.
- Each string value (`policy_id_string`, `sid_string`, `principal_ID_string`, `action_string`, `resource_string`, `condition_type_string`, `condition_key_string`, and the string version of `condition_value`) can have its own minimum and maximum length restrictions, specific allowed values, or required internal format.

Notes About String Values

This section provides additional information about string values that are used in different elements in a policy.

`action_string`

Consists of a service namespace, a colon, and the name of an action. Action names can include wildcards. Examples:

```
"Action":"ec2:StartInstances"

"Action":[
  "ec2:StartInstances",
  "ec2:StopInstances"
]

"Action":"cloudformation:*"

"Action":"*"

"Action":[
  "s3:Get*",
  "s3:List*"
]
```

`policy_id_string`

Provides a way to include information about the policy as a whole. Some services, such as Amazon SQS and Amazon SNS, use the `Id` element in reserved ways. Unless otherwise restricted by an individual service, policy_id_string can include spaces. Some services require this value to be unique within an AWS account.

> **Note**
> The `id_block` is allowed in resource-based policies, but not in user-based policies.

There is no limit to the length, although this string contributes to the overall length of the policy, which is limited.

```
"Id":"Admin_Policy"

"Id":"cd3ad3d9-2776-4ef1-a904-4c229d1642ee"
```

`sid_string`

Provides a way to include information about an individual statement. Some services require this value to be unique within an AWS account. Some services allow spaces in the `Sid` value, and others do not.

```
"Sid":"1"

"Sid": "ThisStatementProvidesPermissionsForConsoleAccess"
```

`resource_string`

In most cases, consists of an Amazon Resource Name (ARN).

```
"Resource":"arn:aws:iam::123456789012:user/Bob"

"Resource":"arn:aws:s3:::examplebucket/*"
```

`condition_type_string`

Identifies the type of condition being tested, such as `StringEquals`, `StringLike`, `NumericLessThan`, `DateGreaterThanEquals`, `Bool`, `BinaryEquals`, `IpAddress`, `ArnEquals`, etc. For a complete list of condition types, see Condition Types (p. 225).

```
"Condition": {
  "NumericLessThanEquals": {
    "s3:max-keys": "10"
  }
}

"Condition": {
  "Bool": {
    "aws:SecureTransport": "true"
  }
}

"Condition": {
  "StringEquals": {
      "s3:x-amz-server-side-encryption": "AES256"
   }
}
```

`condition_key_string`

Identifies the condition key whose value will be tested to determine whether the condition is met. AWS defines a set of condition keys that are available in all AWS services, including `aws:principaltype`, `aws:SecureTransport`, and `aws:userid`.

For a list of AWS condition keys, see Available Keys for Conditions (p. 222). For condition keys that are specific to a service, see the documentation for that service, such as Specifying Conditions in a Policy in the *Amazon Simple Storage Service Developer Guide* and IAM Policies for Amazon EC2 in the *Amazon EC2 User Guide for Linux Instances*.

```
"Condition":{
  "Bool": {
      "aws:SecureTransport": "true"
   }
}

"Condition": {
  "StringNotEquals": {
      "s3:x-amz-server-side-encryption": "AES256"
   }
}

"Condition": {
  "StringEquals": {
    "ec2:ResourceTag/purpose": "test"
  }
}
```

Common Errors in IAM Policies

This section describes common policy errors found in IAM policies and how to correct them.

IAM policies use a syntax that begins with the rules of JavaScript Object Notation (JSON). JSON describes an 'object' and the name and value pairs that make up an object. The IAM policy grammar builds on that by defining what names and values have meaning for, and are understood by, the AWS services that use policies to grant permissions.

Common policy errors

- More than one policy object (p. 253)
- More than one Statement element (p. 254)
- More than one Effect, Action, or Resource element in a Statement element (p. 255)
- Missing Version element (p. 257)

More than one policy object

An IAM policy must consist of one and only one JSON object. You denote an object by placing { } braces around it. Although you can nest other objects within a JSON object by embedding additional { } braces within the outer pair, a policy can contain only one outermost pair of { } braces. The following example is incorrect because it contains two objects at the top level (called out in *`red`*:

```
{
  "Version": "2012-10-17",
  "Statement":
  {
     "Effect":"Allow",
     "Action":"ec2:Describe*",
     "Resource":"arn:aws:ec2:us-east-1:ACCOUNT-ID-WITHOUT-HYPHENS:instance/*"
  }
}
{
  "Statement": {
     "Effect": "Allow",
     "Action": "s3:*",
     "Resource": "arn:aws:s3:::my-bucket/*"
  }
}
```

You can, however, meet the intention of the previous example with the use of correct policy grammar. Instead of including two complete policy objects each with its own`Statement` element, you can combine the two blocks into a single `Statement` element. The `Statement` element has an array of two objects as its value, as shown in the following example:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "ec2:Describe*",
     "Resource":" arn:aws:ec2:us-east-1:ACCOUNT-ID-WITHOUT-HYPHENS:instance/*"

    },
    {
      "Effect": "Allow",
      "Action": "s3:*",
      "Resource": "arn:aws:s3:::my-bucket/*"
    }
  ]
}
```

More than one Statement element

This error might at first appear to be a variation on the previous section. However, syntactically it is a different type of error. In the following example there is only one policy object as denoted by a single pair of { } braces at the top level. However, that object contains two `Statement` elements within it.

An IAM policy must contain only one `Statement` element, consisting of the name (`Statement`) appearing to the left of a colon, followed by its value on the right. The value of a `Statement` element must be an object, denoted by { } braces, containing one `Effect` element, one `Action` element, and one `Resource` element. The following example is incorrect because it contains two `Statement` elements in the policy object.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "ec2:Describe*",
    "Resource": "*"
  },
  "Statement": {
    "Effect": "Allow",
    "Action": "s3:*",
    "Resource": "arn:aws:s3:::my-bucket/*"
  }
}
```

Because a value object can be an array of multiple value objects, you can solve this problem by combining the two `Statement` elements into one element with an object array, as shown in the following example:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "ec2:Describe*",
      "Resource":"*"
    },
    {
      "Effect": "Allow",
      "Action": "s3:*",
      "Resource": "arn:aws:s3:::my-bucket/*"
    }
 ]
}
```

The value of the `Statement` element is an object array. The array in this example consists of two objects, each of which is by itself is a correct value for a `Statement` element. Each object in the array is separated by commas.

More than one Effect, Action, or Resource element in a Statement element

On the value side of the `Statement` name/value pair, the object must consist of only one `Effect` element, one `Action` element, and one `Resource` element. The following policy is incorrect because it has two `Effect` elements in the value object of the `Statement`:

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Deny",
    "Effect": "Allow",
    "Action": "ec2:* ",
    "Resource": "*"
  }
}
```

Note

Although the policy engine has been updated to block such errors in new or edited policies, the policy engine continues to permit policies that were saved before the engine was updated. The behavior of existing policies with the error is as follows:

- Multiple `Effect` elements: only the last `Effect` element is observed. The others are ignored.

- Multiple `Action` elements: all `Action` elements are combined internally and treated as if they were a single list.
- Multiple `Resource` elements: all `Resource` elements are combined internally and treated as if they were a single list.

Beginning in March 2015, the policy engine no longer allows you to save any policy with these errors. You must correct the policy before you can save it. The Policy Validator (p. 201) tool can help you to find all older policies with errors and can recommend corrections for them.

In each case, the solution is to remove the incorrect extra element. For `Effect` elements, this is straightforward: if you want the previous example to *deny* permissions to Amazon EC2 instances, then you must remove the line `"Effect": "Allow",` from the policy, as follows:

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Deny",
    "Action": "ec2:* ",
    "Resource": "*"
  }
}
```

However, if the duplicate element is `Action` or `Resource`, then the resolution can be more complicated. You might have multiple actions that you want to allow (or deny) permission to, or you might want to control access to multiple resources. For example, the following example is incorrect because it has multiple `Resource` elements:

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "s3:*",
    "Resource": "arn:aws:s3:::my-bucket",
    "Resource": "arn:aws:s3:::my-bucket/*"
  }
}
```

Each of the required elements in a `Statement` element's value object can be present only once. The solution is to place each value in an array. The following example illustrates this by making the two separate resource elements into one `Resource` element with an array as the value object:

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "s3:*",
    "Resource": [
      "arn:aws:s3:::my-bucket",
      "arn:aws:s3:::my-bucket/*"
    ]
  }
}
```

Missing Version element

As AWS features evolve, new capabilities are added to IAM policies to support those features. Sometimes, an update to the policy syntax includes a new version number. If you use newer features of the policy grammar in your policy, then you must tell the policy parsing engine which version you are using. The default policy version is "2008-10-17". If you want to use any policy feature that was introduced later, then you must specify the version number that supports the feature you want. We recommend that you *always* include the latest policy syntax version number, which is `"Version": "2012-10-17"`. For example, the following policy is incorrect because it uses a policy variable `${...}` in the ARN for a resource without specifying a policy syntax version that supports policy variables:

```
{
  "Statement":
  {
    "Action": "iam:*AccessKey*",
    "Effect": "Allow",
    "Resource": "arn:aws:iam::accountid:user/${aws:username}"
  }
}
```

Adding a `Version` element at the top of the policy with the value `2012-10-17`, the first version to support policy variables, solves this problem:

```
{
  "Version": "2012-10-17",
  "Statement":
  {
    "Action": "iam:*AccessKey*",
    "Effect": "Allow",
    "Resource": "arn:aws:iam::accountid:user/${aws:username}"
  }
}
```

Resources for Learning About Permissions and Policies

For more information about permissions and about creating policies, see the following resources:

- Overview of IAM Policies (p. 174) – This section discusses types of permissions, how to grant them, and how to manage permissions.
- Managing Policies (p. 192) – This section discusses how to specify what actions are allowed, which resources to allow the actions on, and what the effect will be when the user requests access to the resources.
- The following entries in the AWS Security Blog cover common ways to write policies for access to Amazon S3 buckets and objects.
  - Writing IAM Policies: How to grant access to an Amazon S3 bucket
  - Writing IAM policies: Grant access to user-specific folders in an Amazon S3 bucket
  - IAM Policies and Bucket Policies and ACLs! Oh My! (Controlling Access to S3 Resources)
  - A primer on RDS resource-level permissions
  - Demystifying EC2 Resource-Level Permissions

- IAM Policy Simulator – This tool lets you test the effects of IAM policies to determine whether they will allow or deny access to actions in AWS services.
- Example Policies for Administering IAM Resources (p. 97) – This section contains example policies that show how to perform tasks specific to IAM, like administer users, groups, and credentials.

Logging IAM Events with AWS CloudTrail

AWS Identity and Access Management (IAM) is integrated with AWS CloudTrail, a service that logs AWS events made by or on behalf of your AWS account. CloudTrail logs authenticated AWS API calls and also AWS sign-in events, and collects this event information in files that are delivered to Amazon S3 buckets. Using information collected by CloudTrail, you can determine what requests were successfully made to AWS services, who made the request, when it was made, and so on.

To learn more about CloudTrail, including how to configure and enable it, see the *AWS CloudTrail User Guide*.

Topics

Types of IAM Information Logged in CloudTrail

IAM information is available to CloudTrail in these ways:

- **API requests to IAM and AWS Security Token Service (AWS STS)** – All authenticated API requests to IAM and AWS STS APIs are logged by CloudTrail.

  For example, calls to the IAM `CreateUser`, `DeleteRole`, `ListGroups`, and other API operations are all logged by CloudTrail.

  Details about this type of log entry are discussed later in this topic.

  > **Important**
  > If you activate AWS STS endpoints in regions other than the default global endpoint, then you must also turn on CloudTrail logging in those regions to record any AWS STS API calls made in those regions. For more information, see Turning On CloudTrail in Additional Regions in the AWS CloudTrail User Guide.

- **API requests to other AWS services** – Authenticated requests to other AWS service APIs are logged by CloudTrail, and these log entries contain information about who generated the request.

For example, if a request is made to list Amazon EC2 instances or to create a new DynamoDB table, the user identity of the person or service that made the request is contained in the log entry for that request. The user identity information helps you determine whether the request was made with AWS account (root) or IAM user credentials, with temporary security credentials for a role or federated user, or by another AWS service.

For more details about the user identity information in CloudTrail log entries, see userIdentity Element in the *AWS CloudTrail User Guide*.

- **AWS sign-in events** – Sign-in events to the AWS Management Console, the AWS Discussion Forums, and the AWS Marketplace are logged by CloudTrail.

  For example, IAM and federated user sign-in events—successful sign-ins and failed sign-in attempts—are logged by CloudTrail. Additionally, successful sign-in events by the AWS account (root) are logged by CloudTrail.

  Note
  Unsuccessful sign-in events by the AWS account (root) are not logged by CloudTrail.

  Details about log entries for AWS sign-in events are discussed later in this topic.

Examples of Logged Events in CloudTrail Files

CloudTrail log files contain events that are formatted using JSON. An event represents a single API request or sign-in event and includes information about the requested action, any parameters, and the date and time of the action.

IAM API Event in CloudTrail Log File

The following example shows a CloudTrail log entry for a request made for the IAM `GetUserPolicy` action.

```
{
  "eventVersion": "1.01",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE",
    "arn": "arn:aws:iam::444455556666:user/Alice",
    "accountId": "444455556666",
    "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
    "userName": "Alice",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2014-07-15T21:39:40Z"
      }
    },
    "invokedBy": "signin.amazonaws.com"
  },
  "eventTime": "2014-07-15T21:40:14Z",
  "eventSource": "iam.amazonaws.com",
  "eventName": "GetUserPolicy",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "signin.amazonaws.com",
  "userAgent": "signin.amazonaws.com",
  "requestParameters": {
    "userName": "Bob",
    "policyName": "ReadOnlyAccess-Bob-201407151307"
  },
  "responseElements": null,
  "requestID": "9b4bb6fe-0c68-11e4-a24e-d5e160cfd347",
  "eventID": "cf6228c3-127e-4632-980d-505a4d39c01e"
}
```

From this event information, you can determine that the request was made to get a user policy named `ReadOnlyAccess-Bob-201407151307` for user `Bob`, as specified in the `requestParameters` element. You can also see that the request was made by an IAM user named `Alice` on July 15, 2014 at 9:40 PM (UTC). In this case, the request originated in the AWS Management Console, as you can tell from the `userAgent` element.

AWS STS API Event in CloudTrail Log File

The following example shows a CloudTrail log entry for a request made for the AWS STS `AssumeRole` action.

```
{
  "eventVersion": "1.01",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE",
    "arn": "arn:aws:iam::777788889999:user/Bob",
    "accountId": "777788889999",
    "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
    "userName": "Bob"
  },
  "eventTime": "2014-07-18T15:07:39Z",
  "eventSource": "sts.amazonaws.com",
  "eventName": "AssumeRole",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.0.2.101",
  "userAgent": "aws-cli/1.3.23 Python/2.7.6 Linux/2.6.18-164.el5",
  "requestParameters": {
    "roleArn": "arn:aws:iam::777788889999:role/EC2-dev",
    "roleSessionName": "Bob-EC2-dev"
  },
  "responseElements": {
    "credentials": {
      "sessionToken": "encoded session token blob",
      "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
      "expiration": "Jul 18, 2014 4:07:39 PM"
      },
    "assumedRoleUser": {
      "assumedRoleId": "AIDACKCEVSQ6C2EXAMPLE:Bob-EC2-dev",
      "arn": "arn:aws:sts::777788889999:assumed-role/EC2-dev/Bob-EC2-dev"
    }
  },
  "requestID": "4268a7ca-0e8d-11e4-96e4-e55c0ccbab4c",
  "eventID": "dfea2f96-ac7f-466c-a608-4ac8d286ba0b"
}
```

Sign-in Failure Event in CloudTrail Log File

The following example shows a CloudTrail log entry for a failed sign-in event.

```
{
  "eventVersion": "1.01",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE",
    "arn":"arn:aws:iam::111122223333:user/Alice",
    "accountId": "111122223333",
    "userName": "Alice"
  },
  "eventTime": "2014-07-08T17:35:27Z",
  "eventSource": "signin.amazonaws.com",
  "eventName": "ConsoleLogin",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.0.2.100",
  "userAgent": "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:24.0) Gecko/20100101
Firefox/24.0",
  "errorMessage": "Failed authentication",
  "requestParameters": null,
  "responseElements": {
    "ConsoleLogin": "Failure"
  },
  "additionalEventData": {
    "MobileVersion": "No",
    "LoginTo": "https://console.aws.amazon.com/sns",
    "MFAUsed": "No"
  },
  "eventID": "11ea990b-4678-4bcd-8fbe-62509088b7cf"
}
```

From this information, you can determine that the sign-in attempt was made by an IAM user named Alice, as shown in the `userIdentity` element. You can also see that the sign-in attempt failed, as shown in the `responseElements` element. You can see that Alice tried to sign in to the Amazon SNS console at 5:35 PM (UTC) on July 8th, 2014.

Sign-in Success Event in CloudTrail Log File

The following example shows a CloudTrail log entry for a successful sign-in event.

```
{
  "eventVersion": "1.01",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE",
    "arn":"arn:aws:iam::111122223333:user/Bob",
    "accountId": "111122223333",
    "userName": "Bob"
  },
  "eventTime": "2014-07-16T15:49:27Z",
  "eventSource": "signin.amazonaws.com",
  "eventName": "ConsoleLogin",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.0.2.110",
  "userAgent": "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:24.0) Gecko/20100101 
Firefox/24.0",
  "requestParameters": null,
  "responseElements": {
    "ConsoleLogin": "Success"
  },
  "additionalEventData": {
    "MobileVersion": "No",
    "LoginTo": "https://console.aws.amazon.com/s3",
    "MFAUsed": "No"
  },
  "eventID": "3fcfb182-98f8-4744-bd45-10a395ab61cb"
}
```

For more details about the information contained in CloudTrail log files, see CloudTrail Event Reference in the *AWS CloudTrail User Guide*.

Preventing Duplicate Log Entries in CloudTrail

CloudTrail creates trails in each region separately. These trails include information for events that occur in those regions, plus global (that is, non-region-specific) events such as IAM API calls, non-region specific AWS STS calls (calls to `sts.amazonaws.com`), and AWS sign-in events. For example, by default, if you have two trails, each in a different region, and you then create a new IAM user, the `CreateUser` event is added to the log files in both regions, creating a duplicate log entry.

Note
By default, AWS Security Token Service (STS) is a global service with a single endpoint at https://sts.amazonaws.com. Calls to this endpoint are logged as calls to a global service. However, because this endpoint is physically located in the US East (N. Virginia) region, your logs list "us-east-1" as the event region. CloudTrail does not write these logs to the US East (N. Virginia) region unless you choose to include global service logs in that region. CloudTrail writes calls to all regional endpoints to their respective regions. For example, calls to `sts.us-east-1.amazonaws.com` are published to the US East (N. Virginia) region, calls to `sts.eu-central-1.amazonaws.com` are published to the EU (Frankfurt) region, etc.
For more information about multiple regions and AWS STS see Activating AWS STS in a New Region in Using Temporary Security Credentials.

The following table lists the regions and how CloudTrail logs AWS STS requests in each region. The "Location" column indicates which logs CloudTrail writes to. "Global" means the event is logged in any region for which you choose to include global service logs in that region. "Region" means that the event is logged only in the region where the endpoint is located. The last column indicates how the request's region is identified in the log entry.

Endpoint	Location of CloudTrail logs	Region identity in log
sts.amazonaws.com	global	us-east-1
sts.us-east-1.amazonaws.com	region	us-east-1
sts.us-west-2.amazonaws.com	region	us-west-2
sts.us-west-1.amazonaws.com	region	us-west-1
sts.eu-west-1.amazonaws.com	region	eu-west-1

Endpoint	Location of CloudTrail logs	Region identity in log
sts.eu-central-1.amazonaws.com	region	eu-central-1
sts.ap-southeast-1.amazonaws.com	region	ap-southeast-1
sts.ap-southeast-2.amazonaws.com	region	ap-southeast-2
sts.ap-northeast-1.amazonaws.com	region	ap-northeast-1
sts.sa-east-1.amazonaws.com	region	sa-east-1

When you configure CloudTrail to aggregate trail information from multiple regions in your account into a single Amazon S3 bucket, IAM events are duplicated in the logs—the trail for each region writes the same IAM event to the aggregated log. To prevent this duplication, you can include global events selectively. A typical approach is to enable global events in one trail and to disable global events in all other trails that write to the same Amazon S3 bucket. That way only one set of global events is written.

For more information, see Aggregating Logs in the *AWS CloudTrail User Guide*.

AWS Services That Support IAM

This section links to topics that describe how AWS Identity and Access Management integrates with other services from AWS, and how to write policies to control access to a particular service and its resources.

Note
Some services support resource-based permissions (p. 172), which let you attach policies to the service's resources in addition to IAM users, groups, or roles. Resource-based permissions are supported by Amazon S3, Amazon SNS, Amazon SQS, and AWS Key Management Service. For information about resource-based policies in these services, see the links in the following table for those services.

In the following table, the columns have the following meanings:

- **Supports action-level permissions**. The service supports IAM policies in which you can specify individual actions in a policy's `Action` element (p. 216). If the service does not support action-level permissions, policies for the service use `*` in the `Action` element.
- **Supports resource-level permissions**. The service supports IAM policies in which you can specify individual resources (using ARNs) in the policy's `Resource` element (p. 217). If the service does not support resource-level permissions, policies for the service use `*` in the `Resource` element.

  Note
  Some services support resource-level permissions only for some actions. See the notes that follow the table for more information.
- **Supports tag-based permissions**. The service supports IAM policies that let you create resource-level permissions by testing tags attached to a resource in a `Condition` element (p. 219).
- **Supports temporary security credentials**. The service lets users make requests using temporary security credentials that are obtained by calling AWS STS APIs like AssumeRole or GetFederationToken. For more information, see the *Using Temporary Security Credentials* guide.
- **More information**. Links to more information in the documentation of the service.

AWS Service Category: Compute	Supports action-level permissions	Supports resource-level permissions	Supports tag-based permissions	Supports temporary security credentials	More information

Amazon Elastic Compute Cloud (Amazon EC2)	Yes	Yes; see Notes (p. 270)	Yes; see Notes (p. 270)	Yes	Controlling Access to Amazon EC2 Resources
Amazon EC2 Container Service (Amazon ECS)	Yes	No	No	Yes	Amazon ECS IAM Policies
Auto Scaling	Yes	No	No	Yes	Auto Scaling and AWS Identity and Access Management
Elastic Load Balancing	Yes	Yes	No	Yes	Control User Access to Your AWS Account
AWS Lambda	Yes	Yes	No	Yes	
AWS Service Category: Storage and Content Delivery	**Supports action-level permissions**	**Supports resource-level permissions**	**Supports tag-based permissions**	**Supports temporary security credentials**	**More information**
Amazon Simple Storage Service (Amazon S3)	Yes	Yes	No	Yes	Using IAM Policies
AWS Storage Gateway	Yes	Yes	No	Yes	Access Control Using AWS Identity and Access Management (IAM)
Amazon Glacier	Yes	Yes	No	Yes	Access Control Using AWS Identity and Access Management (IAM)
Amazon CloudFront	Yes; see Notes (p. 270)	No	No	Yes	Using IAM to Control Access to CloudFront Resources
Amazon Elastic Block Store (Amazon EBS)	Yes	Yes; see Notes (p. 270)	Yes; see Notes (p. 270)	Yes	Controlling Access to Amazon EC2 Resources
AWS Import/Export	Yes	No	No	Yes	Using IAM with AWS Import/Export
AWS Service Category: Database	**Supports action-level permissions**	**Supports resource-level permissions**	**Supports tag-based permissions**	**Supports temporary security credentials**	**More information**

Amazon Relational Database Service (Amazon RDS)	Yes	Yes	Yes	Yes	Controlling Access to Amazon RDS Resources
Amazon DynamoDB	Yes	Yes	No	Yes	Controlling Access to Amazon DynamoDB Resources
Amazon ElastiCache	Yes	No	No	Yes	Controlling User Access to Your AWS Account
Amazon Redshift	Yes	Yes	No	Yes	Controlling Access to Amazon Redshift Resources
Amazon SimpleDB	Yes	Yes	No	Yes	Managing Users of Amazon SimpleDB
AWS Service Category: Networking	**Supports action-level permissions**	**Supports resource-level permissions**	**Supports tag-based permissions**	**Supports temporary security credentials**	**More information**
Amazon Virtual Private Cloud (Amazon VPC)	Yes	Yes	Yes	Yes	Controlling VPC Management
AWS Direct Connect	Yes	No	No	Yes	Using AWS Identity and Access Management with AWS Direct Connect
Amazon Route 53	Yes	Yes	No	Yes	Using IAM to Control Access to Route 53 Resources
AWS Service Category: Administration and Security	**Supports action-level permissions**	**Supports resource-level permissions**	**Supports tag-based permissions**	**Supports temporary security credentials**	**More information**
AWS Directory Service	Yes	No	No	Yes	Controlling Access to AWS Directory Service Resources
AWS Identity and Access Management (IAM)	Yes	Yes	No	Yes. See Summary of AWS STS API Functionality in *Using Temporary Security Credentials*.	Permissions for Administering IAM Users, Groups, and Credentials (p. 95)

AWS Security Token Service (AWS STS)	Yes	Not applicable	No	Yes. See Summary of AWS STS API Functionality in *Using Temporary Security Credentials*.	Controlling Permissions for Temporary Security Credentials
AWS Trusted Advisor	Yes; see Notes (p. 270)	Yes	No	Yes; see Notes (p. 270)	Controlling Access to the Trusted Advisor Console
AWS CloudTrail	Yes	No	No	Yes	Controlling User Access to AWS CloudTrail Actions
AWS Config	Yes	No	No	Yes	Recommended IAM Permissions for Using the AWS Config Console and the AWS CLI
Amazon CloudWatch	Yes	No	No	Yes	Controlling User Access to Your AWS Account
AWS Key Management Service (AWS KMS)	Yes	Yes	No	Yes	Key Policies
AWS CloudHSM	Yes	No	No	No	Controlling Access to AWS CloudHSM Resources
AWS Service Category: Deployment and Management	**Supports action-level permissions**	**Supports resource-level permissions**	**Supports tag-based permissions**	**Supports temporary security credentials**	**More information**
AWS Elastic Beanstalk	Yes	Yes	No	Yes	Using AWS Elastic Beanstalk with AWS Identity and Access Management (IAM)
AWS OpsWorks	Yes	Yes	No	Yes	Granting Users Permissions to Work with AWS OpsWorks
AWS CloudFormation	Yes	Yes	No	Yes	Controlling User Access with AWS Identity and Access Management
AWS CodeDeploy	Yes	Yes	No	Yes	AWS CodeDeploy User Access Permissions Reference

AWS Service Category: Analytics	Supports action-level permissions	Supports resource-level permissions	Supports tag-based permissions	Supports temporary security credentials	More information
Amazon Elastic MapReduce (Amazon EMR)	Yes	No	No	Yes	Configure IAM User Permissions
Amazon Kinesis	Yes	Yes	No	Yes	Controlling Access to Amazon Kinesis Resources with IAM
AWS Data Pipeline	Yes	No	No	Yes; see Notes (p. 270)	IAM Roles
AWS Service Category: Application Services	**Supports action-level permissions**	**Supports resource-level permissions**	**Supports tag-based permissions**	**Supports temporary security credentials**	**More information**
Amazon Simple Queue Service (Amazon SQS)	Yes	Yes	No	Yes	Controlling User Access to Your AWS Account
Amazon Simple Workflow Service (Amazon SWF)	Yes	Yes	Yes	Yes	Using IAM to Manage Access to Amazon SWF Resources
Amazon AppStream	Yes	No	No	Yes	Security Considerations for Amazon AppStream
Amazon Elastic Transcoder	Yes	Yes	No	Yes	Security Considerations for Elastic Transcoder
Amazon Simple Email Service (Amazon SES)	Yes	No	No	Yes	Controlling User Access to Amazon SES
Amazon CloudSearch	Yes	Yes	No	Yes	Configuring Access for an Amazon CloudSearch Domain
AWS Service Category: Mobile Services	**Supports action-level permissions**	**Supports resource-level permissions**	**Supports tag-based permissions**	**Supports temporary security credentials**	**More information**
Amazon Cognito	Yes	No	No	Yes	

Amazon Mobile Analytics	Yes	No	No	Yes	
Amazon Simple Notification Service (Amazon SNS)	Yes	Yes	No	Yes	Controlling User Access to Your AWS Account
AWS Service Category: Enterprise Applications	**Supports action-level permissions**	**Supports resource-level permissions**	**Supports tag-based permissions**	**Supports temporary security credentials**	**More information**
Amazon WorkDocs	Yes	No	No	Yes	IAM Policies for Amazon WorkDocs
Amazon WorkSpaces	Yes	No	No	Yes	Controlling Access to Amazon WorkSpaces Resources
AWS Service Category: Additional Resources	**Supports action-level permissions**	**Supports resource-level permissions**	**Supports tag-based permissions**	**Supports temporary security credentials**	**More information**
AWS Billing and Cost Management	Yes	No	No	Yes	Controlling User Access to Your AWS Billing and Cost Management Information
AWS Marketplace	Yes	Yes	No	Yes	Controlling Access to AWS Marketplace Subscriptions
AWS Support	No	No	No	Yes	Accessing AWS Support

Notes

- **Amazon EC2**: Amazon EC2 supports resource-level permissions and tags only for some APIs. For more information, see Supported Resources and Conditions for Amazon EC2 API Actions in the *Amazon EC2 User Guide for Linux Instances*.
- **Amazon CloudFront**: Amazon CloudFront does not support action-level permissions for creating CloudFront key pairs. You must use an AWS root account to create a CloudFront key pair. For more information, see Creating CloudFront Key Pairs for Your Trusted Signers in the *Amazon CloudFront Developer Guide*.
- **Amazon EBS**: Amazon EBS supports resource-level permissions and tags only for some APIs. For more information, see Supported Resources and Conditions for Amazon EC2 API Actions in the *Amazon EC2 User Guide for Linux Instances*.
- **Trusted Advisor**: API access to Trusted Advisor is through the AWS Support API and is controlled by AWS Support IAM policies.

- **AWS Data Pipeline**: AWS Data Pipeline does not support temporary security credentials for the `CreatePipeline` API. All other AWS Data Pipeline API actions support temporary security credentials.

Troubleshooting IAM

The following section discusses common issues that you might encounter when you work with IAM.

Topics

General: Access is denied when I make a request to an AWS service.

Verify that you have permission to call the action and resource that you have requested. If any conditions are set, you must also meet those conditions when you send the request. For information about viewing or modifying policies for an IAM user, group, or role, see Managing Policies (p. 192).

If you're trying to access a service that has resource-based (or access control) policies, such as Amazon S3, Amazon SNS, or Amazon SQS, verify that the resource policy specifies you as a principal and grants you access. For more information about resource-based policies, see the documentation for that service.

If you are signing requests manually (without using the AWS SDKs), verify that you have correctly signed the request.

General: Access is denied when I make a request with temporary security credentials.

- Verify that the service accepts temporary security credentials, see Using Temporary Security Credentials to Access AWS.
- Verify that your requests are being signed correctly and that the request is well-formed. For details, see your toolkit documentation or Using Temporary Security Credentials to Authenticate an AWS Request.
- Verify that your temporary security credentials haven't expired. For more information, see Using Temporary Security Credentials.
- Verify that the IAM user or role has the correct permissions. Permissions for temporary security credentials are derived from an IAM user or role, so the permissions are limited to those granted to the IAM user or role. For more information about how permission for temporary security credentials are determined, see Controlling Permissions for Temporary Security Credentials in *Using Temporary Security Credentials*.
- If you are accessing a resource that has a resource-based policy by using a role, verify that the policy grants permissions to the role. For example, the following policy allows `MyRole` from account `111122223333` to access `MyBucket`.

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Sid": "S3BucketPolicy",
    "Effect": "Allow",
    "Principal": {"AWS": ["arn:aws:iam::111122223333:role/MyRole"]},
    "Action": ["s3:PutObject"],
    "Resource": ["arn:aws:s3:::MyBucket/*"]
  }]
}
```

General: Policy variables aren't working.

- Verify that all policies that include variables include the following version number in the policy:

```
"Version": "2012-10-17"
```

  Without the correct version number, the variables are not replaced during evaluation. Instead, the variables are evaluated literally. Any policies that don't include variables will still work if you include the latest version number.
- Verify that your policy variables are in the right case. For details, see IAM Policy Variables Overview (p. 232).

General: I cannot assume a role.

- Verify that your IAM policy grants you privilege to call `sts:AssumeRole` for the role that you want to assume. The `action` element of your IAM policy must allow you to call the assume role action, and the `resource` element of your IAM policy must allow you to call the role that you want to assume. For example, the `resource` element can specify the role's Amazon Resource Name (ARN) or a wildcard (*).

- Verify that you meet all the conditions that are specified in the role. Conditions are defined as part of the role's trust relationship. A condition can specify an expiration date, an external ID, or that request come from specific IP addresses.
- Verify that the AWS account that you are calling `AssumeRole` from is a trusted entity for the role that you are assuming. Trusted entities are defined in a role's trust policy.

Amazon EC2: When attempting to launch an instance, I don't see the role I expected to see in the Amazon EC2 console IAM role list.

Check the following:

- If you are signed in as an IAM user, verify that you have permission to call `ListInstanceProfiles`. For information about the permissions necessary to work with roles, see "Permissions Required for Using Roles with Amazon EC2" in Using IAM Roles to Delegate Permissions to Applications that Run on Amazon EC2 (p. 140). For information about adding permissions to a user, see Managing Policies (p. 192).

  If you cannot modify your own permissions, you must contact an administrator who can work with IAM in order to update your permissions.
- If you created a role by using the IAM CLI or API, verify that you created an instance profile and added the role to that instance profile. Also, if you name your role and instance profile differently, you won't see the correct role name in the list of IAM roles in the Amazon EC2 console. The **IAM Role** list in the Amazon EC2 console lists the names of instance profiles, not the names of roles. You will have to select the name of the instance profile that contains the role you want. For details about instance profiles, see Instance Profiles (p. 144).

  Note
  If you use the IAM console to create roles, you don't need to work with instance profiles. For each role that you create in the IAM console, an instance profile is created with the same name as the role, and the role is automatically added to that instance profile.

Amazon EC2: The credentials on my instance are for the wrong role.

If the role in the instance profile was replaced recently, your application will need to wait for the next automatically scheduled credential rotation before credentials for your role become available.

Amazon EC2: When I attempt to call the `AddRoleToInstanceProfile`, I get an `AccessDenied` error.

If you are making requests as an IAM user, verify that you have the following permissions:

- `iam:AddRoleToInstanceProfile` with the resource matching the instance profile ARN (for example, `arn:aws:iam::999999999999:instance-profile/ExampleInstanceProfile`).

For more information about the permissions necessary to work with roles, see "How Do I Get Started?" in Using IAM Roles to Delegate Permissions to Applications that Run on Amazon EC2 (p. 140). For information about adding permissions to a user, see Managing Policies (p. 192).

Amazon EC2: When I attempt to launch an instance with a role, I get an `AccessDenied` error.

Check the following:

- Launch an instance without an instance profile. This will help ensure that the problem is limited to IAM roles for Amazon EC2 instances.
- If you are making requests as an IAM user, verify that you have the following permissions:
  - `ec2:RunInstances` with a wildcard resource ("*")
  - `iam:PassRole` with the resource matching the role ARN (for example, `arn:aws:iam::999999999999:role/ExampleRoleName`)
- Call the IAM `GetInstanceProfile` action to ensure that you are using a valid instance profile name or a valid instance profile ARN. For more information, see Using IAM roles with Amazon EC2 instances.
- Call the IAM `GetInstanceProfile` action to ensure that the instance profile has a role. Empty instance profiles will fail with an `AccessDenied` error. For more information about creating a role, see Creating IAM Roles (p. 121).

For more information about the permissions necessary to work with roles, see "How Do I Get Started?" in Using IAM Roles to Delegate Permissions to Applications that Run on Amazon EC2 (p. 140). For information about adding permissions to a user, see Managing Policies (p. 192).

Amazon EC2: I can't access the temporary security credentials on my EC2 instance.

Check the following:

- Can you can access another part of the instance metadata service (IMDS)? If not, check that you have no firewall rules blocking access to requests to the IMDS.

```
[ec2-user@domU-12-31-39-0A-8D-DE ~]$ GET http://169.254.169.254/latest/meta-
data/hostname; echo
```

- Does the `iam` subtree of the IMDS exist? If not, verify that your instance has an IAM instance profile associated with it by calling `ec2:DescribeInstances`.

```
[ec2-user@domU-12-31-39-0A-8D-DE ~]$ GET http://169.254.169.254/latest/meta-
data/iam; echo
```

- Check the `info` document in the IAM subtree for an error. If you have an error, see Amazon EC2: What do the errors from the `info` document in the IAM subtree mean? (p. 276) for more information.

```
[ec2-user@domU-12-31-39-0A-8D-DE ~]$ GET http://169.254.169.254/latest/meta-
data/iam/info; echo
```

Amazon EC2: What do the errors from the `info` document in the IAM subtree mean?

The `iam/info` document indicates `"Code":"InstanceProfileNotFound"`.

Your IAM instance profile has been deleted and Amazon EC2 can no longer provide credentials to your instance. You will need to terminate your instances and restart with a valid instance profile.

If an instance profile with that name exists, check that the instance profile wasn't deleted and another was created with the same name.

To verify the status of the instance profile:

1. Call the IAM `GetInstanceProfile` action to get the `InstanceProfileId`.
2. Call the Amazon EC2 `DescribeInstances` action to get the `IamInstanceProfileId` for the instance.
3. Verify that the `InstanceProfileId` from the IAM action matches the `IamInstanceProfileId` from the Amazon EC2 action.

If the IDs are different, then the instance profile attached to your instances is no longer valid. You will need to terminate your instances and restart with a valid instance profile.

The `iam/info` document indicates a success but indicates `"Message":"Instance Profile does not contain a role..."`

The role has been removed from the instance profile by the IAM `RemoveRoleFromInstanceProfile` action. You can use the IAM `AddRoleToInstanceProfile` action to attach a role to the instance profile. Your application will need to wait until the next scheduled refresh to access the credentials for the role.

The `iam/security-credentials/[role-name]` document indicates `"Code":"AssumeRoleUnauthorizedAccess"`.

Amazon EC2 does not have permission to assume the role. Permission to assume the role is controlled by the trust policy attached to the role, like the example that follows. Use the IAM `UpdateAssumeRolePolicy` API to update the assume role policy.

```
{"Version": "2012-10-17","Statement": [{"Effect": "Allow","Principal": {"Ser
vice": ["ec2.amazonaws.com"]},"Action": ["sts:AssumeRole"]}]}
```

Your application will need to wait until the next automatically scheduled refresh to access the credentials for the role.

Amazon S3: How do I grant anonymous access to an Amazon S3 bucket?

You use an Amazon S3 bucket policy that specifies a wildcard (*) in the `principal` element, which means anyone can access the bucket. With anonymous access, anyone (including users without an AWS account) will be able to access the bucket. For a sample policy, see Example Cases for Amazon S3 Bucket Policies in the *Amazon Simple Storage Service Developer Guide*.

Amazon S3: I'm signed in as a root user, why can't I access an Amazon S3 bucket under my account?

In some cases, you might have an IAM user with full access to IAM and Amazon S3. If the IAM user assigns a bucket policy to an Amazon S3 bucket and doesn't specify the root user as a principal, the root user is denied access to that bucket. However, as the root user, you can still access the bucket by modifying the bucket policy to allow root user access.

Making Query Requests

Topics

This section contains general information about using the Query API for AWS Identity and Access Management (IAM) and AWS Security Token Service (AWS STS). For details about the API actions and errors, go to the IAM API Reference or the AWS Security Token Service API Reference.

Note
Instead of making direct calls to the IAM or AWS STS APIs, you can use one of the AWS SDKs. The AWS SDKs consist of libraries and sample code for various programming languages and platforms (Java, Ruby, .NET, iOS, Android, etc.). The SDKs provide a convenient way to create programmatic access to IAM and AWS. For example, the SDKs take care of tasks such as cryptographically signing requests (see below), managing errors, and retrying requests automatically. For information about the AWS SDKs, including how to download and install them, see the Tools for Amazon Web Services page.

The Query API for IAM and AWS STS lets you call service actions. Query API requests are HTTPS requests that must contain an *`Action`* parameter to indicate the action to be performed. IAM and AWS STS support GET and POST requests for all actions. That is, the API does not require you to use GET for some actions and POST for others. However, GET requests are subject to the limitation size of a URL; although this limit is browser dependent, a typical limit is 2048 bytes. Therefore, for Query API requests that require larger sizes, you must use a POST request.

The response is an XML document. For details about the response, see the individual action pages in the IAM API Reference or the AWS Security Token Service API Reference.

Endpoints

IAM and AWS STS each have a single global endpoint:

- (IAM) https://iam.amazonaws.com
- (AWS STS) https://sts.amazonaws.com

Note
AWS STS also supports sending requests to regional endpoints in addition to the global endpoint. Before you can use AWS STS in a region, you must first activate STS in that region for your AWS account. For more information about activating additional regions for AWS STS, see Enabling AWS STS in a New Region in the Using Temporary Security Credentials guide.

For more information about AWS endpoints and regions for all services, see Regions and Endpoints in the *AWS General Reference*.

HTTPS Required

Because the Query API returns sensitive information such as security credentials, you must use HTTPS with all API requests.

Signing IAM API Requests

Requests must be signed using an access key ID and a secret access key. We strongly recommend that you do not use your AWS root account credentials for everyday work with IAM. You can use the credentials for an IAM user or you can use AWS STS to generate temporary security credentials.

To sign your API requests, we recommend using AWS Signature Version 4. For information about using Signature Version 4, go to Signature Version 4 Signing Process in the *AWS General Reference*.

If you need to use Signature Version 2, information about using Signature Version 2 is available in the AWS General Reference.

For more information, see the following:

- AWS Security Credentials. Provides general information about the types of credentials used for accessing AWS.
- IAM Best Practices (p. 26). Presents a list of suggestions for using IAM service to help secure your AWS resources.
- Using Temporary Security Credentials. Describes how to create and use temporary security credentials.

AWS Glossary

Numbers and Symbols

100-continue	A method that enables a client to see if a server can accept a request before actually sending it. For large PUT requests, this method can save both time and bandwidth charges.

A

access control list	A document that defines who can access a particular bucket or object. Each bucket and object in Amazon S3 has an ACL. The document defines what each type of user can do, such as write and read permissions.
access identifiers	See credentials.
access key ID	A unique identifier that's associated with a secret access key (p. 303); the access key ID and secret access key are used together to sign programmatic AWS requests cryptographically.
access key rotation	A method to increase security by changing the AWS access key ID. This method enables you to retire an old key at your discretion.
access policy language	A language for writing documents (that is, *policies*) that specify who can access a particular AWS resource and under what conditions.
account	The AWS account associated with a particular AWS login ID and password. IAM: The AWS account that centrally controls all the resources created under its umbrella and pays for all AWS activity for those resources. The AWS account has permission to do anything and everything with all the AWS account resources. This is in contrast to the user (p. 308).
account activity	A web page showing your month-to-date AWS usage and costs. The account activity page is located at http://www.amazonaws.cn/account-activity/.

action	An API function. Also called *operation* or *call*. The activity the principal (p. 299) has permission to perform. The action is B in the statement "A has permission to do B to C where D applies." For example, Jane sends a request to Amazon SQS with Action=ReceiveMessage. Amazon CloudWatch: The response initiated by the change in an alarm's state: for example, from OK to ALARM. The state change may be triggered by a metric reaching the alarm threshold, or by a SetAlarmState request. Each alarm can have one or more actions assigned to each state. Actions are performed once each time the alarm changes to a state that has an action assigned, such as an Amazon Simple Notification Service notification, an Auto Scaling policy execution or an Amazon EC2 instance stop/terminate action.
active trusted signers	A list showing each of the trusted signers you've specified and the IDs of the corresponding active key pairs that CloudFront is aware of. To be able to create working signed URLs, a trusted signer must appear in this list with at least one key pair ID.
administrative suspension	Auto Scaling might suspend processes for Auto Scaling group (p. 283) that repeatedly fail to launch instances. Auto Scaling groups that most commonly experience administrative suspension have zero running instances, have been trying to launch instances for more than 24 hours, and have not succeeded in that time.
alarm	An item that watches a single metric over a specified time period, and triggers an Amazon SNS topic (p. 307) or an Auto Scaling policy (p. 299) if the value of the metric crosses a threshold value over a predetermined number of time periods.
allow	One of two possible outcomes (the other is deny (p. 288)) when an IAM access policy (p. 299) is evaluated. When a user makes a request to AWS, AWS evaluates the request based on all permissions that apply to the user and then returns either allow or deny.
Amazon CloudFront	An AWS content delivery service that helps you improve the performance, reliability, and availability of your websites and applications. See Also http://www.amazonaws.cn/cloudfront.
Amazon CloudSearch	A fully-managed service in the AWS cloud that makes it easy to set up, manage, and scale a search solution for your website or application.
Amazon CloudWatch	A web service that enables you to monitor and manage various metrics, and configure alarm actions based on data from those metrics. See Also http://www.amazonaws.cn/cloudwatch.
Amazon DevPay	An easy-to-use online billing and account management service that makes it easy for you to sell an Amazon EC2 AMI or an application built on Amazon S3. See Also http://www.amazonaws.cn/devpay.
Amazon Elastic Block Store	A service that provides block level storage volumes for use with EC2 instances. See Also http://www.amazonaws.cn/ebs.
Amazon EBS-backed AMI	Instances launched from this type of AMI use an Amazon EBS volume as their root device. Compare this with instances launched from instance store-backed AMIs, which use the instance store as the root device.
Amazon Elastic Compute Cloud	A web service that enables you to launch and manage Linux/UNIX and Windows server instances in Amazon's data centers. See Also http://www.amazonaws.cn/ec2.

Amazon EC2 VM Import Connector	See http://www.amazonaws.cn/ec2/vm-import.
Amazon Elastic MapReduce	A web service that makes it easy to process large amounts of data efficiently. Amazon EMR uses Hadoop processing combined with several AWS products to do such tasks as web indexing, data mining, log file analysis, machine learning, scientific simulation, and data warehousing. See Also http://www.amazonaws.cn/elasticmapreduce.
Amazon Machine Image	An encrypted machine image stored in Amazon Elastic Block Store (p. 281) or Amazon Simple Storage Service. AMIs are like a template of a computer's root drive. They contain the operating system and can also include software and layers of your application, such as database servers, middleware, web servers, and so on.
Amazon Relational Database Service	A web service that makes it easier to set up, operate, and scale a relational database in the cloud. It provides cost-efficient, resizable capacity for an industry-standard relational database and manages common database administration tasks. See Also http://www.amazonaws.cn/rds.
Amazon Resource Name	A standardized way to refer to an AWS resource. For example: arn:aws:iam::123456789012:user/division_abc/subdivision_xyz/Bob.
Amazon Route 53	A web service you can use to create a new DNS service or to migrate your existing DNS service to the cloud. See Also http://www.amazonaws.cn/route53.
Amazon S3	See Amazon Simple Storage Service.
Amazon S3-Backed AMI	See instance store-backed AMI.
Amazon Simple Email Service	An easy-to-use, cost-effective email solution for applications. See Also http://www.amazonaws.cn/ses.
Amazon Simple Notification Service	A web service that enables applications, end-users, and devices to instantly send and receive notifications from the cloud. See Also http://www.amazonaws.cn/sns.
Amazon Simple Queue Service	Reliable and scalable hosted queues for storing messages as they travel between computers. See Also http://www.amazonaws.cn/sqs.
Amazon Simple Storage Service	Storage for the internet. You can use it to store and retrieve any amount of data at any time, from anywhere on the web. See Also http://www.amazonaws.cn/s3.
Amazon SimpleDB	A highly-available, scalable, and flexible non-relational data store that enables you to store and query data items using web service requests. See Also http://www.amazonaws.cn/simpledb.
Amazon Virtual Private Cloud	A web service that enables you to create a virtual network for your AWS resources. See Also http://www.amazonaws.cn/vpc.
Amazon Web Services	An infrastructure web services platform in the cloud for companies of all sizes. See Also http://www.amazonaws.cn.
AMI	See Amazon Machine Image.

application	A logical collection of AWS Elastic Beanstalk components, including environments, versions, and environment configurations. An application is conceptually similar to a folder.
analysis scheme	Language-specific Amazon CloudSearch text analysis options that are applied to a text field to control stemming and configure stopwords and synonyms.
Application Billing	The location where your customers manage the Amazon DevPay products they've purchased. This is the URL http://www.amazon.com/dp-applications.
application version	A specific, labeled iteration of an application that represents a functionally consistent set of deployable application code. A version points to an Amazon S3 object (a JAVA WAR file) that contains the application code.
ARN	See Amazon Resource Name.
asynchronous bounce	A type of bounce (p. 284) that occurs when a receiver (p. 301) initially accepts an email message for delivery and then subsequently fails to deliver it.
attribute	Similar to a column on a spreadsheet, an attribute represents a data category. In Amazon SimpleDB, an attribute has a name (such as *color*), which has a value (such as *blue*) when applied to a data item.
authentication	The process of proving your identity to a system.
Auto Scaling	A web service designed to launch or terminate instance (p. 293)s automatically based on user-defined policies, schedules, and health checks. See Also http://www.amazonaws.cn//autoscaling.
Auto Scaling group	A representation of multiple Amazon Elastic Compute Cloud (p. 281) instance (p. 293)s that share similar characteristics, and that are treated as a logical grouping for the purposes of instance scaling and management.
Availability Zone	A distinct location within a region (p. 301) that is insulated from failures in other Availability Zones, and provides inexpensive, low-latency network connectivity to other Availability Zones in the same region.
AWS	See Amazon Web Services.
AWS CloudFormation	A service for writing or changing templates that create and delete related AWS resources together as a unit. See Also http://www.amazonaws.cn/cloudformation.
AWS Consolidated Billing	A billing option that lets you get a single bill for multiple AWS accounts. See Also http://www.amazonaws.cn/consolidated-billing.
AWS Elastic Beanstalk	See Also http://www.amazonaws.cn/elasticbeanstalk.
AWS Import/Export	A service for transferring large amounts of data between AWS and portable storage devices. See Also http://www.amazonaws.cn/importexport.
AWS Identity and Access Management	A web service that enables Amazon Web Services (p. 282) customers to manage users and user permissions within AWS. See Also http://www.amazonaws.cn/iam.
AWS Management Console	A graphical interface to manage compute, storage, and other cloud resources. See Also http://www.amazonaws.cn/console.
AWS Multi-Factor Authentication	An optional AWS account security feature. Once you enable AWS MFA, you must provide a six-digit, single-use code in addition to your sign-in credentials whenever

	you access secure AWS web site pages or the AWS Management Console. You get this single-use code from an authentication device that you keep in your physical possession. See Also http://www.amazonaws.cn/mfa/.
AWS VPN CloudHub	Enables secure communication between branch offices using a simple hub-and-spoke model, with or without a VPC.

B

Numbers and Symbols (p. 280) | A (p. 280) | B (p. 284) | C (p. 285) | D (p. 287) | E (p. 289) | F (p. 290) | G (p. 291) | H (p. 291) | I (p. 292) | J (p. 294) | K (p. 294) | L (p. 295) | M (p. 295) | N (p. 297) | O (p. 298) | P (p. 298) | Q (p. 300) | R (p. 300) | S (p. 303) | T (p. 307) | U (p. 308) | V (p. 308) | W (p. 309) | X, Y, Z (p. 309)

basic monitoring	Monitoring of AWS-provided metrics derived at a 5-minute frequency.
batch	See document batch.
BGP ASN	Border Gateway Protocol Autonomous System Number. A unique identifier for a network, for use in BGP routing. Amazon EC2 supports all 2-byte ASN numbers in the range of 1 - 65334, with the exception of 7224, which is reserved.
blacklist	A list of IP addresses, email addresses, or domains that an Internet Service Provider (p. 293) suspects to be the source of spam (p. 305). The ISP blocks incoming emails from these addresses or domains.
block	A data set. Amazon EMR breaks large amounts of data into subsets. Each subset is called a data block. Amazon EMR assigns an ID to each block and uses a hash table to keep track of block processing.
block device	A storage device that supports reading and (optionally) writing data in fixed-size blocks, sectors, or clusters.
block device mapping	A mapping structure for every AMI and instance that specifies the block devices attached to the instance.
bootstrap action	A user-specified default or custom action that runs a script or an application on all nodes of a job flow before Hadoop starts.
Border Gateway Protocol Autonomous System Number	See BGP ASN.
bounce	A failed email delivery attempt.
breach	The condition in which a user-set threshold (upper or lower boundary) is passed. If the duration of the breach is significant, as set by a breach duration parameter, it can possibly start a scaling activity (p. 303).
bucket	A container for objects stored in Amazon S3. Every object is contained in a bucket. For example, if the object named `photos/puppy.jpg` is stored in the `johnsmith` bucket, then authorized users can access the object with the URL `http://johnsmith.s3.amazonaws.com/photos/puppy.jpg`.
bucket owner	Just as Amazon is the only owner of the domain name Amazon.com, only one person or organization can own a bucket in Amazon S3.
bundling	A commonly used term for creating an Amazon Machine Image (p. 282). It specifically refers to creating instance store-backed AMIs.

C

Numbers and Symbols (p. 280) | A (p. 280) | B (p. 284) | C (p. 285) | D (p. 287) | E (p. 289) | F (p. 290) | G (p. 291) | H (p. 291) | I (p. 292) | J (p. 294) | K (p. 294) | L (p. 295) | M (p. 295) | N (p. 297) | O (p. 298) | P (p. 298) | Q (p. 300) | R (p. 300) | S (p. 303) | T (p. 307) | U (p. 308) | V (p. 308) | W (p. 309) | X, Y, Z (p. 309)

cache cluster	A logical cache distributed over multiple cache node (p. 285)s. A cache cluster can be set up with a specific number of cache nodes.
cache cluster identifier	Customer-supplied identifier for the cache cluster that must be unique for that customer in an AWS region.
cache engine version	The version of the Memcached service that is running on the cache node.
cache node	A fixed-size chunk of secure, network-attached RAM. Each cache node runs an instance of the Memcached service, and has its own DNS name and port. Multiple types of cache nodes are supported, each with varying amounts of associated memory.
cache node type	EC2 instance type used to run the cache node.
cache parameter group	A container for cache engine parameter values that can be applied to one or more cache clusters.
cache security group	A group maintained by ElastiCache that combines ingress authorizations to cache nodes for hosts belonging to Amazon EC2 security groups specified through the console or the API or command line tools.
canned access policy	A standard access control policy that you can apply to a bucket or object. Options include: private, public-read, public-read-write, and authenticated-read.
canonicalization	The process of converting data into a standard format that a service such as Amazon S3 can recognize.
capacity	Each Auto Scaling group (p. 283) is defined with a minimum and maximum compute size. The amount of available compute size at any time is the current capacity. A scaling activity (p. 303) increases or decreases the capacity—within the defined minimum and maximum values.
Cascading	Cascading is an open-source Java library that provides a query API, a query planner, and a job scheduler for creating and running Hadoop MapReduce applications. Applications developed with Cascading are compiled and packaged into standard Hadoop-compatible JAR files similar to other native Hadoop applications.
certificate	A credential that some AWS products use to authenticate AWS accounts and users. Also known as an X.509 certificate. The certificate is paired with a private key.
chargeable resources	Features or services whose use incurs fees. Although some AWS products are free, others include charges. For example, in an AWS CloudFormation stack (p. 305), AWS resources that have been created incur charges. The amount charged depends on the usage load. Use the Amazon Web Services Simple Monthly Calculator at http://calculator.s3.amazonaws.com/calc5.html to estimate your cost prior to creating instances, stacks, or other resources.
CIDR block	Classless Inter-Domain Routing. An Internet protocol address allocation and route aggregation methodology. See Also http://en.wikipedia.org/wiki/CIDR_notation.

ClassicLink	A feature that allows you to link an EC2-Classic instance to a VPC, allowing your EC2-Classic instance to communicate with VPC instances using private IP addresses. See Also link to VPC, unlink from VPC.
CloudHub	See AWS VPN CloudHub.
cluster compute instance	A type of instance (p. 293) that provides a great amount of CPU power coupled with increased networking performance, making it well suited for High Performance Compute (HPC) applications and other demanding network-bound applications.
cluster placement group	A logical cluster compute instance (p. 286) grouping to provide lower latency and high-bandwidth connectivity between the instances.
CNAME	Canonical Name Record. A type of resource record in the Domain Name System (DNS) that specifies that the domain name is an alias of another, canonical domain name. More simply, it is an entry in a DNS table that lets you alias one fully qualified domain name to another.
complaint	The event in which a recipient (p. 301) who does not want to receive an email message clicks "Mark as Spam" within the email client, and the Internet Service Provider (p. 293) sends a notification to Amazon SES.
compound query	A search request that specifies multiple search criteria using the Amazon CloudSearch structured search syntax.
condition	Any restriction or detail about a permission. The condition is *D* in the statement "A has permission to do B to C where D applies."
conditional parameter	See mapping.
configuration API	The Amazon CloudSearch API that you use to create, configure, and manage search domains.
configuration template	A series of key–value pairs that define parameters for various AWS products so that AWS Elastic Beanstalk can provision them for an environment.
consistency model	The method a service uses to achieve high availability. For example, it could involve replicating data across multiple servers in a data center. See Also eventual consistency.
consistent read	When data is written or updated successfully, all copies of the data are updated in all AWS regions. However, it takes time for the data to propagate to all storage locations. A consistent read returns a result that reflects any writes that received a successful response before the read request—regardless of the region. By contrast, an eventually consistent read returns data from only one region and might not show the most recent write information. See Also eventual consistency.
console	See AWS Management Console.
Consolidated Billing	See AWS Consolidated Billing.
cooldown period	Amount of time during which Auto Scaling does not allow the desired size of the Auto Scaling group (p. 283) to be changed by any other notification from a CloudWatch alarm (p. 281).
core node	An EC2 instance (p. 289) that runs Hadoop map and reduce tasks and stores data using the Hadoop Distributed File System (HDFS). Core nodes are managed by the master node (p. 296), which assigns Hadoop tasks to nodes and monitors their

	status. The EC2 instances you assign as core nodes are capacity that must be allotted for the entire job flow run. Because core nodes store data, you can't remove them from a job flow. However, you can add more core nodes to a running job flow. Core nodes run both the DataNodes and TaskTracker Hadoop daemons.
corpus	A collection of data that you want to search.
credentials	Also called *access credentials* or *security credentials*. In authentication and authorization, a system uses credentials to identify who is making a call and whether to allow the requested access. In AWS, these credentials are typically the access key ID (p. 280) and the secret access key (p. 303).
customer gateway	A router or software application on your side of a VPN tunnel that is managed by Amazon VPC. The internal interfaces of the customer gateway are attached to one or more devices in your home network. The external interface is attached to the VPG (p. 309) across the VPN tunnel.

D

Numbers and Symbols (p. 280) | A (p. 280) | B (p. 284) | C (p. 285) | D (p. 287) | E (p. 289) | F (p. 290) | G (p. 291) | H (p. 291) | I (p. 292) | J (p. 294) | K (p. 294) | L (p. 295) | M (p. 295) | N (p. 297) | O (p. 298) | P (p. 298) | Q (p. 300) | R (p. 300) | S (p. 303) | T (p. 307) | U (p. 308) | V (p. 308) | W (p. 309) | X, Y, Z (p. 309)

dashboard	See service health dashboard.
database engine	The database software and version running on the DB instance (p. 287).
database name	The name of a database hosted in a DB instance (p. 287). A DB instance can host multiple databases, but databases hosted by the same DB instance must each have a unique name within that instance.
DB compute class	Size of the database compute platform used to run the instance.
DB instance	An isolated database environment running in the cloud. A DB instance can contain multiple user-created databases.
DB instance identifier	User-supplied identifier for the DB instance. The identifier must be unique for that user in an AWS region (p. 301).
DB parameter group	A container for database engine parameter values that apply to one or more DB instance (p. 287)s.
DB security group	A method that controls access to the DB instance (p. 287). By default, network access is turned off to DB instances. After ingress is configured for a security group, the same rules apply to all DB instances associated with that group.
DB snapshot	A user-initiated point backup of a DB instance.
Dedicated Instance	An instance that is physically isolated at the host hardware level and launched within a VPC.
Dedicated Reserved Instance	An option you purchase to guarantee that sufficient capacity will be available to launch Dedicated Instances into a VPC.
delete marker	An object with a key and version ID, but without content. Amazon S3 inserts delete markers automatically into versioned buckets when an object is deleted.

deliverability	The likelihood that an email message will arrive at its intended destination.
deliveries	The number of emails, sent through Amazon SES, that were accepted by an Internet Service Provider (p. 293) for delivery to recipient (p. 301)s over a period of time.
deny	The result of a policy statement that includes deny as the effect, so that a specific action or actions are expressly forbidden for a user, group, or role. Explicit deny take precedence over explicit allow (p. 281).
detailed monitoring	Monitoring of AWS-provided metrics derived at a 1-minute frequency.
Description property	A property added to parameters, resources, resource properties, mappings, and outputs, to help you to document AWS CloudFormation template elements.
dimension	A name/value pair (for example, InstanceType=m1.small, or EngineName=mysql), that contains additional information to identify a metric.
discussion forums	A place where AWS users can post technical questions and feedback to help accelerate their development efforts and to engage with the AWS community. The discussion forums are located at http://www.amazonaws.cn/forums/.
distributed cache	A Hadoop feature that allow you to transfer files from a distributed file system to the local file system. It can distribute data and text files as well as more complex types such as archives and JARs.
distribution	A link between an origin server (such as an Amazon S3 bucket) and a domain name, which CloudFront automatically assigns. Through this link, CloudFront identifies the object you have stored in your origin server (p. 298).
DKIM	DomainKeys Identified Mail. A standard that email senders use to sign their messages. ISPs use those signatures to verify that messages are legitimate. For more information, see http://www.dkim.org.
DNS	See Domain Name System (DNS).
document	Represents an item that can be returned as a search result in Amazon CloudSearch. Each document has a collection of fields that contain the data that can be searched or returned. The value of a field can be either a string or a number. Each document must have a unique ID and at least one field.
document batch	A collection of add and delete document operations for Amazon CloudSearch. You use the document service API to submit batches to update the data in your search domain.
document service API	The Amazon CloudSearch API that you use to submit document batches to update the data in a search domain.
document service endpoint	The URL that you connect to when sending document updates to an Amazon CloudSearch domain. Each search domain has a unique document service endpoint that remains the same for the life of the domain.
domain	All Amazon SimpleDB information is stored in domains. Domains are like tables that contain similar data. You can execute queries against a domain, but cannot execute joins between domains. See Also search domain.
Domain Name System (DNS)	A distributed naming system that associates network information with human-readable domain names on the Internet.

Donation button	An HTML-coded button to provide an easy and secure way for US-based, IRS-certified 501(c)3 nonprofit organizations to solicit donations.

E

Numbers and Symbols (p. 280) | A (p. 280) | B (p. 284) | C (p. 285) | D (p. 287) | E (p. 289) | F (p. 290) | G (p. 291) | H (p. 291) | I (p. 292) | J (p. 294) | K (p. 294) | L (p. 295) | M (p. 295) | N (p. 297) | O (p. 298) | P (p. 298) | Q (p. 300) | R (p. 300) | S (p. 303) | T (p. 307) | U (p. 308) | V (p. 308) | W (p. 309) | X, Y, Z (p. 309)

EBS	See Amazon Elastic Block Store.
EC2	See Amazon Elastic Compute Cloud.
EC2 compute unit	An AWS standard for compute CPU and memory. This measure enables you to evaluate the CPU capacity of different EC2 instance types.
EC2 instance	In Amazon EC2, this is simply an instance. Other AWS services use the term EC2 instance to distinguish these instances from other types of instances they support.
edge location	A site that CloudFront uses to cache copies of your content for faster delivery to users at any location.
Elastic Block Store	See Amazon Elastic Block Store.
Elastic IP address	A fixed (static) IP address that you have allocated in Amazon EC2 or Amazon VPC and then attached to an instance. Elastic IP addresses are associated with your account, not a specific instance. They are *elastic* because you can easily allocate, attach, detach, and free them as your needs change. Unlike traditional static IP addresses, Elastic IP addresses allow you to mask instance or Availability Zone failures by rapidly remapping your public IP addresses to another instance.
Elastic Load Balancing	A web service that improves an application's availability by distributing incoming traffic between two or more EC2 instance (p. 289)s. See Also http://www.amazonaws.cn/elasticloadbalancing.
elastic network interface	An additional network interface that can be attached to an instance (p. 293). ENIs include a primary private IP address, one or more secondary private IP addresses, an elastic IP address (optional), a MAC address, membership in specified security groups, a description, and a source/destination check flag. You can create an ENI, attach it to an instance, detach it from an instance, and attach it to another instance.
endpoint	A URL that identifies a host and port as the entry point for a web service. Every web service request contains an endpoint. Most AWS products provide regional endpoints to enable faster connectivity. For more information, see Regions and Endpoints in the *Amazon Web Services General Reference* ElastiCache: The DNS name of a cache node (p. 285). Amazon RDS: The DNS name of a DB instance (p. 287). AWS CloudFormation: The DNS name or IP address of the server that receives an HTTP request.
endpoint port	ElastiCache: The port number used by a cache node (p. 285). Amazon RDS: The port number used by a DB instance (p. 287).

environment	A specific running instance of an application (p. 283). The application has a CNAME and includes an application version and a customizable configuration (which is inherited from the default container type).
environment configuration	A collection of parameters and settings that define how an environment and its associated resources behave.
ephemeral store	See instance store.
epoch	The date from which time is measured. For most Unix environments, the epoch is January 1, 1970.
eventual consistency	The method through which AWS products achieve high availability, which involves replicating data across multiple servers in Amazon's data centers. When data is written or updated and "Success" is returned, all copies of the data are updated. However, it takes time for the data to propagate to all storage locations. The data will eventually be consistent, but an immediate read might not show the change. Consistency is usually reached within seconds, but a high system load might increase this time.
eventually consistent read	See consistent read.
eviction	An *eviction* occurs when CloudFront deletes an object from an edge location (p. 289) before its expiration time. If an object in an edge location isn't frequently requested, CloudFront might evict the object (remove the object before its expiration date) to make room for objects that are more popular.
exbibyte	A contraction of exa binary byte, an exbibyte is 2^60 or 1,152,921,504,606,846,976 bytes. An exabyte (EB) is 10^18 or 1,000,000,000,000,000,000 bytes. 1,024 EiB is a zebibyte (p. 310).
expiration	*Expiration* occurs when CloudFront stops serving an object from an edge location (p. 289). The next time the edge location needs to serve that object, CloudFront gets a new copy from the origin server (p. 298).
explicit launch permission	An Amazon Machine Image (p. 282) launch permission granted to a specific AWS account.
exponential backoff	A strategy that incrementally increases the wait between retry attempts in order to reduce the load on the system and increase the likelihood that repeated requests will succeed. For example, client applications might wait up to 400 milliseconds before attempting the first retry, up to 1600 milliseconds before the second, up to 6400 milliseconds (6.4 seconds) before the third, and so on.
expression	A numeric expression that you can use to control how search hits are sorted. You can construct Amazon CloudSearch expressions using numeric fields, other rank expressions, a document's default relevance _score, and standard numeric operators and functions. When you use the `sort` option to specify an expression in a search request, the expression is evaluated for each search hit and the hits are listed according to their expression values.

F

Numbers and Symbols (p. 280) | A (p. 280) | B (p. 284) | C (p. 285) | D (p. 287) | E (p. 289) | F (p. 290) | G (p. 291) | H (p. 291) | I (p. 292) | J (p. 294) | K (p. 294) | L (p. 295) | M (p. 295) | N (p. 297) | O (p. 298) | P (p. 298) | Q (p. 300) | R (p. 300) | S (p. 303) | T (p. 307) | U (p. 308) | V (p. 308) | W (p. 309) | X, Y, Z (p. 309)

facet	An Amazon CloudSearch index field that represents a category that you want to use to refine and filter search results.

facet enabled	An Amazon CloudSearch index field option that enables facet information to be calculated for the field.
FBL	See feedback loop.
federated identity management	Allows individuals to sign in to different networks or services, using the same group or personal credentials to access data across all networks. With identity federation in AWS, external identities (federated users) are granted secure access to resources in an AWS account without having to create IAM users. These external identities can come from a corporate identity store (such as LDAP or Windows Active Directory) or from a third party (such as Login with Amazon, Facebook, or Google). AWS federation also supports SAML 2.0.
federated user	See federated identity management.
feedback loop	The mechanism by which a mailbox provider (for example, an Internet Service Provider (p. 293)) forwards a recipient (p. 301)'s complaint (p. 286) back to the sender (p. 304).
field weight	The relative importance of a text field in a search index. Field weights control how much matches in particular text fields affect a document's relevance _score.
filter	A criterion you specify to limit the results when you list or describe your Amazon EC2 resources.
filter query	A way to filter search results without affecting how the results are scored and sorted. Specified with the Amazon CloudSearch `fq` parameter.
FIM	See federated identity management.
format version	See template format version.
forums	See discussion forums.
function	See intrinsic function.
fuzzy search	A simple search query that uses approximate string matching (fuzzy matching) to correct for typographical errors and misspellings.

G

Numbers and Symbols (p. 280) | A (p. 280) | B (p. 284) | C (p. 285) | D (p. 287) | E (p. 289) | F (p. 290) | G (p. 291) | H (p. 291) | I (p. 292) | J (p. 294) | K (p. 294) | L (p. 295) | M (p. 295) | N (p. 297) | O (p. 298) | P (p. 298) | Q (p. 300) | R (p. 300) | S (p. 303) | T (p. 307) | U (p. 308) | V (p. 308) | W (p. 309) | X, Y, Z (p. 309)

geospatial search	A search query that uses locations specified as a latitude and longitude to determine matches and sort the results.
gibibyte	A contraction of giga binary byte, a gibibyte is 2^30 or 1,073,741,824 bytes. A gigabyte (GB) is 10^9 or 1,000,000,000 bytes. 1,024 GiB is a tebibyte (p. 307).

H

Numbers and Symbols (p. 280) | A (p. 280) | B (p. 284) | C (p. 285) | D (p. 287) | E (p. 289) | F (p. 290) | G (p. 291) | H (p. 291) | I (p. 292) | J (p. 294) | K (p. 294) | L (p. 295) | M (p. 295) | N (p. 297) | O (p. 298) | P (p. 298) | Q (p. 300) | R (p. 300) | S (p. 303) | T (p. 307) | U (p. 308) | V (p. 308) | W (p. 309) | X, Y, Z (p. 309)

Hadoop	See http://hadoop.apache.org.

hard bounce	A persistent email delivery failure such as "mailbox does not exist."
hardware VPN	A hardware-based IPsec VPN connection over the Internet.
HDFS	Hadoop Distributed File System. The HDFS file system stores large files across multiple machines. It achieves reliability by replicating the data across multiple hosts, and hence does not require RAID storage on hosts.
health check	A system call to check on the health status of each instance in an Auto Scaling group.
high-quality email	Email that recipients find valuable and want to receive. Value means different things to different recipients and can come in the form of offers, order confirmations, receipts, newsletters, etc.
highlights	Excerpts returned with Amazon CloudSearch results that show where the search terms appear within the text of the matching documents.
highlight enabled	An Amazon CloudSearch index field option that enables matches within the field to be highlighted.
hit	A document that matches the criteria specified in a search request. Also referred to as a *search result*.
Hive	An open source, data warehouse and analytic package that runs on top of Hadoop. Hive scripts use an SQL-like language called Hive QL (query language) that abstracts the MapReduce programming model and supports typical data warehouse interactions.
HMAC	Hash-based Message Authentication Code. A specific construction for calculating a message authentication code (MAC) involving a cryptographic hash function in combination with a secret key. You can use it to verify both the data integrity and the authenticity of a message at the same time. AWS calculates the HMAC using a standard, cryptographic hash algorithm, such as SHA-256.
hosted zone	A collection of resource record sets that Amazon Route 53 hosts. Like a traditional DNS zone file, a hosted zone represents a collection of records that are managed together under a single domain name.
HVM virtualization	Hardware Virtual Machine virtualization. Allows the guest VM to run as though it is on a native hardware platform, except that it still uses paravirtual (PV) network and storage drivers for improved performance. See Also PV virtualization.

I

Numbers and Symbols (p. 280) | A (p. 280) | B (p. 284) | C (p. 285) | D (p. 287) | E (p. 289) | F (p. 290) | G (p. 291) | H (p. 291) | I (p. 292) | J (p. 294) | K (p. 294) | L (p. 295) | M (p. 295) | N (p. 297) | O (p. 298) | P (p. 298) | Q (p. 300) | R (p. 300) | S (p. 303) | T (p. 307) | U (p. 308) | V (p. 308) | W (p. 309) | X, Y, Z (p. 309)

image	See Amazon Machine Image.
import/export station	A machine that uploads or downloads your data to, or from, Amazon S3.
import log	A report that contains details about how AWS Import/Export processed your data.
index	See search index.

index field	A name-value pair that is included in an Amazon CloudSearch domain's index. An index field can contain text or numeric data, dates, or a location.
indexing options	Configuration settings that define an Amazon CloudSearch domain's index fields, how document data is mapped to those index fields, and how the index fields can be used.
instance	A copy of an Amazon Machine Image running as a virtual server in the AWS cloud.
instance family	A general instance type (p. 293) grouping using either storage or CPU capacity.
instance group	A Hadoop cluster contains one master instance group that contains one master node (p. 296), a core instance group containing one or more core node (p. 286) and an optional task node (p. 307) instance group, which can contain any number of task nodes.
instance store	Disk storage that is physically attached to the host computer for an EC2 instance, and therefore has the same lifespan as the instance. When the instance terminates, you lose any data in the instance store.
instance store-backed AMI	Instances launched from this type of AMI use an instance store volume as the root device. Compare this with instances launched from Amazon EBS-backed AMIs, which use an Amazon EBS volume as the root device.
instance type	A specification that defines the memory, CPU, storage capacity, and hourly cost for an instance. Some instance types are designed for standard applications, whereas others are designed for CPU-intensive, memory-intensive applications, and so on.
Internet gateway	Connects a network to the Internet. You can route traffic for IP addresses outside your VPC (p. 309) to the Internet gateway.
Internet Service Provider	A company that provides subscribers with access to the Internet. Many ISPs are also mailbox provider (p. 295)s. Mailbox providers are sometimes referred to as ISPs, even if they only provide mailbox services.
intrinsic function	A special action in a template that assigns values to properties not available until runtime. These functions follow the format *Fn::Attribute*, such as `Fn::GetAtt`. Arguments for intrinsic functions can be parameters, pseudo parameters, or the output of other intrinsic functions.
IP address	All EC2 instances are assigned two IP addresses at launch, which are directly mapped to each other through network address translation (NAT): a private IP address (following RFC 1918) and a public IP address. Instances launched in a VPC are assigned only a private IP address. Instances launched in your default VPC are assigned both a private IP address and a public IP address.
ISP	See Internet Service Provider.
issuer	The issuer is the person who writes a policy to grant permissions to a resource. The issuer (by definition) is always the resource owner. AWS does not permit Amazon SQS users to create policies for resources they don't own. If John is the resource owner, AWS authenticates John's identity when he submits the policy he's written to grant permissions for that resource.
item	Similar to rows on a spreadsheet, items represent individual objects that contain one or more value-attribute pairs.
item name	An identifier for an item. The identifier must be unique within the domain (p. 288).

J

Numbers and Symbols (p. 280) | A (p. 280) | B (p. 284) | C (p. 285) | D (p. 287) | E (p. 289) | F (p. 290) | G (p. 291) | H (p. 291) | I (p. 292) | J (p. 294) | K (p. 294) | L (p. 295) | M (p. 295) | N (p. 297) | O (p. 298) | P (p. 298) | Q (p. 300) | R (p. 300) | S (p. 303) | T (p. 307) | U (p. 308) | V (p. 308) | W (p. 309) | X, Y, Z (p. 309)

job flow	A job flow specifies the complete processing of the data. It's comprised of one or more steps, which specify all of the functions to be performed on the data.
job ID	A five-character, alphanumeric string that uniquely identifies a storage device in your shipment. AWS issues the job ID in response to a `CREATE JOB` email command.
job prefix	The AWS Import/Export process generates a log file. The log file name always ends with the phrase *import-log-* followed by your Job ID. There is a remote chance that you already have an object with this name. To avoid a key collision, you can add an optional prefix to the log file. See Also key prefix.
JSON	JavaScript Object Notation. A lightweight data-interchange format. For information about JSON, see http://www.json.org/.
junk folder	The location where email messages that various filters determine to be of lesser value are collected so that they do not arrive in the recipient (p. 301)'s inbox, but are still accessible to the recipient. This is also referred to as a spam (p. 305) or bulk folder.

K

Numbers and Symbols (p. 280) | A (p. 280) | B (p. 284) | C (p. 285) | D (p. 287) | E (p. 289) | F (p. 290) | G (p. 291) | H (p. 291) | I (p. 292) | J (p. 294) | K (p. 294) | L (p. 295) | M (p. 295) | N (p. 297) | O (p. 298) | P (p. 298) | Q (p. 300) | R (p. 300) | S (p. 303) | T (p. 307) | U (p. 308) | V (p. 308) | W (p. 309) | X, Y, Z (p. 309)

key	A credential that identifies an AWS account or user to AWS (such as the AWS secret access key (p. 303)). Amazon S3, Amazon EMR: The unique identifier for an object in a bucket. Every object in a bucket has exactly one key. Because a bucket and key together uniquely identify each object, you can think of Amazon S3 as a basic data map between the *bucket* + *key*, and the object itself. You can uniquely address every object in Amazon S3 through the combination of the web service endpoint, bucket name, and key, for example: `http://doc.s3.amazonaws.com/2006-03-01/AmazonS3.wsdl`, where `doc` is the name of the bucket, and `2006-03-01/AmazonS3.wsdl` is the key. AWS Import/Export: The name of an object in Amazon S3. It is a sequence of Unicode characters whose UTF-8 encoding cannot exceed 1024 bytes. If a key, for example, logPrefix + import-log-JOBID, is longer than 1024 bytes, AWS Elastic Beanstalk returns an `InvalidManifestField` error. IAM: In the context of writing a policy (p. 299): A specific characteristic that is the basis for restricting access (such as the current time, or the IP address of the requester).
key pair	A set of security credentials you use to prove your identity electronically. A key pair consists of a private key and a public key.
key prefix	A logical grouping of the objects in a bucket (p. 284). The prefix value is similar to a directory name that enables you to store similar data under the same directory in a bucket.

kibibyte	A contraction of kilo binary byte, a kibibyte is 2^10 or 1,024 bytes. A kilobyte (KB) is 10^3 or 1,000 bytes. 1,024 KiB is a mebibyte (p. 296).

L

Numbers and Symbols (p. 280) | A (p. 280) | B (p. 284) | C (p. 285) | D (p. 287) | E (p. 289) | F (p. 290) | G (p. 291) | H (p. 291) | I (p. 292) | J (p. 294) | K (p. 294) | L (p. 295) | M (p. 295) | N (p. 297) | O (p. 298) | P (p. 298) | Q (p. 300) | R (p. 300) | S (p. 303) | T (p. 307) | U (p. 308) | V (p. 308) | W (p. 309) | X, Y, Z (p. 309)

launch configuration	A set of descriptive parameters used to create new EC2 instances in an Auto Scaling activity. A template that an Auto Scaling group (p. 283) uses to launch new EC2 instances. The launch configuration contains information such as the Amazon Machine Image (p. 282) ID, the instance type, key pairs, security groups, and block device mappings, among other configuration settings.
launch permission	An Amazon Machine Image (p. 282) (AMI) attribute that allows users to launch an AMI.
lifecycle	The lifecycle state of the EC2 instance (p. 289) contained in an AutoScalingGroup. EC2 instances progress through several states over their lifespan; these include *Pending*, *InService*, *Terminating* and *Terminated*.
link to VPC	The process of linking (or attaching) an EC2-Classic instance to a ClassicLink-enabled VPC. See Also ClassicLink, unlink from VPC.
load balancer	A load balancer is a combination of a DNS name and a set of ports, which together provide a destination for all requests intended for your application. A load balancer can distribute traffic to multiple application instances across every Availability Zone (p. 283) within a region (p. 301). Load balancers can span multiple Availability Zones within an Amazon EC2 region, but they cannot span multiple regions.
logical name	A case-sensitive unique string within an AWS CloudFormation template that identifies a resource (p. 302), mapping (p. 296), parameter, or output. In an AWS CloudFormation template, each parameter, resource, property, mapping, and output must be declared with a unique logical name. You use the logical name when dereferencing these items using the `Ref` function.

M

Numbers and Symbols (p. 280) | A (p. 280) | B (p. 284) | C (p. 285) | D (p. 287) | E (p. 289) | F (p. 290) | G (p. 291) | H (p. 291) | I (p. 292) | J (p. 294) | K (p. 294) | L (p. 295) | M (p. 295) | N (p. 297) | O (p. 298) | P (p. 298) | Q (p. 300) | R (p. 300) | S (p. 303) | T (p. 307) | U (p. 308) | V (p. 308) | W (p. 309) | X, Y, Z (p. 309)

machine utilization	The amount of machine capacity used to complete a particular request (for example SELECT, GET, PUT, and so on), normalized to the hourly capacity of a standard processor. Machine utilization is measured in machine hour increments.
Mail Transfer Agent (MTA)	Software that transports email messages from one computer to another by using a client-server architecture.
mailbox provider	An organization that provides email mailbox hosting services. Mailbox providers are sometimes referred to as Internet Service Provider (p. 293)s, even if they only provide mailbox services.

mailbox simulator	A set of email addresses that you can use to test an Amazon SES-based email sending application without sending messages to actual recipients. Each email address represents a specific scenario (such as a bounce or complaint) and generates a typical response that is specific to the scenario.
main route table	The default route table that any new VPC subnet uses for routing. You can associate a subnet with a different route table of your choice. You can also change which route table is the main route table.
manifest	When sending a *create job* request for an import or export operation you describe your job in a text file called a manifest. The manifest file is a YAML-formatted file that specifies how to transfer data between your storage device and the AWS cloud.
MapReduce	See http://hadoop.apache.org/docs/r1.2.0/mapred_tutorial.html.
mapper	An executable that splits the raw data into key/value pairs. The reducer uses the output of the mapper, called the *intermediate results*, as its input.
mapping	A way to add conditional parameter values to an AWS CloudFormation template. You specify mappings in the template's optional Mappings section and retrieve the desired value using the `FN::FindInMap` function.
marker	See pagination.
master node	A process running on an Amazon Machine Image (p. 282) that keeps track of the work its core and task nodes complete.
maximum price	The maximum price you will pay to launch one or more Spot Instances. If your maximum price exceeds the current Spot Price (p. 305) and your restrictions are met, Amazon EC2 launches instances on your behalf.
maximum send rate	The maximum number of emails that you can send per second using Amazon SES.
mebibyte	A contraction of mega binary byte, a mebibyte is 2^20 or 1,048,576 bytes. A megabyte (MB) is 10^6 or 1,000,000 bytes. 1,024 MiB is a gibibyte (p. 291).
member resources	See resource.
message ID	Amazon SES: A unique identifier that is assigned to every email message that is sent. Amazon SQS: The identifier returned when you send a message to a queue.
metadata	Amazon S3, Amazon EMR: A set of name/value pairs that describe the object. These include default metadata such as the date last modified and standard HTTP metadata such as Content-Type. Users can also specify custom metadata at the time they store an object. Amazon EC2: Data about an EC2 instance (p. 289) that the instance can retrieve to determine things about itself, such as the instance type, the IP address, and so on.
metric	An element of time-series data defined by a unique combination of exactly one namespace, exactly one metric name, and between zero and ten dimensions. Metrics and the statistics derived from them are the basis of Amazon CloudWatch.
metric name	The primary identifier of a metric, used in combination with a namespace and optional dimensions.

MFA	See AWS Multi-Factor Authentication.
micro instance	A type of EC2 instance (p. 289) that is more economical to use if you have occasional bursts of high CPU activity.
MIME	See Multipurpose Internet Mail Extensions (MIME).
MTA	See Mail Transfer Agent (MTA).
Multi-AZ deployment	A primary DB instance (p. 287) that has a synchronous standby replica in a different Availability Zone (p. 283). The primary DB instance is synchronously replicated across Availability Zones to the standby replica.
Multi-Factor Authentication	See AWS Multi-Factor Authentication.
multi-valued attribute	An attribute with more than one value.
multipart upload	A feature that allows you to upload a single object as a set of parts.
Multipurpose Internet Mail Extensions (MIME)	An Internet standard that extends the email protocol to include non-ASCII text and non-text elements like attachments.
Multitool	A Cascading (p. 285) application that provides a simple command-line interface for managing large datasets.

N

Numbers and Symbols (p. 280) | A (p. 280) | B (p. 284) | C (p. 285) | D (p. 287) | E (p. 289) | F (p. 290) | G (p. 291) | H (p. 291) | I (p. 292) | J (p. 294) | K (p. 294) | L (p. 295) | M (p. 295) | N (p. 297) | O (p. 298) | P (p. 298) | Q (p. 300) | R (p. 300) | S (p. 303) | T (p. 307) | U (p. 308) | V (p. 308) | W (p. 309) | X, Y, Z (p. 309)

namespace	An abstract container that provides context for the items (names, or technical terms, or words) it holds, and allows disambiguation of homonym items residing in different namespaces.
NAT	Network address translation.
NAT instance	An instance that is configured to perform NAT (p. 297) in a VPC. A NAT instance enables private instances in the VPC to initiate Internet-bound traffic without being directly reachable from the Internet.
network ACL	An optional layer of security that acts as a firewall for controlling traffic in and out of a subnet. You can associate multiple subnets with a single network ACL, but a subnet can be associated with only one network ACL at a time.
node	After an Amazon Machine Image (p. 282) is launched, the resulting running system is referred to as a node. All instances based on the same AMI are identical at start-up. Any information about the node is lost when the node terminates or fails.
NoEcho	A property of AWS CloudFormation parameters that will prevent the otherwise default reporting of names and values of a template parameter. Declaring the *`NoEcho`* property causes the parameter value to be masked with asterisks in the report by the `cfn-describe-stacks` command.
null object	A null object is one whose version ID is null. Amazon S3 adds a null object to a bucket when versioning (p. 309) for that bucket is suspended. It is possible to have only one null object for each key in a bucket.

O

Numbers and Symbols (p. 280) | A (p. 280) | B (p. 284) | C (p. 285) | D (p. 287) | E (p. 289) | F (p. 290) | G (p. 291) | H (p. 291) | I (p. 292) | J (p. 294) | K (p. 294) | L (p. 295) | M (p. 295) | N (p. 297) | O (p. 298) | P (p. 298) | Q (p. 300) | R (p. 300) | S (p. 303) | T (p. 307) | U (p. 308) | V (p. 308) | W (p. 309) | X, Y, Z (p. 309)

object	Amazon S3: The fundamental entity type stored in Amazon S3. Objects consist of object data and metadata. The data portion is opaque to Amazon S3. CloudFront: Any entity that can be served either over HTTP or a version of RTMP.
on-demand instance	An Amazon EC2 pricing option that charges you for compute capacity by the hour with no long-term commitment.
operation	An API function. Also called an *action*.
origin access identity	Also called OAI. A virtual identity you use when giving your distribution permission to fetch a private object from your origin server (Amazon S3 bucket).
origin server	The Amazon S3 bucket or custom origin containing the definitive original version of the content you deliver through CloudFront.

P

Numbers and Symbols (p. 280) | A (p. 280) | B (p. 284) | C (p. 285) | D (p. 287) | E (p. 289) | F (p. 290) | G (p. 291) | H (p. 291) | I (p. 292) | J (p. 294) | K (p. 294) | L (p. 295) | M (p. 295) | N (p. 297) | O (p. 298) | P (p. 298) | Q (p. 300) | R (p. 300) | S (p. 303) | T (p. 307) | U (p. 308) | V (p. 308) | W (p. 309) | X, Y, Z (p. 309)

pagination	Some APIs that return a potentially large list of records can return a subset by using a value to set the maximum number of returned records. They then provide a marker, which identifies the last record returned so that in a subsequent call, the user can get the next sequence of records.
paid AMI	An Amazon Machine Image (AMI) that you sell to other Amazon EC2 users using AWS Marketplace.
paravirtual virtualization	See PV virtualization.
part	In a multipart upload request, each part is a contiguous portion of the object's data.
PAT	Port address translation.
pebibyte	A contraction of peta binary byte, a pebibyte is 2^50 or 1,125,899,906,842,624 bytes. A petabyte (PB) is 10^15 or 1,000,000,000,000,000 bytes. 1,024 PiB is an exbibyte (p. 290).
period	See sampling period.
permission	A statement within a policy (p. 299) that allows or denies access to a particular resource. You can state any permission like this: "A has permission to do B to C." For example, Jane (A) has permission to read messages (B) from John's Amazon SQS queue (C). Whenever Jane sends a request to Amazon SQS to use John's queue, the service checks to see if she has permission and if the request satisfies the conditions John set forth in the permission.
persistent storage	A long-term data storage solution. Options within AWS are: Amazon S3, Amazon EBS, and Amazon SimpleDB.

physical name	A unique label AWS CloudFormation assigns to each resource when creating a stack (p. 305). Some AWS CloudFormation commands accept the physical name as a value with the `--physical-name` parameter.
Pig	An open-source Apache library that runs on top of Hadoop. The library takes SQL-like commands written in a language called Pig Latin and converts those commands into MapReduce job flows.
policy	A document defining permissions that apply to a user, group, or role; the permissions in turn determine what users can do in AWS. A policy typically allow (p. 281)s access to specific actions, and can optionally grant that the actions are allowed for specific resources, like EC2 instances, S3 buckets, and so on. Policies can also explicitly deny (p. 288) access. Auto Scaling: An object that stores the information needed to launch or terminate instances for an Auto Scaling group. Executing the policy causes instances to be launched or terminated. You can configure an alarm (p. 281) to invoke an Auto Scaling policy.
pre-signed URL	A URL that uses query string authentication (p. 300).
prefix	See job prefix.
Premium Support	A one-on-one, fast-response support channel that AWS customers can subscribe to for support for AWS infrastructure services. See Also http://www.amazonaws.cn/premiumsupport/.
principal	The user, service, or account that receives permissions that are defined in a policy (p. 299). The principal is A in the statement "A has permission to do B to C."
private IP address	All EC2 instances are assigned two IP addresses at launch, which are directly mapped to each other through Network Address Translation (NAT): a private address (following RFC 1918) and a public address. *Exception:* Instances launched in Amazon VPC are assigned only a private IP address.
private subnet	A VPC subnet whose instances cannot be reached from the Internet.
product code	The product code is an identifier provided by AWS when you submit a product to AWS Marketplace.
properties	See resource property.
property rule	A JSON (p. 294)-compliant markup standard for declaring properties, mappings, and output values in an AWS CloudFormation template.
Provisioned IOPS	A storage option designed to deliver fast, predictable, and consistent I/O performance. When you specify an IOPS rate while creating a DB instance, Amazon RDS provisions that IOPS rate for the lifetime of the DB instance.
pseudo parameter	A predefined setting, such as `AWS:StackName` that can be used in AWS CloudFormation templates without having to declare them. You can use pseudo parameters anywhere you can use a regular parameter.
public AMI	An Amazon Machine Image (p. 282) that all AWS accounts have permission to launch.
public data set	A large set of public data that can be seamlessly integrated into AWS cloud-based applications. Amazon stores public data sets at no charge to the community and, like all AWS services, users pay only for the compute and storage they use for

	their own applications. These data sets currently include data from the Human Genome Project, the U.S. Census, Wikipedia, and other sources. See Also http://www.amazonaws.cn/publicdatasets.
public IP address	All EC2 instances are assigned two IP addresses at launch, which are directly mapped to each other through Network Address Translation (NAT): a private address (following RFC 1918) and a public address. *Exception:* Instances launched in Amazon VPC are assigned only a private IP address.
public subnet	A subnet whose instances can be reached from the Internet.
PV virtualization	Paravirtual virtualization. Allows guest VMs to run on host systems that do not have special support extensions for full hardware and CPU virtualization. Because PV guests run a modified operating system that does not use hardware emulation, they cannot provide hardware-related features such as enhanced networking or GPU support. See Also HVM virtualization.

Q

Numbers and Symbols (p. 280) | A (p. 280) | B (p. 284) | C (p. 285) | D (p. 287) | E (p. 289) | F (p. 290) | G (p. 291) | H (p. 291) | I (p. 292) | J (p. 294) | K (p. 294) | L (p. 295) | M (p. 295) | N (p. 297) | O (p. 298) | P (p. 298) | Q (p. 300) | R (p. 300) | S (p. 303) | T (p. 307) | U (p. 308) | V (p. 308) | W (p. 309) | X, Y, Z (p. 309)

Query	A type of HTTP-based request interface that generally uses only the GET or POST HTTP method and a query string with parameters. See Also REST, REST-Query.
query string authentication	An AWS feature that lets you place the authentication information in the HTTP request query string instead of in the Authorization header, which enables URL-based access to objects in a bucket.
queue	A sequence of messages or jobs held in temporary storage awaiting transmission or processing.
queue URL	A URL that uniquely identifies a queue.
quota	Amazon RDS: The maximum number of DB instance (p. 287)s and available storage you can use. ElastiCache: The maximum number of the following items: • The number of cache clusters for each AWS account • The number of cache nodes per cache cluster • The total number of cache nodes per AWS account across all cache clusters created by that AWS account

R

Numbers and Symbols (p. 280) | A (p. 280) | B (p. 284) | C (p. 285) | D (p. 287) | E (p. 289) | F (p. 290) | G (p. 291) | H (p. 291) | I (p. 292) | J (p. 294) | K (p. 294) | L (p. 295) | M (p. 295) | N (p. 297) | O (p. 298) | P (p. 298) | Q (p. 300) | R (p. 300) | S (p. 303) | T (p. 307) | U (p. 308) | V (p. 308) | W (p. 309) | X, Y, Z (p. 309)

range GET	A range GET specifies a byte range of data to get for a download. If an object is large, you can break up a download into smaller units by sending multiple range GET requests that each specify a different byte range to GET.
raw email	A type of *sendmail* request that allows you to specify the email headers and MIME types.
RDS	See Amazon Relational Database Service.
read replica	An active copy of another DB instance. Any updates to the data on the source DB instance are replicated to the read replica DB instance using the built-in replication feature of MySQL 5.1.
receipt handle	An identifier you get when you receive a message from the queue. This identifier is required to delete a message from the queue or when changing a message's visibility timeout.
receiver	The entity that consists of the network systems, software, and policies that manage email delivery for a recipient (p. 301).
recipient	Amazon SES: The person or entity receiving an email message. For example, a person named in the "To" field of a message.
reducer	An executable in the MapReduce process that uses the intermediate results from the mapper and processes them into the final output.
reference	A means of inserting a property from one AWS resource into another. For example, you could insert an Amazon EC2 security group property into an Amazon RDS resource.
region	A named set of AWS resources in the same geographical area. A region comprises at least two Availability Zones.
reply path	The email address to which an email reply is sent. This is different from the return path (p. 302).
reputation	1. An Amazon SES metric, based on factors that might include bounces, complaints, and other metrics, regarding whether or not a customer is sending high-quality emails.
	2. A measure of confidence, as judged by an Internet Service Provider (p. 293) or other entity that an IP address that they are receiving emails from is not the source of spam (p. 305).
requester	The person (or application) that sends a request to AWS to perform a specific action. When AWS receives a request, it first evaluates the requester's permissions to determine whether the requester is allowed to perform the request action (if applicable, for the requested resource).
Requester Pays	An Amazon S3 feature that allows a bucket owner (p. 284) to specify that anyone who requests access to objects in a particular bucket must pay the data transfer and request costs.
reservation	A collection of EC2 instances started as part of the same launch request. Not to be confused with a Reserved Instance (p. 301).
Reserved Instance	A pricing option that lets you make a low, one-time payment for each instance to reserve and receive a significant discount on the hourly usage charge for that instance.

Reserved Instance Marketplace	Matches sellers who have reserved capacity that they no longer need with buyers who are looking to purchase additional capacity. Reserved Instances that you purchase from third-party sellers will have less than a full standard term remaining and can be sold at different upfront prices. The usage or reoccurring fees will remain the same as the fees set when the Reserved Instances were originally purchased. Full standard terms for Reserved Instances available from AWS run for one year or three years.
resource	1. An entity that users can work with in AWS, such as an EC2 instance, a DynamoDB table, an IAM user, an AWS OpsWorks stack, and so on. 2. Tools, code, and documents that AWS provides to support users. 3. A required element of an AWS CloudFormation stack (p. 305). Each stack contains at least one resource, such as an Auto Scaling LaunchConfiguration. All resources in a stack must be created successfully for the stack to be created.
resource property	A value required when including an AWS resource in an AWS CloudFormation stack (p. 305). Each resource may have one or more properties associated with it. For example, an `AWS::EC2::Instance` resource may have a `UserData` property. In an AWS CloudFormation template, resources must declare a properties section, even if the resource has no properties.
resource record	Also called *resource record set*. Standard DNS terminology. See Also http://en.wikipedia.org/wiki/Domain_Name_System.
REST	A type of HTTP-based request interface that generally uses only the GET or POST HTTP method and a query string with parameters. Sometimes known as Query. In some implementations of a REST interface, other HTTP verbs besides GET and POST are used.
REST-Query	Also known as Query or HTTP Query. This is a type of HTTP request that generally uses only the GET or POST HTTP method and a query string with parameters. Compare this with REST, which is a type of HTTP request that uses any HTTP method (GET, DELETE, POST, etc.), a resource, HTTP headers, and possibly a query string with parameters.
return enabled	An Amazon CloudSearch index field option that enables the field's values to be returned in the search results.
return path	The email address to which bounced emails are returned. The return path is specified in the header of the original email. This is different from the reply path (p. 301).
rollback	A return to a previous state that follows the failure to create an object, such as AWS CloudFormation stack (p. 305). All resources associated with the failure are deleted during the rollback. For AWS CloudFormation, you can override this behavior using the `--disable-rollback` option on the command line.
root device volume	Contains the image used to boot the instance. If you launched the instance from an AMI backed by instance store, this is an instance store volume created from a template stored in Amazon S3. If you launched the instance from an AMI backed by Amazon EBS, this is an Amazon EBS volume created from an Amazon EBS snapshot.
route table	A set of routing rules that controls the traffic leaving any subnet that is associated with the route table. You can associate multiple subnets with a single route table, but a subnet can be associated with only one route table at a time.

S

Numbers and Symbols (p. 280) | A (p. 280) | B (p. 284) | C (p. 285) | D (p. 287) | E (p. 289) | F (p. 290) | G (p. 291) | H (p. 291) | I (p. 292) | J (p. 294) | K (p. 294) | L (p. 295) | M (p. 295) | N (p. 297) | O (p. 298) | P (p. 298) | Q (p. 300) | R (p. 300) | S (p. 303) | T (p. 307) | U (p. 308) | V (p. 308) | W (p. 309) | X, Y, Z (p. 309)

sampling period	A defined duration of time, such as one minute, over which CloudWatch computes a statistic (p. 305).
sandbox	A testing location where you can test the functionality of your application without affecting production, incurring charges, or purchasing products. Amazon SES: An Amazon SES environment that is designed for developers to test and evaluate the service. In the sandbox, you have full access to the Amazon SES API, but you can only send messages to verified email addresses and the mailbox simulator. To get out of the sandbox, you need to apply for production access. Accounts in the sandbox also have lower sending limits (p. 304) than production accounts.
scaling activity	A process that changes the size, configuration, or makeup of an Auto Scaling group (p. 283) by launching or terminating instances. For more information, see Auto Scaling Concepts in the Auto Scaling Developer Guide.
search API	The Amazon CloudSearch API that you use to submit search requests to a search domain.
search domain	Encapsulates your searchable data and the search instances that handle your search requests. You typically set up a separate Amazon CloudSearch domain for each different collection of data that you want to search.
search domain configuration	An Amazon CloudSearch domain's indexing options, analysis schemes, expressions, suggesters, access policies, and scaling and availability options.
search enabled	An Amazon CloudSearch index field option that enables the field data to be searched.
search endpoint	The URL that you connect to when sending search requests to a search domain. Each Amazon CloudSearch domain has a unique search endpoint that remains the same for the life of the domain.
search index	A representation of your searchable data that facilitates fast and accurate data retrieval.
search instance	A compute resource that indexes your data and processes search requests. An Amazon CloudSearch domain has one or more search instances, each with a finite amount of RAM and CPU resources. As your data volume grows, more search instances or larger search instances are deployed to contain your indexed data. When necessary, your index is automatically partitioned across multiple search instances. As your request volume or complexity increases, each search partition is automatically replicated to provide additional processing capacity.
search request	A request that is sent to an Amazon CloudSearch domain's search endpoint to retrieve documents from the index that match particular search criteria.
search result	A document that matches a search request. Also referred to as a *search hit*.
secret access key	A key that is used in conjunction with the access key ID (p. 280) to cryptographically sign programmatic AWS requests. Signing a request identifies the sender and

	prevents the request from being altered. You can generate secret access keys for your AWS account, individual IAM users, and temporary sessions.
security group	A named set of allowed inbound network connections for an instance. (Security groups in Amazon VPC also include support for outbound connections.) Each security group consists of a list of protocols, ports, and IP address ranges. A security group can apply to multiple instances, and multiple groups can regulate a single instance.
sender	The person or entity sending an email message.
Sender ID	A Microsoft-controlled version of SPF. An email authentication and anti-spoofing system. For more information about Sender ID, go to http://wikipedia.org/wiki/Sender_ID.
sending limits	The sending quota (p. 304) and maximum send rate (p. 296) that are associated with every Amazon SES account.
sending quota	The maximum number of emails that you can send using Amazon SES in a 24-hour period.
service endpoint	See endpoint.
service health dashboard	A web page showing up-to-the-minute information about AWS service availability. The dashboard is located at http://status.amazonaws.cn/.
SHA	Secure Hash Algorithm. SHA1 is an earlier version of the algorithm, which AWS has deprecated in favor of SHA256.
shared AMI	An Amazon Machine Image (p. 282) that a developer builds and makes available for others to use.
shutdown action	A predefined bootstrap action that launches a script that executes a series of commands in parallel before terminating the job flow.
signature	Refers to a *digital signature*, which is a mathematical way to confirm the authenticity of a digital message. AWS uses signatures to authenticate the requests you send to our web services. For more information, to http://www.amazonaws.cn/security.
SIGNATURE file	A file you copy to the root directory of your storage device. The file contains a job ID, manifest file, and a signature.
Simple Mail Transfer Protocol	See SMTP.
Single-AZ DB Instance	A standard (non-Multi-AZ) DB instance (p. 287) that is deployed in one Availability Zone (p. 283), without a standby replica in another Availability Zone. See Also Multi-AZ deployment.
single-valued attribute	An attribute with one value.
sloppy phrase search	A search for a phrase that specifies how close the terms must be to one another to be considered a match.
SMTP	Simple Mail Transfer Protocol. The standard that is used to exchange email messages between internet hosts for the purpose of routing and delivery.
snapshot	Amazon Elastic Block Store (p. 281) creates *snapshots* or backups of your volumes and stores them in Amazon S3. You can use these snapshots as the starting point for new Amazon EBS volumes or to protect your data for long-term durability.

soft bounce	A temporary email delivery failure such as "mailbox full."
software VPN	A software appliance-based VPN connection over the Internet.
sort enabled	An Amazon CloudSearch index field option that enables a field to be used to sort the search results.
source/destination checking	A security measure to verify that an EC2 instance is the origin of all traffic that it sends and the ultimate destination of all traffic that it receives, that is, that the instance is not relaying traffic. Source/destination checking is enabled by default. For instances that function as gateways, such as VPC NAT instances, source/destination checking must be disabled.
spam	Unsolicited bulk email.
spamtrap	An email address that is set up by an anti-spam (p. 305) entity, not for correspondence, but to monitor unsolicited email. This is also called a *honeypot*.
SPF	Sender Policy Framework. A standard for authenticating email. See Also http://www.openspf.org.
Spot Instance	A type of EC2 instance (p. 289) that you can bid on to take advantage of unused Amazon EC2 capacity.
Spot Price	The price for a Spot Instance (p. 305) at any given time. If your maximum price exceeds the current price and your restrictions are met, Amazon EC2 launches instances on your behalf.
stack	AWS CloudFormation: A collection of AWS resources you create and delete as a single unit. AWS OpsWorks: A set of instances you manage collectively, typically because they have a common purpose such as serving PHP applications. A stack serves as a container and handles tasks that apply to the group of instances as a whole, such as managing applications and cookbooks.
station	A place at an AWS facility where we transfer your AWS Import/Export data on to, or off of, your storage device.
statistic	One of five functions of the values submitted for a given sampling period (p. 303). These functions are "Maximum", "Minimum," "Sum," "Average," and "SampleCount."
stem	The common root or substring shared by a set of related words.
stemming	The process of mapping related words to a common stem. This enables matching on variants of a word. For example, a search for "horse" could return matches for horses, horseback, and horsing, as well as horse. Amazon CloudSearch supports both dictionary based and algorithmic stemming.
step	A single function applied to the data in a job flow (p. 294). The sum of all steps comprises a job flow.
step type	The type of work done in a step. There are a limited number of step types, such as moving data from Amazon S3 to Amazon EC2 or from Amazon EC2 to Amazon S3.
sticky session	A feature of the load balancer that binds a user's session to a specific application instance so that all requests coming from the user during the session are sent to

	the same application instance. By contrast, a load balancer defaults to route each request independently to the application instance with the smallest load.
stopping	The process of filtering stop words from an index or search request.
stopword	A word that is not indexed and is automatically filtered out of search requests because it is either insignificant or so common that including it would result in too many matches to be useful. Stop words are language-specific.
streaming	Amazon EMR: A utility that comes with Hadoop that enables you to develop MapReduce executables in languages other than Java. CloudFront: The ability to use a media file in real time—as it is transmitted in a steady stream from a server.
streaming distribution	A special kind of distribution (p. 288) that serves streamed media files using a Real Time Messaging Protocol (RTMP) connection.
string-to-sign	Before you calculate an HMAC signature, you first assemble the required components in a canonical order. The pre-encrypted string is the string-to-sign.
structured query	Search criteria specified using the Amazon CloudSearch structured query language. You use the structured query language to construct compound queries that use advanced search options and combine multiple search criteria using Boolean operators.
subnet	A segment of the IP address range of a VPC (p. 309) that EC2 instances can be attached to. You can create subnets to group instances according to security and operational needs.
Subscription button	An HTML-coded button that enables an easy way to charge customers a recurring fee.
suggester	Specifies an Amazon CloudSearch index field you want to use to get autocomplete suggestions and options that can enable fuzzy matches and control how suggestions are sorted.
suggestions	Documents that contain a match for the partial search string in the field designated by the suggester. Amazon CloudSearch suggestions include the document IDs and field values for each matching document. To be a match, the string must match the contents of the field starting from the beginning of the field.
supported AMI	An Amazon Machine Image (p. 282) similar to a paid AMI (p. 298), except that the owner charges for additional software or a service that customers use with their own AMIs.
synchronous bounce	A type of bounce (p. 284) that occurs while the email servers of the sender (p. 304) and receiver (p. 301) are actively communicating.
synonym	A word that is the same or nearly the same as an indexed word and that should produce the same results when specified in a search request. For example, a search for "Rocky Four" or "Rocky 4" should return the fourth *Rocky* movie. This can be done by designating that `four` and `4` are synonyms for `IV`. Synonyms are language-specific.

T

Numbers and Symbols (p. 280) | A (p. 280) | B (p. 284) | C (p. 285) | D (p. 287) | E (p. 289) | F (p. 290) | G (p. 291) | H (p. 291) | I (p. 292) | J (p. 294) | K (p. 294) | L (p. 295) | M (p. 295) | N (p. 297) | O (p. 298) | P (p. 298) | Q (p. 300) | R (p. 300) | S (p. 303) | T (p. 307) | U (p. 308) | V (p. 308) | W (p. 309) | X, Y, Z (p. 309)

tag	Metadata (consisting of up to 10 key/value pairs) that you can define and assign to Amazon EC2 resources.
tagging	Also called *labeling*. A way to format return path (p. 302) email addresses so that you can specify a different return path for each recipient of a message. Tagging enables you to support VERP (p. 308). For example, if Andrew manages a mailing list, he can use the return paths andrew+recipient1@example.net and andrew+recipient2@example.net so that he can determine which email bounced.
task node	An EC2 instance (p. 289) that runs Hadoop map and reduce tasks, but does not store data. Task nodes are managed by the master node (p. 296), which assigns Hadoop tasks to nodes and monitors their status. While a job flow is running you can increase and decrease the number of task nodes. Because they don't store data and can be added and removed from a job flow, you can use task nodes to manage the EC2 instance capacity your job flow uses, increasing capacity to handle peak loads and decreasing it later. Task nodes only run a TaskTracker Hadoop daemon.
tebibyte	A contraction of tera binary byte, a tebibyte is 2^40 or 1,099,511,627,776 bytes. A terabyte (TB) is 10^12 or 1,000,000,000,000 bytes. 1,024 TiB is a pebibyte (p. 298).
template format version	The version of an AWS CloudFormation template design that determines the available features. If you omit the `AWSTemplateFormatVersion` section from your template, AWS CloudFormation assumes the most recent format version.
template validation	The process of confirming the use of JSON (p. 294) code in an AWS CloudFormation template. You can validate any AWS CloudFormation template using the `cfn-validate-template` command.
throttling	The means by which Amazon SES rejects your attempts to send email because you have exceeded your sending limits (p. 304).
time series data	Data provided as part of a metric. The time value is assumed to be when the value occurred. A metric is the fundamental concept for CloudWatch and represents a time-ordered set of data points. You publish metric data points into CloudWatch and later retrieve statistics about those data points as a time-series ordered data set.
time stamp	A date/time string in ISO 8601 format.
TLS	See Transport Layer Security.
tokenization	The process of splitting a stream of text into separate tokens on detectable boundaries such as whitespace and hyphens.
topic	A communication channel to send messages and subscribe to notifications. It provides an access point for publishers and subscribers to communicate with each other.
Transport Layer Security	A cryptographic protocol that provides security for communication over the Internet. Its predecessor is Secure Sockets Layer (SSL).

trusted signers	AWS accounts that the CloudFront distribution owner has given permission to create signed URLs for a distribution's content.
tuning	Selecting the number and type of AMIs (p. 282) to run a Hadoop job flow most efficiently.
tunnel	A route for transmission of private network traffic that uses the Internet to connect nodes in the private network. The tunnel uses encryption and secure protocols such as PPTP to prevent the traffic from being intercepted as it passes through public routing nodes.

U

Numbers and Symbols (p. 280) | A (p. 280) | B (p. 284) | C (p. 285) | D (p. 287) | E (p. 289) | F (p. 290) | G (p. 291) | H (p. 291) | I (p. 292) | J (p. 294) | K (p. 294) | L (p. 295) | M (p. 295) | N (p. 297) | O (p. 298) | P (p. 298) | Q (p. 300) | R (p. 300) | S (p. 303) | T (p. 307) | U (p. 308) | V (p. 308) | W (p. 309) | X, Y, Z (p. 309)

unbounded	The number of potential occurrences is not limited by a set number. This value is often used when defining a data type that is a list (for example, `maxOccurs="unbounded"`), in Web Services Description Language (p. 309).
unit	Standard measurement for the values submitted to CloudWatch as metric data. Units include Seconds, Percent, Bytes, Bits, Count, Bytes/Second, Bits/Second, Count/Second, and None.
unlink from VPC	The process of unlinking (or detaching) an EC2-Classic instance from a ClassicLink-enabled VPC. See Also ClassicLink, link to VPC.
usage report	An AWS report giving details of your usage of a particular AWS service. You can generate and download usage reports from http://www.amazonaws.cn/usage-reports/.
user	A person or application under an account (p. 280) that needs to make API calls to AWS products. Each user has a unique name within the AWS account, and a set of security credentials not shared with other users. These credentials are separate from the AWS account's security credentials. Each user is associated with one and only one AWS account.

V

Numbers and Symbols (p. 280) | A (p. 280) | B (p. 284) | C (p. 285) | D (p. 287) | E (p. 289) | F (p. 290) | G (p. 291) | H (p. 291) | I (p. 292) | J (p. 294) | K (p. 294) | L (p. 295) | M (p. 295) | N (p. 297) | O (p. 298) | P (p. 298) | Q (p. 300) | R (p. 300) | S (p. 303) | T (p. 307) | U (p. 308) | V (p. 308) | W (p. 309) | X, Y, Z (p. 309)

validation	See template validation.
value	Instances of attributes (p. 283) for an item, such as cells in a spreadsheet. An attribute might have multiple values.
Variable Envelope Return Path	See VERP.
verification	The process of confirming that you own an email address or a domain so that you can send emails from or to it.
VERP	Variable Envelope Return Path. A way in which email sending applications can match bounced emails with the undeliverable address that caused the bounce

	by using a different return path (p. 302) for each recipient. VERP is typically used for mailing lists. With VERP, the recipient's email address is embedded in the address of the return path, which is where bounced emails are returned. This makes it possible to automate the processing of bounced emails without having to open the bounce messages, which may vary in content.
versioning	Every object in Amazon S3 has a key and a version ID. Objects with the same key, but different version IDs can be stored in the same bucket. Versioning is enabled at the bucket layer using PUT Bucket versioning.
virtualization	Allows multiple guest virtual machines (VM) to run on a host operating system. Guest VMs can run on one or more levels above the host hardware, depending on the type of virtualization. See Also PV virtualization, HVM virtualization.
virtual private cloud	See VPC.
virtual private gateway	See VPG.
visibility timeout	The period of time that a message is invisible to the rest of your application after an application component gets it from the queue. During the visibility timeout, the component that received the message usually processes it, and then deletes it from the queue. This prevents multiple components from processing the same message.
VPC	Virtual private cloud. An elastic network populated by infrastructure, platform, and application services that share common security and interconnection.
VPG	Virtual private gateway. The Amazon side of a VPN connection that maintains connectivity. The internal interfaces of the virtual private gateway connect to your VPC via the VPN attachment and the external interfaces connect to the VPN connection, which leads to the customer gateway.
VPN CloudHub	See AWS VPN CloudHub.
VPN connection	Although VPN connection is a general term, we specifically mean the IPsec connection between a VPC (p. 309) and some other network, such as a corporate data center, home network, or co-location facility.

W

Numbers and Symbols (p. 280) | A (p. 280) | B (p. 284) | C (p. 285) | D (p. 287) | E (p. 289) | F (p. 290) | G (p. 291) | H (p. 291) | I (p. 292) | J (p. 294) | K (p. 294) | L (p. 295) | M (p. 295) | N (p. 297) | O (p. 298) | P (p. 298) | Q (p. 300) | R (p. 300) | S (p. 303) | T (p. 307) | U (p. 308) | V (p. 308) | W (p. 309) | X, Y, Z (p. 309)

Web Services Description Language	A language used to describe the actions that a web service can perform, along with the syntax of action requests and responses. Your SOAP or other toolkit interprets a WSDL file to provide your application access to the actions provided by the web service. For most toolkits, your application calls a service action using routines and classes provided or generated by the toolkit.

X, Y, Z

yobibyte	A contraction of yotta binary byte, a yobibyte is 2^80 or 1,208,925,819,614,629,174,706,176 bytes. A yottabyte (YB) is 10^24 or 1,000,000,000,000,000,000,000,000 bytes.

zebibyte	A contraction of zetta binary byte, a zebibyte is 2^70 or 1,180,591,620,717,411,303,424 bytes. A zettabyte (ZB) is 10^21 or 1,000,000,000,000,000,000,000 bytes. 1,024 ZiB is a yobibyte (p. 309).

Document History

The following table describes the important changes to the documentation since the last release of *AWS Identity and Access Management*.

- **API version:** 2010-05-08
- **Latest documentation update:** February 11, 2015

Change	Description	Release Date
Managed Policies	Added new documentation relating to managed policies (p. 179). You can now use the IAM section of the AWS Management Console to attach AWS managed policies to your IAM users, groups, and roles. You can also create customer managed policies, which are standalone policies that you can attach to multiple IAM users, groups, and roles. Standalone policies means that each policy has its own Amazon Resource Name (ARN).	February 11, 2015
Switch Role in the Console	Added new documentation relating to Switching to a Role in the AWS Management Console (p. 137). You can now switch to a role in an account from within the AWS Management Console. This enables you to more easily delegate console tasks.	January 8th, 2015
Create OpenID Connect (OIDC) identity providers and use them with Amazon Cognito	Added new documentation relating to Using OpenID Connect Identity Providers (p. 109). You can now use the IAM section of the AWS Management Console to add any OIDC-compatible identity provider, and then you can use the identity provider in your mobile apps that use Amazon Cognito.	October 23, 2014
View a user's last sign-in time	Updated the documentation relating to Getting Credential Reports for Your AWS Account (p. 93) to add information about the `password_last_used` field.	October 16, 2014
Sign-in Events now in AWS CloudTrail Log Files	Updated the documentation relating to Logging IAM Events with AWS CloudTrail (p. 259) to add sign-in events to the IAM information that is logged by AWS CloudTrail.	July 24, 2014

Change	Description	Release Date
Credential Lifecycle Management	Updated the documentation relating to Setting an Account Password Policy for IAM Users (p. 59) to add new password management options. Added new documentation relating to Getting Credential Reports for Your AWS Account (p. 93).	July 16, 2014
Updates to web identity federation documentation for Amazon Cognito	Updated the documentation relating to web identity federation (for example, pages about how to create a role and about IAM policy condition keys) to include information about Amazon Cognito. For more information about Amazon Cognito, see the *AWS SDK for Android Developer Guide* and the *AWS SDK for iOS Developer Guide*.	July 10, 2014
MFA Support for Cross-Account API Actions	This release lets you enforce multi-factor authentication (MFA) when providing programmatic access across AWS accounts. You can create policies that require an IAM user to be authenticated using an MFA device before assuming an IAM role. For more information, see Configuring MFA-Protected API Access (p. 80).	February 27, 2014
Support for SAML-Based Federation, Updated Documentation	This release adds support for SAML-based federation. In IAM you can create a SAML provider, which is an entity that describes an identity provider (IdP) that supports SAML 2.0 (Security Assertion Markup Language 2.0). You create a SAML provider in IAM if you want to establish trust between AWS and an identity provider. For more information, see Using SAML Providers (p. 105). The documentation was also reorganized to make it easier to find information. In particular, the sections pertaining to IAM users and groups and to working with credentials were updated.	November 7, 2013
Policy Variables, Updated Documentation	This release adds support for including variables in policies; this makes it easier to create policies that apply to the current request context, such as to the current user. For details, see IAM Policy Variables Overview (p. 232). The documentation was also reorganized to make it easier to find information (for example, the table of contents was restructured), and examples were added to Example Policies for Administering AWS Resources (p. 203).	April 3, 2013
Best Practices	This release includes a topic on IAM best practices. For details, see IAM Best Practices (p. 26).	January 10, 2013
Cross-account API access	This release adds support for cross-account API access with IAM roles. With IAM roles, you can delegate access to resources in your AWS account so that IAM users from another AWS account can access your resources. For details, see IAM Roles (Delegation and Federation) (p. 116).	November 19, 2012
MFA-Protected API Access	This release introduces MFA-protected API access, a feature that enables you to add an extra layer of security over AWS APIs using AWS Multi-Factor Authentication (MFA), see Configuring MFA-Protected API Access (p. 80).	July 8, 2012
Business Use Cases	This section has been rewritten and updated. For more information, see Business Use Cases (p. 30).	June 22, 2012

Change	Description	Release Date
IAM Roles for Amazon EC2 Instances	This release introduces IAM roles for Amazon EC2 instances. Use roles to enable applications running on your Amazon EC2 instances to securely access your AWS resources. For more information about IAM roles for EC2 instances, see IAM Roles (Delegation and Federation) (p. 116).	June 7, 2012
AWS Storage Gateway	This release introduces AWS Storage Gateway integration with IAM. For more information about using IAM with AWS Storage Gateway, go to Access Control Using AWS Identity and Access Management (IAM) in the *AWS Storage Gateway User Guide*. For a general description of AWS Storage Gateway, go to AWS Storage Gateway.	May 14, 2012
Updated Documentation	The IAM Getting Started Guide was merged into Using IAM, and Using IAM was reorganized to enhance usability. The Getting Started is now available at Getting Started (p. 19).	May 2, 2012
Signature Version 4	With this release of IAM, you can use Signature Version 4 to sign your IAM API requests. For more information about Signature Version 4, go to Signature Version 4 Signing Process in the *AWS General Reference*.	March 15, 2012
User Password Management	With this release of IAM, you can enable your IAM users to change their password. For more information, see Credentials (Passwords, Access Keys, and MFA devices) (p. 56).	March 8, 2012
Account Password Policy	IAM now includes an account-wide password policy you can use to ensure your IAM users create strong passwords. For more information, see Setting an Account Password Policy for IAM Users (p. 59).	March 8, 2012
IAM User Access to Your AWS Account Billing Information	With this release of IAM, you can enable your IAM users to access your AWS account billing and usage information. For more information, see Controlling Access to Your Billing Information.	March 8, 2012
Amazon Simple Workflow Service (SWF)	This release introduces Amazon Simple Workflow Service (SWF) integration with IAM. For more information about using IAM with Amazon Simple Workflow Service, go to Managing Access to Your Amazon SWF Workflows in the *Amazon Simple Workflow Service Developer Guide* . For a general description of Amazon Simple Workflow Service, go to Amazon Simple Workflow Service.	February 22, 2012
Single Sign-on Access to the AWS Management Console for Federated Users	With this release, you can give your federated users single sign-on access to the AWS Management Console through your identity and authorization system, without requiring users to sign in to Amazon Web Services (AWS). For more information, go to Giving Federated Users Direct Access to the AWS Management Console in *Using Temporary Security Credentials*.	January 19, 2012
New Documentation: Using Temporary Security Credentials	The documentation that describes creating temporary security credentials for federated users and mobile applications has been moved to a new, expanded stand-alone guide named *Using Temporary Security Credentials*.	January 19, 2012

Change	Description	Release Date
Amazon DynamoDB	This release introduces Amazon DynamoDB integration with IAM. For more information about using IAM with Amazon DynamoDB, go to Controlling Access to Amazon DynamoDB Resources in the *Amazon DynamoDB Developer Guide*. For a general description of Amazon DynamoDB, go to Amazon DynamoDB.	January 18, 2012
AWS Elastic Beanstalk	This release introduces AWS Elastic Beanstalk integration with IAM. For more information about using IAM with AWS Elastic Beanstalk, go to Using AWS Elastic Beanstalk with AWS Identity and Access Management (IAM) in the *AWS Elastic Beanstalk Developer Guide*. For a general description of AWS Elastic Beanstalk, go to AWS Elastic Beanstalk. For IAM use cases, see Business Use Cases (p. 30).	November 21, 2011
AWS Virtual MFA	With this release, you can use IAM to configure and enable a virtual MFA device. A virtual MFA device uses a software application that can generate six-digit authentication codes that are compatible with the time-based one-time password (TOTP) standard, as described in RFC 6238. The software application can run on any mobile hardware device, including a smartphone. For more information about virtual MFA and about using IAM to configure and enable a virtual MFA device, see Using a Virtual MFA Device with AWS (p. 71).	November 2, 2011
Policy Generator Integration with the AWS Identity and Access Management Console	This release introduces the integration of the policy generator with the AWS Identity and Access Management (IAM) console. Integrating the policy generator with the IAM console makes it even easier to set permissions for your IAM users and groups. To use the policy generator in the console, select **Policy Generator** in the user or group permissions dialogs. For more information about the AWS access policy syntax, see Overview of IAM Policies (p. 174) in *Using IAM*. If you want to use the policy generator online to create policies for AWS products without accessing the console, go to the AWS Policy Generator.	October 6, 2011
Amazon ElastiCache	This release introduces Amazon ElastiCache integration with IAM. For more information about using IAM with Amazon ElastiCache, go to Controlling User Access to Your AWS Account in the *Amazon ElastiCache User Guide*. For a general description of Amazon ElastiCache, go to Amazon ElastiCache. For IAM use cases, see Business Use Cases (p. 30).	August 23, 2011
Temporary Security Credentials	This release of IAM introduces temporary security credentials that you can use to grant temporary access to non-AWS users (federated users), to IAM users who need temporary access to your AWS resources, and to your mobile and browser-based applications that need to access your AWS resources securely. For more information, go to Using Temporary Security Credentials.	August 3, 2011
Cross-Account Access for IAM Users	This release of IAM introduces cross-account access for IAM users. For more information, see IAM Roles (Delegation and Federation) (p. 116).	June 6, 2011
The AWS Management Console IAM Tab	This release of IAM introduces AWS Management Console support. The **IAM** tab of the console is a graphical user interface (GUI) that enables you to do almost everything you can do with the IAM APIs. For more information, see Accessing IAM (p. 7).	May 3, 2011

Change	Description	Release Date
Amazon CloudFront	This release of IAM includes integration with Amazon CloudFront. For more information, go to Controlling User Access to Your AWS Account in the *Amazon CloudFront Developer Guide*.	March 10, 2011
AWS Cloud-Formation	This release introduces AWS CloudFormation integration with IAM. For more information, go to Controlling User Access With AWS Identity and Access Management in the *Amazon CloudFront Developer Guide*.	February 24, 2011
Amazon Elastic MapReduce	This release introduces Amazon Elastic MapReduce integration with IAM. For more information, go to *Amazon Elastic MapReduce* in Business Use Cases (p. 30) in *Using IAM*.	February 22, 2011
IAM-Enabled User Access to the AWS Management Console and AWS Developer Forums	IAM now provides an IAM-enabled sign-in page for the AWS Management Console. You provide your users with a login profile and with appropriate permissions so they can access your available AWS resources through the AWS Management Console. For information about accessing the AWS Management Console through IAM, see IAM and the AWS Management Console (p. 34). For information about the AWS Management Console, see AWS Management Console.	February 14, 2011
Amazon Simple Email Service	This release introduces Amazon Simple Email Service (Amazon SES) integration with IAM. For more information, see Controlling User Access with IAM.	January 24, 2011
AWS IAM Integration with Amazon Route 53	Amazon Route 53 DNS service is now integrated with IAM. For information about using Amazon Route 53 with IAM, see AWS Services That Support IAM (p. 265). For more information about Amazon Route 53, go to Amazon Route 53 on the AWS website.	December 5, 2010
AWS IAM Integration with Amazon CloudWatch	Amazon CloudWatch is now integrated with IAM. For information about using Amazon CloudWatch with IAM, see Controlling User Access to Your AWS Account. For more information about Amazon CloudWatch, see Amazon CloudWatch on the AWS website.	November 29, 2010
Server Certificate Support	IAM now provides server certificate APIs for use with Elastic Load Balancing server certificates. For information about using IAM to manage server certificates, see Managing Server Certificates (p. 162).	October 14, 2010
Initial Release	This is the first release of *Using IAM*.	September 2, 2010