AWS Identity and Access Management

Using IAM

API Version 2010-05-08

Amazon Web Services

AWS Identity and Access Management: Using IAM

Amazon Web Services

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 SimpleDB, 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. Getting Started with AWS Identity and Access Management

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 products are integrated with IAM, see AWS Services that Support IAM (p. 200). As AWS products integrate with IAM, they will be added to the list on that page.

> **Important**
> If you're an existing AWS account owner, you should know that the APIs for these products do not change because of their integration with IAM. Products that integrate with IAM have no new API actions related to access control.

If the users in your AWS account make API calls to any AWS products other than those that are integrated with IAM, they'll get an error. However, the AWS account itself (the umbrella entity above all the users) can call any AWS product the AWS account is signed up for.

Regarding IAM and Amazon DevPay:

- Users in your AWS account can't create or sign up for products that use Amazon DevPay; only the AWS account can
- If your AWS account purchases an Amazon EC2 paid Amazon Machine Image (AMI), the AWS account's users can launch and use instances of the AMI (assuming they have an IAM policy giving them permission to use the necessary Amazon EC2 actions)
- If your AWS account purchases an Amazon S3 product that uses Amazon DevPay, the AWS account's users can't use the product; only the AWS account can

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. 3)).
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.

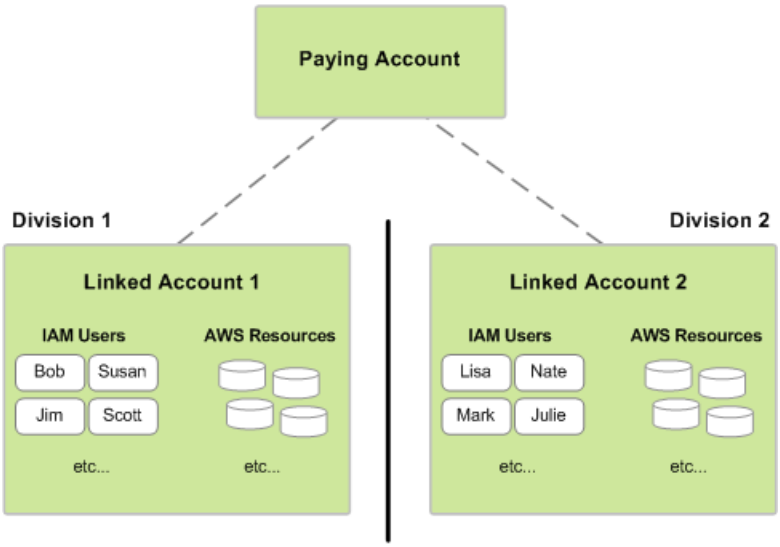
No Change to Basic AWS Account Functions

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.

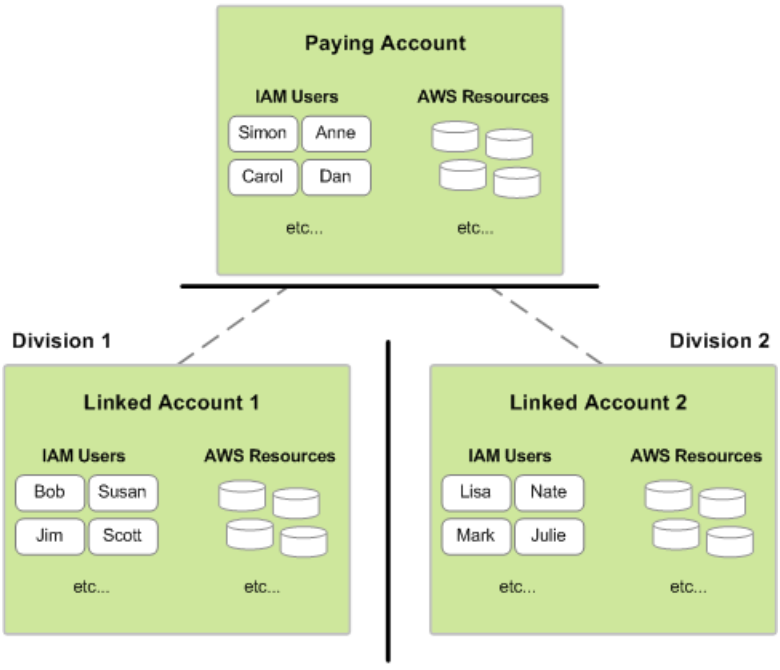
IAM and Consolidated Billing

AWS offers a billing feature called *Consolidated Billing* that is complementary to IAM. Consolidated Billing lets you receive a single bill for multiple AWS accounts. (For more information, see Consolidated Billing in *About AWS Account Billing*.) In contrast, IAM lets you get a single bill across all the *users* in a single AWS account.

Your organization could use both 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 two 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 controls 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. 76).

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.

User

A *user* is an individual, system, or application that interacts with AWS programmatically. 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. Users don't need to sign up for AWS products (such as Amazon EC2), nor do they need to have a payment method on file with AWS. The AWS account is the entity that is subscribed to AWS products and pays for the users' activity with AWS.

Users are the primary actors who interact with AWS. They can use the AWS Management Console, an API, or a command line interface (if the AWS product provides one). Authentication happens at the user level by means of the user's login and password (for the AWS Management Console), or the user's security credentials (for the API or command line interface). Users access the AWS Management Console via a link to your AWS account sign-in page that you provide to them. (For more information on your AWS account sign-in page, see The AWS Management Console Sign-in Page (p. 185).)

By default, newly created users have no password, no security credentials, and no *permissions* to do anything. The AWS account owner (or any user with the authority, such as an administrator) must provide each user with what they need.

There's a limit to the number of users you can have. For more information, see Limitations on IAM Entities (p. 16).

For more information about creating users, see Adding a New User to Your AWS Account (p. 42).

Group

A *group* is a collection of users. Groups don't directly interact with AWS; only users do. The main reason to create groups is to collectively assign permissions to the users so they can do their jobs. For example, you could have a group called *Admins* and give that group the types of permissions admins typically need.

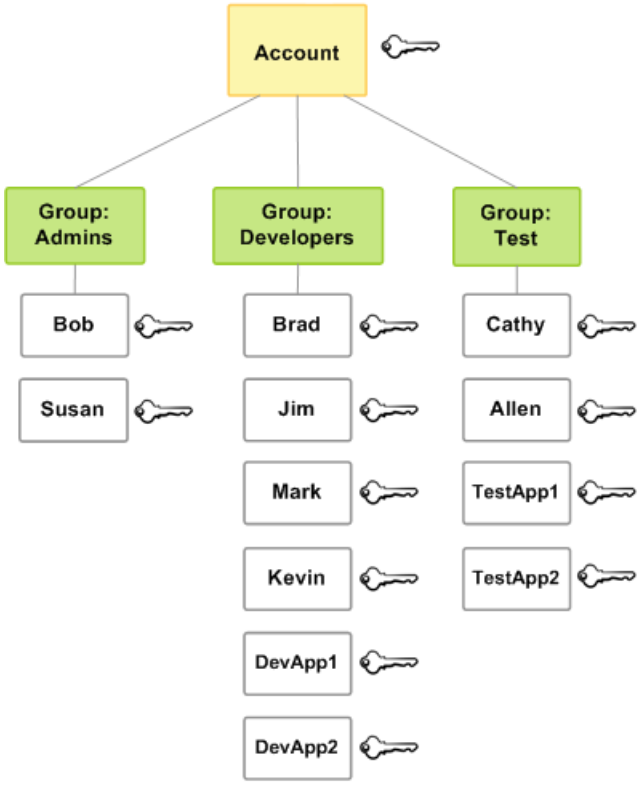
Following are some important characteristics of groups:

- A group can contain many users, and a user can belong to multiple groups.
- Groups can't be nested; they can contain only users.
- There's no default group that automatically includes all users in the AWS account. If you'd like to have a group like that, you need to create it yourself 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. 16).

For more information about creating and managing groups, see Users and Groups (p. 42).

Example

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 create and manage the users as the company grows. The Admins group establishes a Development group and a Test group. Each of these two 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.



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 Roles (p. 151).

Security Credentials

Any person or application making API calls to AWS needs *security credentials*. The entity making the call can be the AWS account, a user, a role. AWS uses these credentials to identify who is making the call and whether to allow the requested access.

The basic credentials consist of a *Secret Access Key*, and a corresponding *Access Key ID*. The requester signs each request with the Secret Access Key and includes the Access Key ID in the request. How this works for a given AWS product is covered in the product's technical documentation.

AWS Account Credentials

As mentioned elsewhere, when you create an AWS account, AWS gives the AWS account its own Secret Access Key and Access Key ID by default. The AWS account can make API calls to AWS with them. We expect that you won't use those credentials on a regular basis, but will use them only to initially set up an administrators group for your organization. We recommend that all further API interaction between your AWS account and your AWS resources be at the user level (for example, using users' security credentials).

To get your AWS account's secret access key and access key ID, go to https://console.aws.amazon.com/iam/home?#security_credential and sign in with your AWS account's login and password.

Note
To help control who has access to the AWS account's Secret Access Key, we recommend that you use Multi-Factor Authentication (MFA) with your AWS account's 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. 76).

User Credentials

By default, a user has no security credentials. You create security credentials for your users as needed. The type of credentials a user needs depends on how the user will access AWS.

Secret Access Keys and Access Key IDs

To make API calls or to work with the command line interface, the user needs a Secret Access Key and Access Key ID. The IAM API and command line interface provide actions that create these for a user. You can give your users permission to create and manage their own credentials if you like, or you can have an administrators group in your organization handle this. For more information about creating keys for a user, see Adding a New User to Your AWS Account (p. 42).

X.509 Certificates

Another type of credential a user might have is an *X.509 certificate* (referred to here as a *signing certificate*) and corresponding *certificate ID*. Some AWS products use this instead of a Secret Access Key for access to certain interfaces. For example, Amazon EC2 uses a Secret Access Key for access to its Query interface, but it uses a signing certificate for access to its SOAP interface and command line tool interface.

Although you can use IAM to create an access key, you can't use IAM to create a signing certificate. However, you can use free third-party tools such as OpenSSL to create the certificate. (For information about OpenSSL, go to http://www.openssl.org/.) After you have the signing certificate, you must upload it to IAM; the user needs to keep the corresponding private key to use for signing requests. You can use IAM to upload the certificate. For more information about using signing certificates, see Managing User Keys and Certificates (p. 94).

Important
For security purposes, we recommend that you rotate your users' credentials on a regular basis. A user can have multiple access keys or signing certificates at a given time for this purpose. For more information, see Rotating Credentials (p. 106).

Passwords

A password enables a user to sign in to the AWS Management Console. You need to create a password for each user who needs to use the AWS Management Console. For more information about managing passwords, see Managing Passwords (p. 63).

Multi-Factor Authentication for Users

AWS offers Multi-Factor Authentication (MFA). MFA is an optional feature that requires the user to possess an authentication device and provide a randomly generated, six-digit code from the device in addition to the standard security credentials. For detailed information about AWS MFA, see the AWS Multi-Factor Authentication FAQs.

Your users can use MFA in these situations:

- When logging in to the AWS Management Console
- When requesting secure API actions

An example of a secure API action is the Amazon S3 action for deleting versioned objects. You can configure Amazon S3 versioning to require the requester to provide an MFA code in order to delete a versioned object. If you do, and you give a particular user permission to delete versioned objects, the user must provide an MFA code in the API call.

You can use IAM to enable an MFA device for a user. You can also use IAM to resynchronize the device, delete it, and list (discover) a user's MFA device. For information about using MFA devices, see Using Multi-Factor Authentication (MFA) Devices with AWS (p. 76).

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. 107).

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. 11).

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 use any Amazon SimpleDB actions with any of the AWS account's domains 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 Permissions (p. 107).

Policy

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

For more information about policies, see Overview of Policies (p. 109) and Managing IAM Policies (p. 112).

Other Permissions Systems

IAM has its own permissions system that you use to give entities the ability to do tasks with IAM resources in the AWS account.

IAM also integrates with individual AWS products so you can control an entity's access to those products' resources. Some of the AWS products 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 the product-specific sections described in AWS Services that Support IAM (p. 200).

Accessing IAM

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

- AWS Management Console
- Command line interface (CLI)
- AWS SDKs
- IAM Query API

The AWS Management Console provides a web-based user interface for managing IAM resources, such as the following:

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

For more information about accessing IAM through the console, see IAM and the AWS Management Console (p. 185). For a tutorial on using the console, see Creating an Admins Group Using the Console (p. 22).

You can also perform IAM tasks using command line interface (CLI) tools. If you are comfortable working at the command line, the CLI 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 IAM CLI, see AWS Identity and Access Management Command Line Interface Reference.

To access IAM programmatically, you can use one of the AWS SDKs. These provide libraries and sample code for various programming languages (Java, Ruby, .NET, etc.). For information about the AWS SDKs, including how to download and install them, see the Tools for Amazon Web Services page.

Alternatively, you can access IAM 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 AWS Identity and Access Management API Reference.

About IAM Entities

Topics

This section describes identifiers and limitations for the basic IAM entities (users, groups, and so on).

IAM Identifiers

Topics

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

Friendly Names and Paths

When you create a user, a role, or a group, or when you upload a server certificate, you give it a friendly name, such as Bob, TestApp1, Developers, or ProdServerCert. Whenever you need to specify a particular entity in an API call to IAM (for example, to delete a user, or update a group with a new user), you use the friendly name.

If you are using the API or command line interface to create users, you can also optionally give the entity a path that you define. 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. 11)).

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. 16).

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

You can use ARNs in IAM for users (IAM and federated), groups, roles, instance profiles, virtual MFA devices, and *Managing Server Certificates* (p. 191). 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:role/application_abc/component_xyz/S3Access
arn:aws:iam::123456789012:instance-profile/Webserver
arn:aws:sts::123456789012:federated-user/Bob
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::123456789012:user/division_abc/subdivi
sion_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 Policy Variables (p. 139)

You can use wildcards in the *`resource-id`* portion of the ARN to specify multiple users or groups. 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 or groups in the AWS account, use a wildcard after the `user/` or `group/`part of the ARN, respectively.

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

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

Don't use a wildcard in the `user/` or `group/` 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, go to 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/widget/*"
            }
         }
      }
   ]
}
```

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

User IDs

When IAM creates a user, group, 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 or group must be unique. For example, you might create an IAM user named David. Your company uses Amazon S3 and has a bucket with folders each employee; the bucket has an IAM resource 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 resource policy on the bucket is set using 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 user 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 user 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 CLI or API calls.

CLI calls:

- iam-instanceprofilegetattributes
- iam-rolegetattributes
- iam-servercertgetattributes
- iam-usergetattributes

API calls:

- ListInstanceProfiles
- ListInstanceProfilesForRole
- ListGroups
- ListRoles
- ListServerCertificates
- ListSigningCertificates
- ListUsers

Limitations on IAM Entities

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

> **Note**
> To retrieve account level information about entity usage and quotas, use the GetAccountSummary API action or the iam-accountgetsummary CLI command.

Following are restrictions on names:

- Names of users, groups, roles, instance profiles, and server certificates must be alphanumeric, including the following common characters: plus (+), equal (=), comma (,), period (.), at (@), and dash (-).
- Path names must begin with a forward slash (/).
- Policy names must be unique to the user, group, or role they are attached to, 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 (for more information, see http://www.ietf.org/rfc/rfc3986.txt).
- 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.

Names for entities are case sensitive and must be unique within the scope of your AWS account (regardless of the path you might give the entity).

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
- 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: 10
- AWS account aliases per AWS account: 1
- Login profiles per user: 1

You can request to increase these quotas for your AWS account on the IAM Limit Increase Contact Us Form.

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
- GUID (applicable to users, groups, roles, 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.
- You can add as many policies as you want to a user, role, or group, but the total aggregate policy size (the sum size of all 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

Getting Set Up

Topics

AWS Identity and Access Management (IAM) helps you securely control access to Amazon Web Services 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.

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. 21). To learn about using the IAM application programming interface (API) or command line interface (CLI) to create users under your AWS account, see Creating an Admins Group Using the CLI or API (p. 26). For more information about working with IAM users, see Users and Groups (p. 42).
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 (p. 151) or Cross-Account Access Using Resource-Based Policies (p. 164).

Do I Need to Sign Up for IAM?

IAM is a feature of your AWS account. If you are already signed up for a product that is integrated with IAM, you don't need to do anything else to sign up for IAM, nor will you be charged extra for using it.

> **Note**
> IAM works only with AWS products that are integrated with IAM. For a list of such products, see AWS Services that Support IAM (p. 200).

If you don't already have an AWS account, you need to create one to use IAM. You create an AWS account when you sign up to use an AWS product for the first time.

To sign up for AWS

1. Go to http://aws.amazon.com, 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.

AWS product documentation is available at AWS Documentation.

How Do I... ?

Here are some resources to help you get things done with AWS Identity and Access Management.

How Do I...	Relevant Resources
Learn more about the business cases for AWS Identity and Access Management	Business Use Cases (p. 38)
Get started with IAM	Getting Started (p. 21)
Get the release notes	Release Notes
Get the FAQ	AWS Identity and Access Management FAQ
Learn more about how IAM works	What Is IAM? (p. 1)
Get developer tools	Developer Tools
Get the Java library	Sample Code & Libraries
Get technical support	AWS Support Center
Get premium technical support	AWS Premium Support Center
Get community support	IAM Discussion Forums
Contact AWS	Contact Us

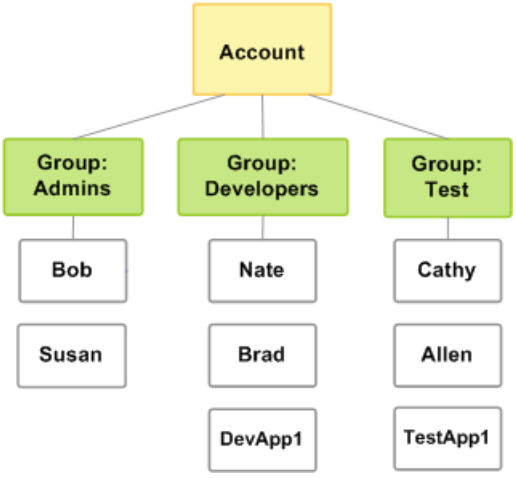
For definitions of AWS terms, go to the Amazon Web Services Glossary.

Getting Started

Topics

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 Admins group
- Create the policy controlling permissions for the group
- Create or add the users who will be in the Admins group
- Create access keys for users who need them
- Create passwords for users who need them

You will grant the Admins 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 cannot access your AWS account information, including the following:

- Account profile information
- Billing and metering information
- Security credentials

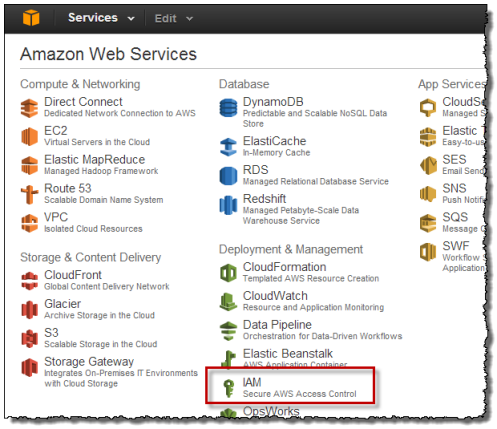
Tip
You should create a user for yourself and add it to your Admins group. Then, after you establish the Admins group and yourself as a user in the group, all interaction with your AWS account should be at the user level, not at the AWS account level. Limiting the use of your AWS account credentials will help ensure that when you want to rotate credentials for a user or for the AWS account, potential impact is limited. For more information about the credentials and the security benefits of rotating credentials, go to Managing User Keys and Certificates (p. 94) in *Using AWS Identity and Access Management*.

Creating an Admins Group Using the Console

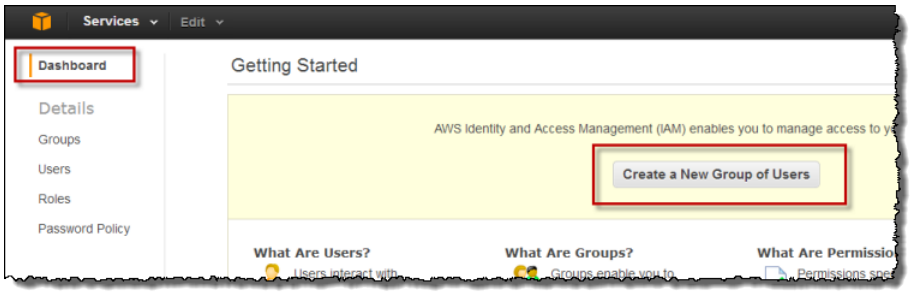
This procedure describes how to create an Admins group and grant the group the necessary permissions to access your AWS resources.

To create the Admins group

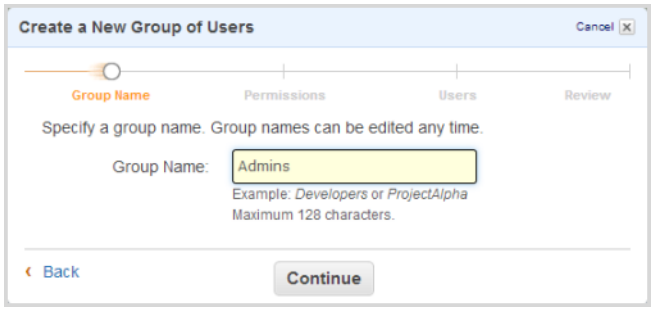
1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the AWS Management Console, choose **IAM.**



3. From the **IAM Dashboard**, click **Create a New Group of Users**.

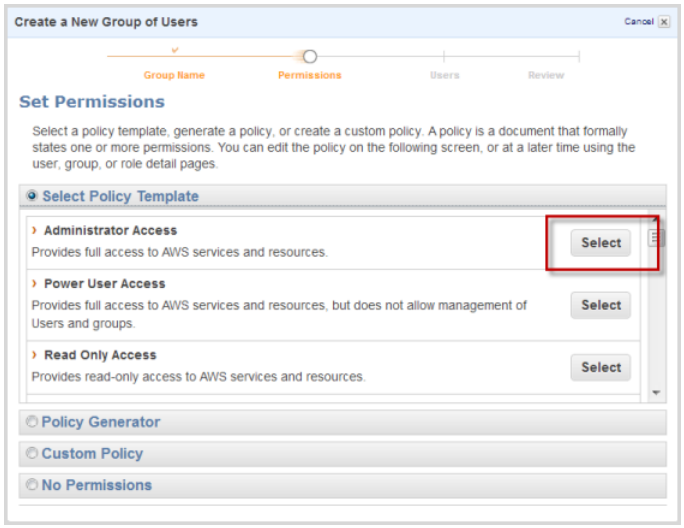


4. Name the group *Admins*, and then click **Continue**.



Note
You can use only certain characters to name a group. For information about limitations on group names, see Limitations on IAM Entities (p. 16) in *Using AWS Identity and Access Management*.

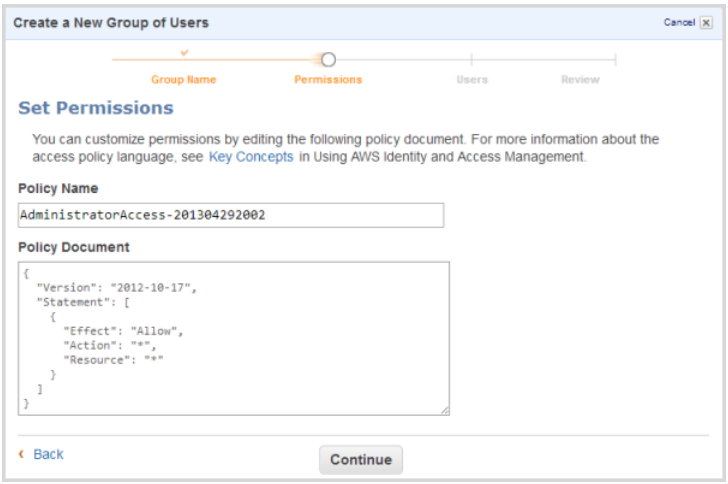
5. Next, you need to use a policy to assign permissions to your Admins group. A policy is a document that formally states one or more permissions. To select a policy template, next to **Administrator Access**, click **Select**.



IAM provides several policy templates you can use to automatically assign permissions to the groups you create. The Administrator Access policy template gives the Admins group permission to access all account resources, *except* your AWS account information as described previously.

6. Next, you can optionally edit the policy and the policy name. If you used a policy template, the default policy name includes the template name and today's date. If you want to make any changes to the

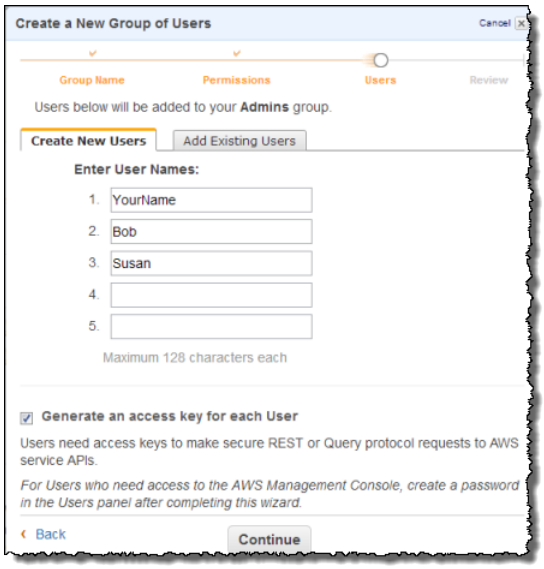
policy before you apply it to the group, you can use the **Edit Permissions** screen to edit the policy. If you don't want to edit the policy, or if you are finished making edits, click **Continue**. (Policies are described in detail in Permissions and Policies (p. 107) in *Using AWS Identity and Access Management*.)



Your Admins group has permissions; now it needs one or more users.

7. Add yourself as an administrator for your AWS account. Also add any other users you want to be administrators for your AWS account.

   a. To add yourself as a user to your Admins group, select the **Create New Users** tab and enter the user name you want to assign to yourself.



   b. Use the same process to create new users or add existing users to the Admins group.

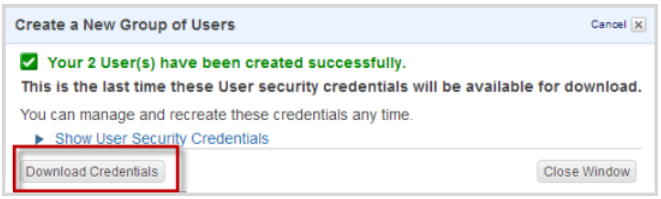
   c. If you want to automatically generate security credentials for your new users, on the **Create New Users** tab, make sure that **Generate an access key for each user** is selected. Users need security credentials if they will be working with an AWS API. For information about creating security credentials for existing users, go to Managing User Keys and Certificates (p. 94) in *Using AWS Identity and Access Management*.

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

Note
You can edit user properties later, but you cannot use the console to change the user name. For information about editing user properties, go to Users and Groups (p. 42) in *Using AWS Identity and Access Management*. For information on limitations on user names, see Limitations on IAM Entities (p. 16) in *Using AWS Identity and Access Management*.

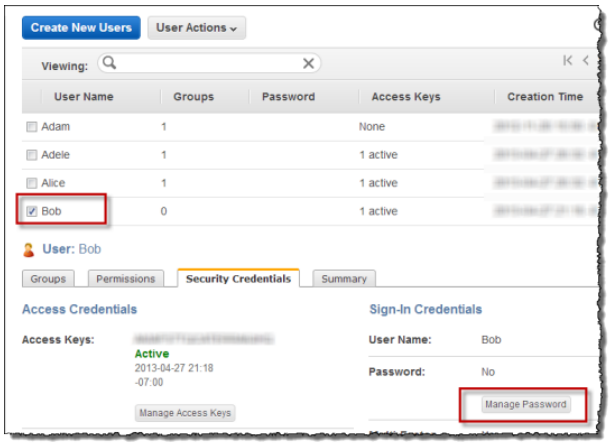
d. When you are finished adding users to this group, click **Continue**.

8. Review the group details on the confirmation screen, and then click **Continue**.
9. If you chose to create security credentials for the users, click **Download Credentials** to save the Access Key IDs and Secret Access Keys to a .csv file on your computer. You will need to provide each user with their Access Key ID and Secret Access Key before they can begin using an AWS API. For security reasons, you cannot retrieve the Secret Access Key after this step.

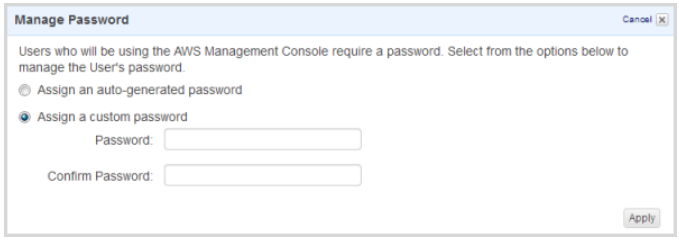


10. When you are finished downloading your users' security credentials, click **Close Window** to continue. Now you are now ready to add passwords for your users.
11. Users who will access the AWS Management Console will need a password. To add a password for a user

    a. In the navigation pane, click **Users**.
    b. Select the user who you want to create a password for, and then select the user **Security Credentials** tab.
    c. Click **Manage Password**.



    d. You can create a custom password, or you can have IAM automatically generate a password.



- To create a custom password, select **Assign a custom password**, and then enter and confirm the password. When you are finished, click **Apply**. The selected user can begin using the password. For information about how users access your sign-in page, see IAM and the AWS Management Console (p. 185) in *Using AWS Identity and Access Management*.
- To have IAM generate a password, select **Assign an auto-generated password**, and then click **Apply**. Click **Download Credentials** to save the password as a .csv file to your computer. You will need to provide this password to the user, and you will not be able to access the password after completing this step. Click **Close Window** to continue. For more information, see Get IAM Users Started in Your AWS Account (p. 31).

Congratulations, you have created an Admins group, added permissions for the group, and added users 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 user access to specific AWS resources, go to AWS Services that Support IAM (p. 200) in *Using AWS Identity and Access Management*.

Continue to Get IAM Users Started in Your AWS Account (p. 31)

Creating an Admins Group Using the CLI or API

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. You can stop using the AWS account's security credentials to make API calls to AWS.

Tip
If you read through the Getting Started (p. 21), 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 the *Getting Started*.

Process for Setting Up an Administrators Group

1	Create a group with a name of your choice (e.g., *Admins*). For more information, see Creating a Group (p. 27).
2	Add a policy that gives the group access to all AWS actions and resources. For more information, see Adding a Policy to the Group (p. 28).
3	Set up at least one user in the group. For more information, see Adding a New User to Your AWS Account (p. 42).

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. 16).

Command Line Interface

To create an administrators group

1. Use the `iam-groupcreate` command with the name you've chosen for the group and an optional path you've chosen. For more information about paths, see Friendly Names and Paths (p. 11).

   In this example, you create a group called *Admins* with the default path (/).

   ```
   PROMPT> iam-groupcreate -g Admins
   ```

   If the command is successful, there's no response.
2. Use the `iam-grouplistbypath` command to list the groups in your AWS account and confirm the group was created.

   ```
   PROMPT> iam-grouplistbypath
   ```

   Sample response:

   ```
   arn:aws:iam::123456789012:group/Admins
   ```

   The response lists the Amazon Resource Name (ARN) for your Admins 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.

API

To create an administrators group

1. Use the `CreateGroup` action with the name you've chosen for the group and an optional path you've chosen. The following example uses *Admins* as the name of the group.

   How you structure the `AUTHPARAMS` depends on how you are signing your API request. For information on `AUTHPARAMS` in Signature Version 4, go to Examples of Signed Signature Version 4 Requests. For information on `AUTHPARAMS` in Signature Version 2, go to Signature Version 2 Signing Process.

   For more information about the `CreateGroup` action, refer to the *AWS Identity and Access Management API Reference*, or refer to your SDK's documentation.

   ```
   https://iam.amazonaws.com/
   ?Action=CreateGroup
   &Path=/
   &GroupName=Admins
   &Version=2010-05-08
   &AUTHPARAMS
   ```

   IAM returns output similar to the following:

```
<CreateGroupResponse>
   <CreateGroupResult>
      <Group>
         <Path>/</Path>
         <GroupName>Admins</GroupName>
         <GroupId>AGPACKCEVSQ6C2EXAMPLE</GroupId>
         <Arn>arn:aws:iam::123456789012:group/Admins</Arn>
      </Group>
   </CreateGroupResult>
   <ResponseMetadata>
      <RequestId>7a62c49f-347e-4fc4-9331-6e8eEXAMPLE</RequestId>
   </ResponseMetadata>
</CreateGroupResponse>
```

2. Use the `ListGroups` action to list the groups in your AWS account and confirm the group was created.

```
https://iam.amazonaws.com/
?Action=ListGroups
&Version=2010-05-08
&AUTHPARAMS
```

Sample response:

```
<ListGroupsResponse>
   <ListGroupsResult>
      <Groups>
         <member>
            <Path>/</Path>
            <GroupName>Admins</GroupName>
            <GroupId>AGPACKCEVSQ6C2EXAMPLE</GroupId>
            <Arn>arn:aws:iam::123456789012:group/Admins</Arn>
         </member>
      </Groups>
      <IsTruncated>False</IsTruncated>
   </ListGroupsResult>
   <ResponseMetadata>
      <RequestId>7a62c49f-347e-4fc4-9331-6e8eEXAMPLE</RequestId>
   </ResponseMetadata>
</ListGroupsResponse>
```

Adding a Policy to the Group

This section shows how to add a policy that lets any user in the group perform any action on any resource in the AWS account. You write the policy using the *access policy language*, and then attach the policy to the Admins group. For more information about the access policy language, see Overview of Policies (p. 109).

Command Line Interface

To add a policy giving root privileges

1. Create a policy document by copying and pasting the following JSON block into a text file.

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

2. Save the text file as `MyPolicy.txt` in the main CLI installation directory.
3. Type the following command to upload this policy to IAM and attach it to your Admins group. The command assigns the following name to the policy: *AdminRoot*.

```
PROMPT> iam-groupuploadpolicy -g Admins -p AdminRoot -f MyPolicy.txt
```

   If the command is successful, there's no response.
4. Type the following command to confirm the policy is attached to the Admins group.

```
PROMPT> iam-grouplistpolicies -g Admins
```

   The response lists the names of the policies attached to the Admins group. Sample response:

```
AdminRoot
```

In the preceding policy, you could have used `iam-groupaddpolicy` instead of the `iam-groupuploadpolicy`. The `iam-groupaddpolicy` command lets you specify the contents of the policy as parameters instead of uploading a JSON policy document file. Following is an example of what the `iam-groupaddpolicy` command would look like. The `-o` option causes the output to include the JSON policy document we construct for you based on the parameters you provided.

```
PROMPT> iam-groupaddpolicy -g Admins -p AdminRoot -e Allow -a "*" -r "*" -o
```

Sample response:

```
{"Version":"2012-10-17","Statement":[{"Effect":"Allow","Action":["*"],"Re
source":["*"]}]}
```

API

To add a policy giving root privileges

1. Create the following JSON policy.

```
{
   "Version":"2012-10-17",
   "Statement":[{
      "Effect":"Allow",
```

```
      "Action":"*",
      "Resource":"*"
      }
   ]
}
```

2. Issue a `PutPolicy` request. The following example names the policy *AdminRoot* and attaches it to the Admins group. The policy document in the request doesn't include white space or line feeds, and it's not URL encoded here so you can more easily see what it looks like (normally you must form URL encode it as part of POST request). For more information about how to do a POST request, go to Examples of Signed Signature Version 4 Requests, or Signature Version 2 Signing Process.

```
POST / HTTP/1.1
Host: iam.amazonaws.com
Content-Type: application/x-www-form-urlencoded

Action=PutPolicy
&GroupName=Admins
&PolicyName=AdminRoot
&PolicyDocument={"Version":"2012-10-17","Statement":[{"Effect":"Allow","Ac
tion":"*","Resource":"*"}]}
&Version=2010-05-08
&AUTHPARAMS
```

Sample response:

```
<PutPolicyResponse>
   <ResponseMetadata>
      <RequestId>7a62c49f-347e-4fc4-9331-6e8eEXAMPLE</RequestId>
   </ResponseMetadata>
</PutPolicyResponse>
```

3. Issue a `ListGroupPolicies` request to confirm the policy is attached to the Admins group.

```
https://iam.amazonaws.com/
?Action=ListGroupPolicies
&GroupName=Admins
&Version=2010-05-08
&AUTHPARAMS
```

Sample response:

```
<ListGroupPoliciesResponse>
   <ListGroupPoliciesResult>
      <PolicyNames>
         <member>AdminRoot</member>
      </PolicyNames>
      <IsTruncated>false</IsTruncated>
   </ListGroupPoliciesResult>
   <ResponseMetadata>
      <RequestId>7a62c49f-347e-4fc4-9331-6e8eEXAMPLE</RequestId>
   </ResponseMetadata>
</ListGroupPoliciesResponse>
```

You can confirm the contents of a particular policy with the `GetUserPolicy` or `GetGroupPolicy` actions. For information about the actions, go to the AWS Identity and Access Management API Reference, or refer to your SDK's documentation.

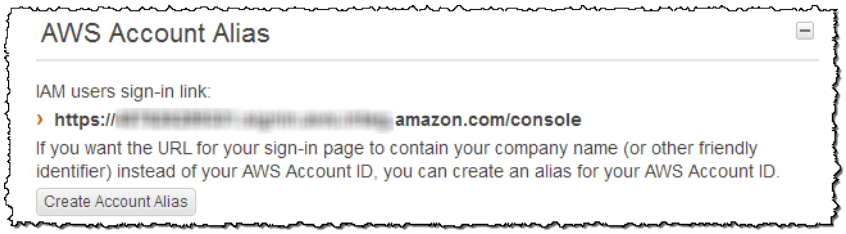
Important
After you have the administrators group set up, you must add at least one user to it. For more information about adding users to a group, see Adding a New User to Your AWS Account (p. 42).

Get IAM Users Started in Your AWS Account

When you are finished creating groups, assigning permissions, and adding users, you need to provide each user with the following information:

- The user name
- The password (if you created one for the user)
- The Access Key ID and Secret Access Key (if you created these for the user)
- The AWS account sign-in page URL (if the user will be accessing your account through the AWS Management Console)

Your AWS account sign-in page URL is displayed on the IAM Dashboard under **AWS Account Alias**. For information about creating an account alias for the URL, go to Using an Alias for Your AWS Account ID (p. 188) in *Using AWS Identity and Access Management*.



Continue to Where to Go Next (p. 31)

Where to Go Next

Topics

The Getting Started 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. 38) describes how you might use IAM with other AWS products.
- Users and Groups (p. 42) has important details you need to know about managing users and groups in your AWS account.
- AWS Services that Support IAM (p. 200) links to information about how IAM works with other AWS products (for example, Amazon EC2, Amazon S3, etc.), and what you can do to control your users' access to specific AWS products and resources.
- *Friendly Names and Paths* in IAM Identifiers (p. 11) 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. 107) describes permissions, policies, and the access policy language you use to write policies.
- Example IAM Policies (p. 116) provides examples of policies that control access for your IAM resources.
- The AWS Policy Generator is a tool you can use to generate AWS policies automatically.

Other Ways to Access IAM

This guide 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 Command Line Interface to Issue Commands

If you want to issue commands with the IAM command line interface (CLI), go to the AWS Identity and Access Management Command Line Interface Reference. The guide describes how to set up and use the IAM 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. The following SDKs are available:

- AWS SDK for Java
- AWS SDK for PHP
- AWS SDK for .NET

Code Directly to the Web Service API

If you want to write code directly to the IAM Query API, go to AWS Identity and Access Management API Reference. The guide describes how to create and authenticate API requests. Making Query Requests (p. 209) describes the basics about using the Query API.

IAM Resources

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

Resource	Description
Using AWS Identity and Access Management	Describes how to use the service and all its features through the AWS Management Console, the CLI, or API.
AWS Identity and Access Management Command Line Interface Reference	Gives complete descriptions of the commands in the CLI.
AWS Identity and Access Management API Reference	Gives the WSDL and schema location; complete descriptions of the API actions, parameters, and data types; and a list of errors that the service returns.
AWS Identity and Access Management Quick Reference Card	Gives a concise listing of the commands you use with the CLI.
Product Information for IAM	The primary web page for information about IAM.
IAM Release Notes	The release notes give a high-level overview of the current release. They specifically note any new features, corrections, and known issues.
AWS Developer Resource Center	A central starting point to find documentation, code samples, release notes, and other information to help you build innovative applications with AWS.
IAM Discussion Forum	A community-based forum for developers to discuss technical questions related to IAM.
AWS Support Center	The home page for AWS Technical Support, including access to AWS developer forums, technical FAQs, service health dashboard, and premium support (if you are subscribed to this program).
AWS Premium Support Information	The primary web page for information about AWS Premium Support, a one-on-one, fast-response support channel to help you build and run applications on AWS Infrastructure Services.
Contact Us	A central contact point for inquiries concerning AWS billing, your AWS account, events, abuse, etc.
Conditions of Use	Detailed information about copyright and trademark usage at Amazon.com 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

Any request you make for an AWS resource requires credentials so that AWS can make sure you have permission to access the resource. When you created your AWS account, you got an access key ID and a secret access key. Our recommendation: don't use those keys.

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

Lock away your AWS root account credentials

The access key ID and secret access key for your AWS account give you access to all your resources, including your billing information. Therefore, protect your AWS account credentials like you would your credit card numbers or any other secret in these ways:

- Delete your AWS root account access key ID and secret access key, or at least rotate (change) the credentials regularly. For information about managing your AWS account password, see Changing Your AWS Account Password (p. 64).
- Never share your AWS root account credentials with anyone. The remaining sections of this document discuss various ways to avoid having to share your root 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.
- Enable AWS multi-factor authentication (MFA) on your AWS root account. For more information, see Using Multi-Factor Authentication (MFA) Devices with AWS (p. 76).

Create individual 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.

For more information, see Users and Groups (p. 42).

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 Admins Group Using the CLI or API (p. 26)
- Adding Users to and Removing Users from a Group (p. 58)

Grant least privilege

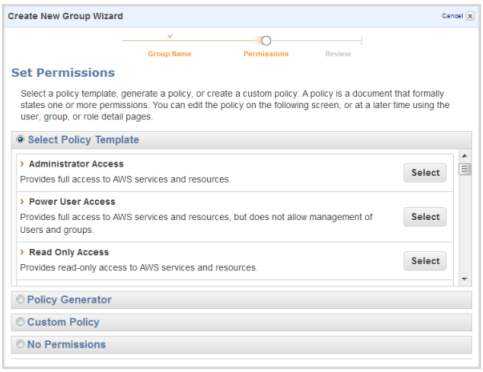
When you create IAM policies, follow the standard security advice of granting *least privilege*—that is, granting only the permissions required to perform a task. Determine what users need to do and then craft policies for them that let the users perform *only* those tasks. Similarly, create policies for individual resources (such as 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 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 policy templates in the console. These templates 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. 107)
- IAM Policy Evaluation Logic (p. 145)
- 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 allow users to change their own passwords, make sure that they create strong passwords. In the Password Policy section of the IAM console, you can choose options for password policy, such as the password's minimum length, whether it requires a non-alphabetic character, and so on.

For more information, see Managing an IAM Password Policy (p. 65).

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. 76).

Use roles for applications that run on EC2 instances

Applications that run on an 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 EC2).

When you launch an 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 Granting Applications that Run on Amazon EC2 Instances Access to AWS Resources (p. 152).

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 Cross-Account Access Using Resource-Based Policies (p. 164).

Rotate credentials regularly

Change your own passwords and access keys regularly, and make sure that all IAM users 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.

To make it easier to rotate credentials, let your users manage their own passwords.

For more information, see Rotating Credentials (p. 106).

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 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. 128).

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 SEC 303: Top 10 AWS Identity and Access Management (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 AWS products, including how to implement individual APIs, see AWS Services that Support IAM (p. 200).

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).

Joe then assigns himself (as Joe the user) to the Admins group. At this point, he stops using the AWS account's credentials to interact with AWS, and instead he begins using 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 Admins Group Using the CLI or API (p. 26). For information about creating users, see Adding a New User to Your AWS Account (p. 42).

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 Elastic Compute Cloud User Guide.

Amazon EC2 Permissions for the Groups

To provide "perimeter" control, Joe adds 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 adds a policy to the SysAdmins group that gives members of the group permission to use all the EC2 actions.
- **Developers**—Need the ability to work with instances only. Joe therefore adds a policy to the Developers group that allows developers to call `DescribeInstances`, `RunInstances`, `StopInstances`, `StartInstances`, and `TerminateInstances`. Amazon EC2 currently doesn't support IAM policies that restrict access to a particular AMI, volume, instance, etc., so Example Corp. can control only which Amazon EC2 APIs the developers can call.

  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 adds a policy to the Managers group that only lets them call EC2 "Describe" APIs.

For examples of what these policies might look like, see Example IAM Policies (p. 116) and Using AWS Identity and Access Management in the *Amazon Elastic Compute Cloud User Guide*.

User's Role Change

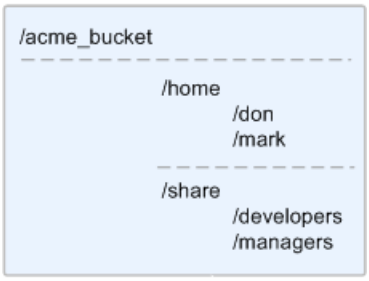
At some point, one of the developers, Don, changes roles and becomes a manager. Joe moves Don from the Developers group to the Managers group. Now that he's in the Managers group, Don's ability to interact with Amazon EC2 instances is limited. He can't launch or start instances. He also can't stop or 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.



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 assigns 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 assigns 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 assigns 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 assigns 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 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. 145).

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.

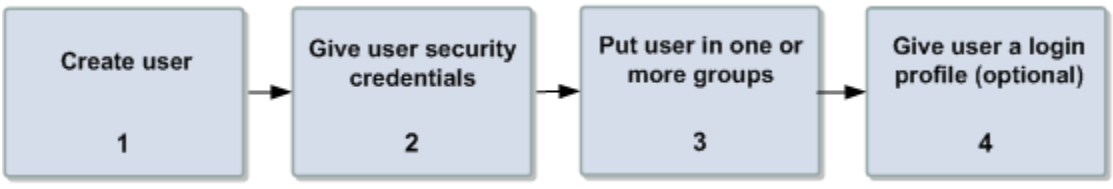
Users and Groups

Topics

This section describes how to create and manage users and groups.

Adding a New User to Your AWS Account

When someone joins your organization, or if you have a new application or system that needs to make API calls to AWS, you need to add a new user to your AWS account. This section describes the process for setting up a user in IAM.



In outline, the process consists of these steps:

1. Create a user.
2. If the user will be making API calls or using the command line interface, create security credentials (an access key ID and a secret access key).

3. Optionally add the user to one or more groups. If you've decided to create a group that includes all your AWS account's users (for example, an *AllUsers* group), make sure to add the new user to that group.
4. If the user needs to use the AWS Management Console, create a password for the user. For more information about passwords and optional MFA devices for securing passwords, see Passwords (p. 8) and Multi-Factor Authentication for Users (p. 8).

   If you want to let the user manage his or her own security credentials, give the user permissions to do so. (By default, users do not have permissions to manage their own credentials.) For more information, see Adding a Credentials Management Policy for Users (p. 104).

How you execute the tasks in the preceding table depends on which interface you're using to access IAM. The interface-specific details are covered in the sections that follow.

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. 16).

AWS-Assigned User Identifiers

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

- 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 specifying the user as a principal in an IAM policy for an S3 bucket.
- A globally unique identifier (GUID) for the user. (This ID is returned only when you use the API or CLI to create the user, not in the console.)

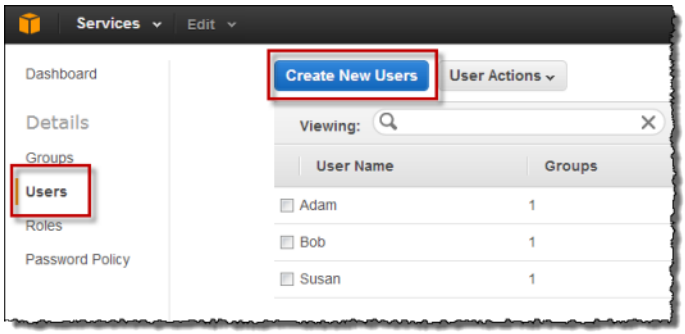
For more information about these identifiers, see IAM Identifiers (p. 11).

AWS Management Console

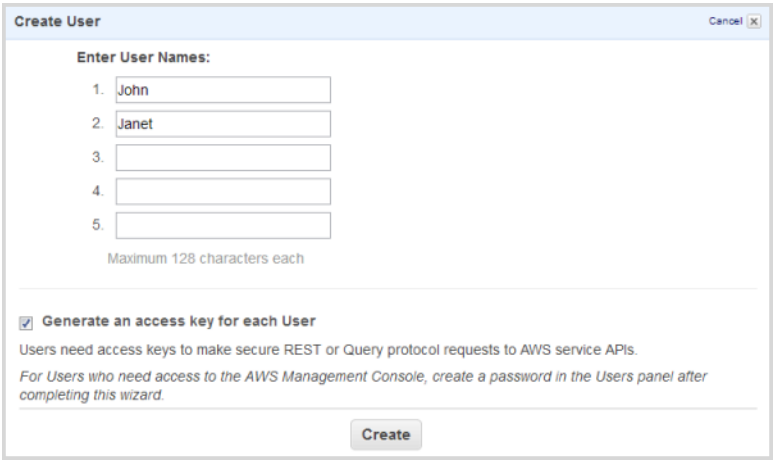
If you're accessing IAM through the AWS Management Console, you can create and manage users and their security credentials in a few simple steps.

To add a user

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane of the console, click **Users**, and then click **Create New Users**.



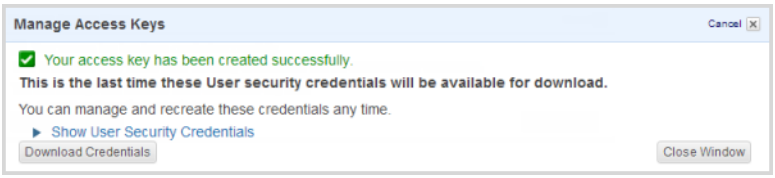
3. Enter the user names for users you want to create. You can create up to five users at one time.



4. If you want to automatically generate an Access Key ID and Secret Access Key for your new users, select **Generate an access key for each user**. Users need keys if they will be working with an AWS API. For more information about creating user keys for existing users, go to Managing User Keys and Certificates (p. 94) in *Using AWS Identity and Access Management*.

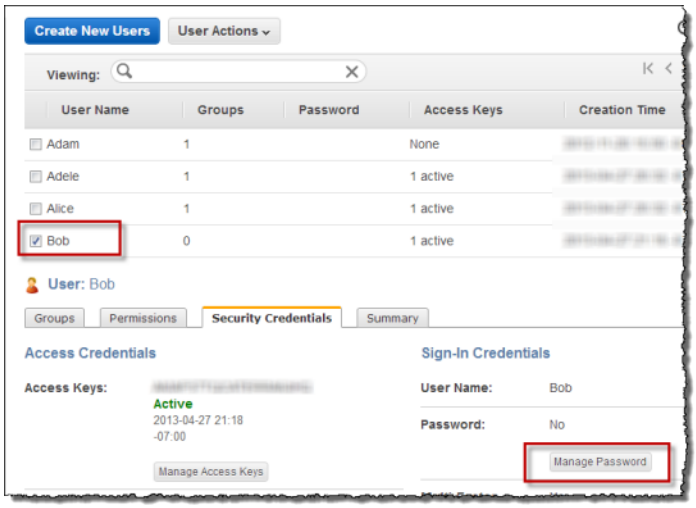
   Note
   If you have users who will be working with the AWS Management Console, you will need to create passwords for each of them. Creating passwords is described later in this procedure.

5. Click **Create**.
6. If you chose to create security credentials for the users, click **Download Credentials** to save the Access Key IDs and Secret Access Keys to a .CSV file on your computer. You will not have access to the Secret Access Keys 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.



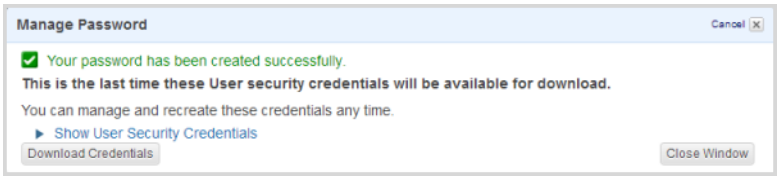
7. When you are finished downloading your users' security credentials, click **Close Window** to continue.
8. Users who will access the AWS Management Console will need a password. To add a password for a user, follow these steps:

   a. In the navigation pane, click **Users**.
   b. Select the user who you want to create a password for, and then select the user **Security Credentials** tab.
   c. Click **Manage Password**.



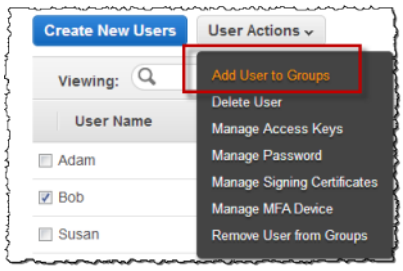
d. You can create a custom password, or you can have IAM automatically generate a password.

- To create a custom password, select **Assign a custom password**, and then enter and confirm the password. When you are finished, click **Apply**. The selected user can begin using the password. For information about how users access your sign-in page, see The AWS Management Console Sign-in Page (p. 185).
- To have IAM generate a password, select **Assign an auto-generated password**, and then click **Apply**. Next, click **Download Credentials** to save the password as a .CSV file to your computer. You will need to provide this password to the user, and you will not be able to access the password after completing this step. Click **Close Window** to continue.

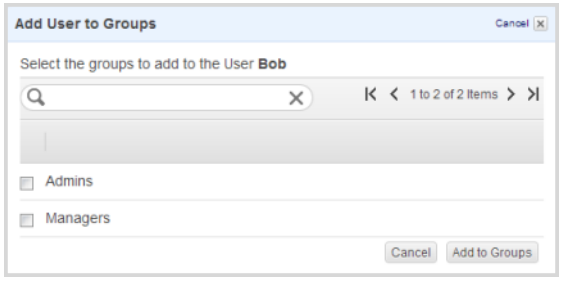


9. To add the user to a group, follow these steps:

a. Select the user name, and then, from the **User Actions** list, select **Add User to Groups**.



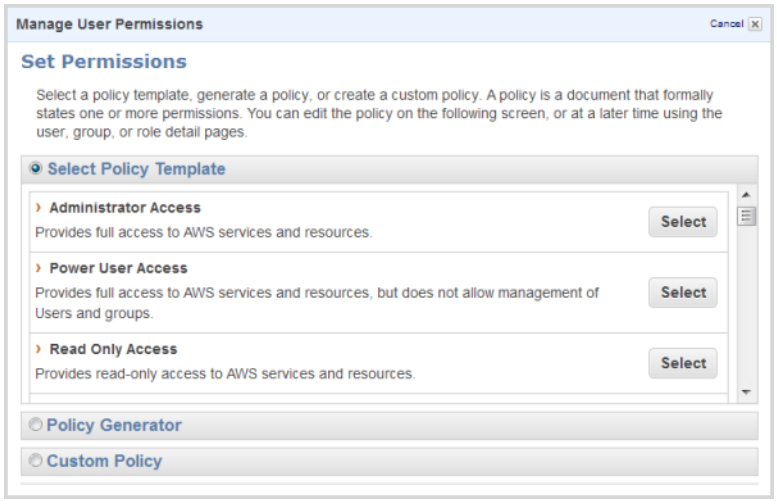
b. Select the groups you want to add the user to, and then click **Add to Groups**.



10. To attach a policy to the user, follow these steps:

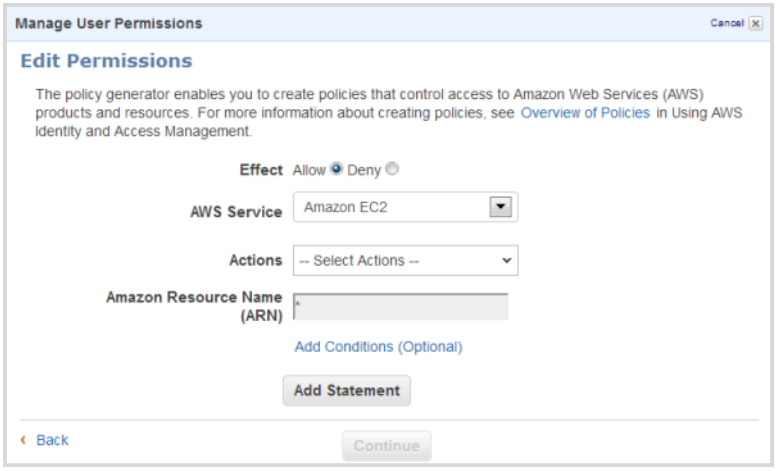
    a. Select the **Permissions** tab, and then click **Attach User Policy**.

    b. Choose the method for creating the policy document by clicking either **Select Policy Template**, **Policy Generator**, or **Custom Policy**. Then click **Select**.



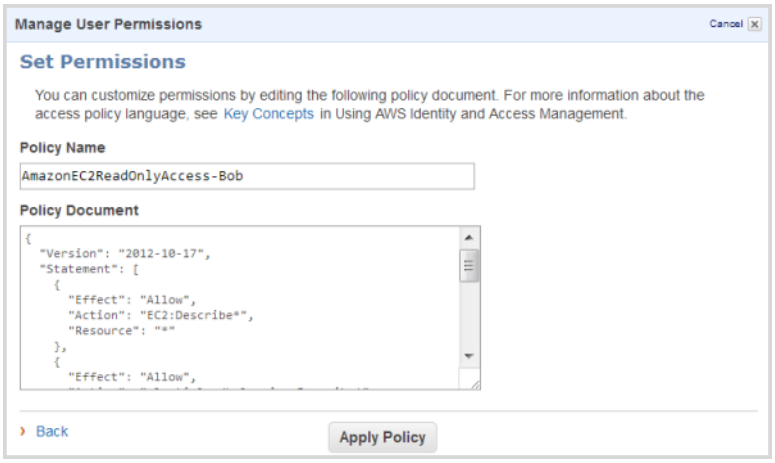
    c. How you complete the next step depends on the method you selected to create the policy.

    - If you are using a template to create the policy, review the policy content in the dialog box, then click **Apply Policy**.
    - If you are using the policy generator, select the appropriate **Effect**, **AWS Service**, and **Actions** options, enter the 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 **Continue**. Review the generated output, then click **Apply Policy**.

- If you are using a custom policy, enter a name for the policy under **Policy Name** and write the policy or paste the policy document from your text editor into the **Policy Document** box. When you are finished, click **Apply Policy**. (For information about policy limitations, see Limitations on IAM Entities (p. 16))



IAM applies the policy to the user.

Users can now perform the tasks in your account that their permissions allow. If you have given users a password and permissions to work in the AWS Management Console, they can log in using a URL that's specific to your account. The form is this:

```
https://account-number.signin.aws.amazon.com/console
```

For example, the URL might look like this:

```
https://123456789012.signin.aws.amazon.com/console
```

For more information about how users sign in to the AWS Management Console, see IAM and the AWS Management Console (p. 185).

For information about optionally giving users permission to manage their own security credentials, see Adding a Credentials Management Policy for Users (p. 104).

Command Line Interface

If you're using the IAM command line interface, you can do tasks 1–3 in the preceding table with one command: `iam-usercreate`. The command's main purpose is task 1 (creating the user), and it optionally does tasks 2 and 3. You can perform tasks 2 and 3 separately with other commands (`iam-useraddkey` and `iam-groupadduser`).

To give the user a password, use the `iam-useraddloginprofile` command. To give the user permission to manage his or her own security credentials, use the `iam-useraddpolicy` or `iam-useruploadpolicy` commands.

The following example creates a user named Bob, creates an access key for Bob, and assigns Bob to the Developer group.

```
PROMPT> iam-usercreate -u Bob -g Developer -k -v
```

Sample response:

```
arn:aws:iam::123456789012:user/Bob
AIDACKCEVSQ6C2EXAMPLE
```

The command used verbose mode (`-v`), so the response includes the user's ARN and GUID. If you don't use verbose mode, the response is empty.

We recommend saving the user's new Access Key ID and Secret Access Key to a text file, such as `Bob-credentials.txt`. You can give that file to Bob so he'll have his credentials to use with calls to AWS.

The following example creates the password *Welcome* for the user Bob.

```
PROMPT> iam-useraddloginprofile -u Bob -p Welcome
```

If the command is successful, there is no response.

You can list the users in your AWS account or in a particular group with the `iam-userlistbypath` and `iam-grouplistusers` commands.

For information about optionally giving users permission to manage their own security credentials, see Adding a Credentials Management Policy for Users (p. 104).

API

If you're programmatically accessing IAM, you use a separate API call for each of the tasks involved in setting up a new user. The following table lists the API actions to use.

Process for Adding a New User

1	Create a user: `CreateUser`

2	Create security credentials for the user: `CreateAccessKey`
3	Optionally add the user to one or more groups: `AddUserToGroup`
4	Optionally give the user a password, which is required if the user needs to use the AWS Management Console: `CreateLoginProfile`

You can also optionally give the user permission to manage his or her own security credentials. For more information, go to `PutUserPolicy`.

For more information about the actions, go to the AWS Identity and Access Management API Reference, or refer to your SDK's documentation.

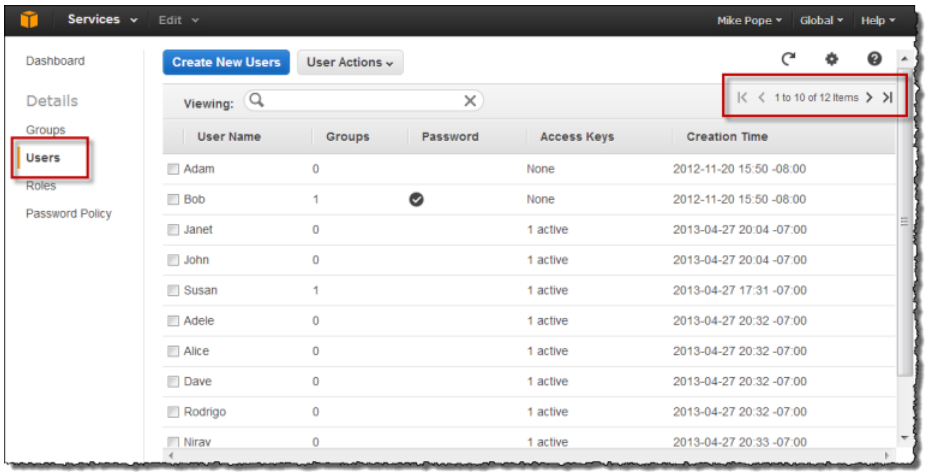
Listing Users

This section describes how to list the users in your AWS account or in a particular group.

AWS Management Console

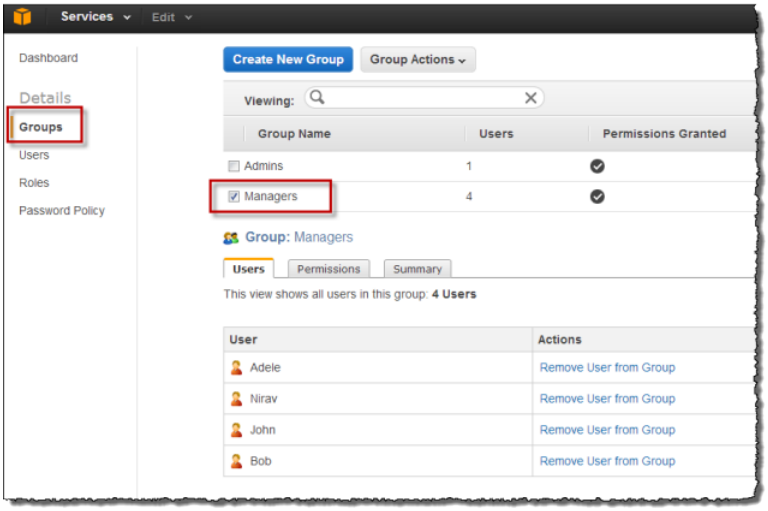
To list all users in your AWS account

- In the navigation pane, click **Users**. The console displays ten users per screen. To see more users, use the page controls on the right side of the console to advance to the next screen.



To list all users in a group

- In the navigation pane, click **Groups**, and then select the group name. The console displays group details. If necessary, select the **Users** tab to display all users belonging to the group.



Command Line Interface

The `iam-userlistbypath` command lets you list all the users in the AWS account or list all the users with a particular path prefix. The output lists the ARN for each resulting user.

The `iam-userlistgroups` command lists all the groups a particular user is in. The output lists the ARNs for the groups the user is in. For more information about ARNs and paths, see IAM Identifiers (p. 11).

For more information about these commands, refer to the *AWS Identity and Access Management Command Line Interface Reference*.

API

The `ListUsers` action lets you list all the users in the AWS account, or list all the users with a particular path prefix. The response lists the path, ARN, and GUID for each resulting user. For more information about paths, ARNs, and GUIDs, see IAM Identifiers (p. 11).

If you want to get a list of users in a particular group, use `GetGroup`.

For more information about `ListUsers` and `GetGroup`, go to the AWS Identity and Access Management API Reference, or refer to your SDK's documentation.

Paginating the Results

Some of the API actions such as `ListUsers` and `GetGroup` let you paginate the results. Note that these actions limit the number of users returned in the response, so to get the full list of users, you must call the action multiple times.

To paginate the list results

1. In the first request, use a *`MaxItems`* parameter specifying how many users you want in the response.

```
https://iam.amazonaws.com/
?Action=ListUsers
&MaxItems=10
```

```
&Version=2010-05-08
&AUTHPARAMS
```

The response includes up to *MaxItems* number of users (assuming there are at least that many total). If there are more users left beyond that maximum, the response includes an element called IsTruncated with a true value (otherwise, the value is false). The response also includes a Marker element with a long string as the value.

```
<ListUsersResponse>
   <ListUsersResult>
      <Users>
         <member>
            ...
         </member>
         <member>
            ...
        </member>
      ...
      </Users>
      <IsTruncated>true</IsTruncated>
      <Marker>AAHu6JklqUhsapsw5lZ5xq/bjVoYLSbLeLoLhUS4GlRQK2UYzMW40g
39qbmiJeRLnkYI4WUSXIT45gZEXAMPLE</Marker>
   </ListUsersResult>
   <ResponseMetadata>
      <RequestId>7a62c49f-347e-4fc4-9331-6e8eEXAMPLE</RequestId>
   </ResponseMetadata>
</ListUsersResponse>
```

2. In the subsequent request to continue listing the users, again include the *MaxItems* request parameter, and a *Marker* request parameter set to that long Marker value from the response.

```
https://iam.amazonaws.com/
?Action=ListUsers
&MaxItems=10
&Marker=AAHu6JklqUhsapsw5lZ5xq/bjVoYLSbLeLoLhUS4GlRQK2UYzMW40g39qbmiJeRLnkYI4
WUSXIT45gZEXAMPLE
&Version=2010-05-08
&AUTHPARAMS
```

If there are still more users to be listed, the IsTruncated value in the response is true, and there's a new value for Marker for you to use in the next request to continue listing.

Deleting a User from Your AWS Account

Topics

You can remove a user from your AWS account at any time. For example, you might do this if someone quits your company. The process of removing a user deletes the user's credentials and policies. If the user is only temporarily leaving the company, you could instead just disable the user's credentials instead of deleting the user entirely from the AWS account. That way, you can easily prevent the user from

accessing the AWS account's resources during the absence, without removing the user's credentials or policies. For more information about disabling credentials, see Managing User Keys and Certificates (p. 94).

AWS Management Console

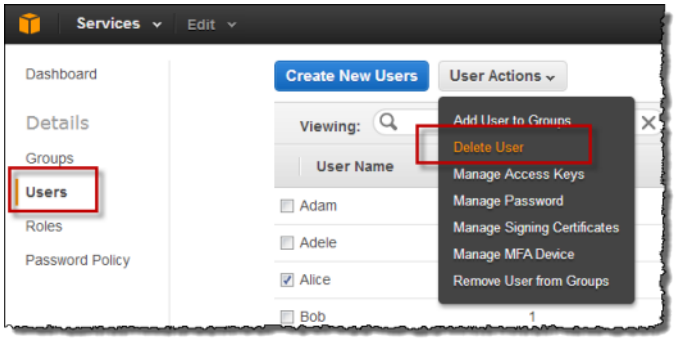
When you use the AWS Management Console to delete an IAM user, IAM also deletes all the information associated with the user. IAM deletes the following information when you delete a user:

- The user
- Any signing certificates belonging to the user
- Any access keys belonging to the user
- Any associated MFA device
- Any password associated with the user
- All policies attached to the user (policies that are applied to a user via group permissions are not affected)

Also, the user is removed from any groups it belongs to.

To use the AWS Management Console to delete a user

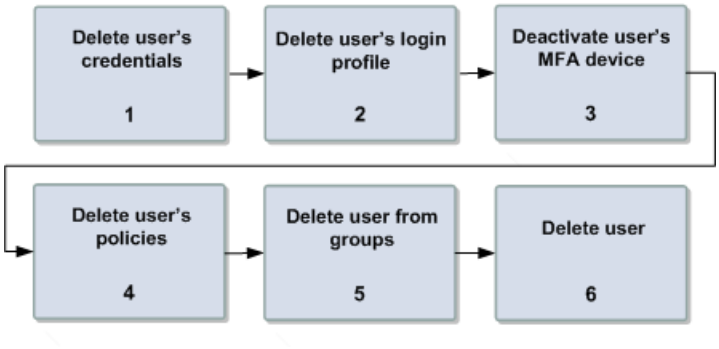
1. In the navigation pane, click **Users**, and then select the user name.
2. From the **User Actions** list, select **Delete User**.



3. Review your changes, and then click **Yes, Delete**.

Command Line Interface or API

The following figure and table describe the process for using the IAM CLI or API to delete a user from your AWS account.



Process for Deleting a User from Your AWS Account

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.
2	Delete the user's password, if the user has one. For a general description of using IAM to manage passwords, see Passwords (p. 8).
3	Deactivate the user's MFA device, if the user has one. For a general description of MFA, see Multi-Factor Authentication for Users (p. 8).
4	Delete any policies that were attached to the user.
5	Get a list of any groups the user was in, and delete the user from those groups.
6	Delete the user.

> **Note**
> If you delete a user, any residual remote references to that user (e.g., an Amazon SQS policy) display the GUID in the user's ARN instead of the user's friendly name. If you've stored the GUID in your own system, you can then use the displayed GUID to identify the deleted user being referred to.

How you actually execute the tasks in the preceding table depends on which interface you're using to access IAM. The interface-specific details are covered in the sections that follow.

Command Line Interface

If you're using the command line interface to access IAM, you can use a single command for each of the tasks involved in deleting a user from your AWS account, or you can perform each step individually. The following table lists the commands to use when deleting the user in steps.

To delete the user recursively, use the `iam-userdel` with the `-r` option. Recursively deleting the user automatically deletes it from any associated groups and deletes any attached entities such as keys, signing certificates, and policies. Use the `-r` option with caution. Before performing a recursive delete, to ensure you are not deleting anything you don't want to, use the `-p` option along with the `-r` option to list all the user's associated entities and groups without actually performing the recursive deletion. For more information about the commands, go to the AWS Identity and Access Management Command Line Interface Reference.

Process for Deleting a User from Your AWS Account

1	Delete the user's keys and certificates: `iam-userdelkey` and `iam-userdelcert`

2	Delete the user's password, if the user has one: `iam-userdelloginprofile`
3	Deactivate the user's MFA device, if the user has one: `iam-userdeactivatemfadevice` or `iam-virtualmfadevicedelete` (if the user has a virtual MFA device)
4	Delete any policies that were attached to the user: `iam-userlistpolicies` (to list the policies attached to the user) and `iam-userdelpolicy`
5	Get a list of any groups the user was in, and delete the user from those groups: `iam-userlistgroups` and `iam-groupremoveuser`
6	Delete the user: `iam-userdel`

API

If you're programmatically accessing IAM, you use a separate API call for each of the tasks involved in setting up a new user. The following table lists the API actions to use.

Process for Deleting a User from Your AWS Account

1	Delete the user's keys and certificates: `DeleteAccessKey` and `DeleteSigningCertificate`
2	Delete the user's password, if the user has one: `DeleteLoginProfile`
3	Deactivate the user's MFA device, if the user has one: `DeactivateMFADevice` or `DeleteVirtualMFADevice` (if the user has a virtual MFA device)
4	Delete any policies that were attached to the user: `ListUserPolicies` (to list the policies attached to the user) and `DeleteUserPolicy`
5	Get a list of any groups the user was in and delete the user from those groups: `ListGroupsForUser` and `RemoveUserFromGroup`
6	Delete the user: `DeleteUser`

For more information about the actions, go to the AWS Identity and Access Management API Reference, or refer to your SDK's documentation.

Creating and Listing Groups

Topics

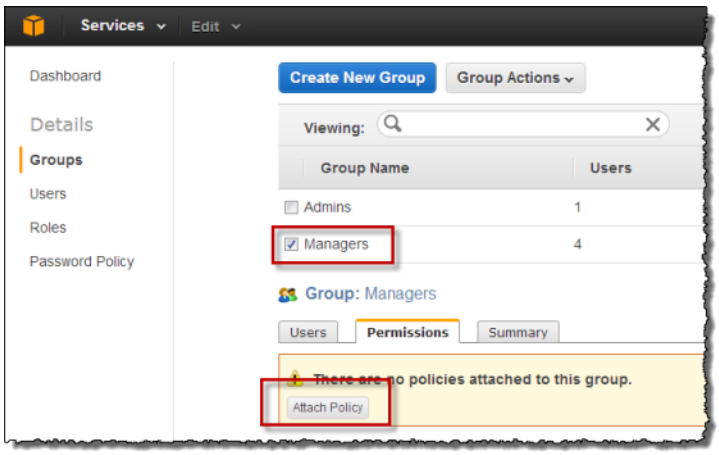
To set up a group, you need to create the group and then give it permissions based on the type of work you expect the users in the group to do. For an example of how to set up a group, go to the Getting Started (p. 21).

Attaching a Policy to a Group

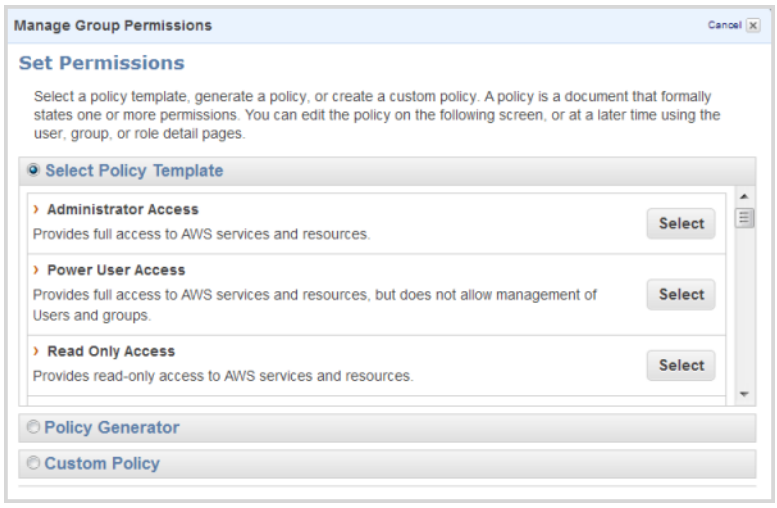
To give a group permissions, you attach a policy to the group as described in the following procedure. For information about permissions and policies, see Permissions and Policies (p. 107).

To attach a policy to a group

1. In the navigation pane, select **Groups**.
2. Under **Group Name**, select the group that you want to attach a policy to.
3. Select the **Permissions** tab, and then click **Attach Policy**.

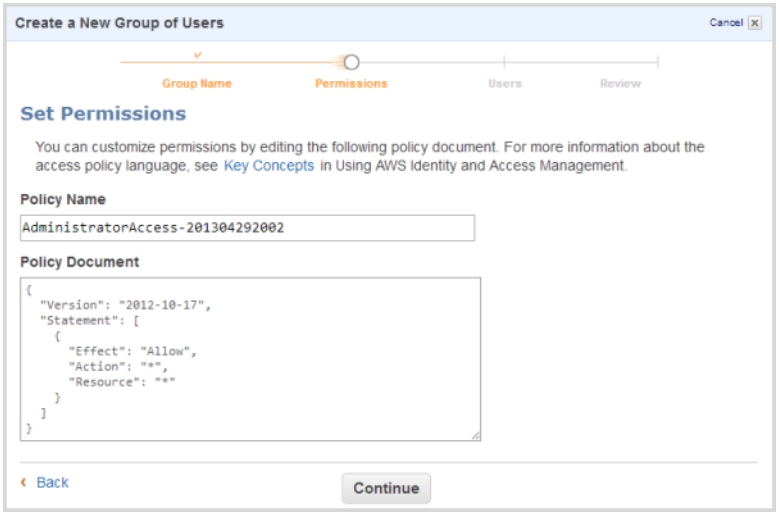


4. Choose the method for creating the policy document by clicking either **Select Policy Template**, **Policy Generator**, or **Custom Policy**. Then click **Select**.



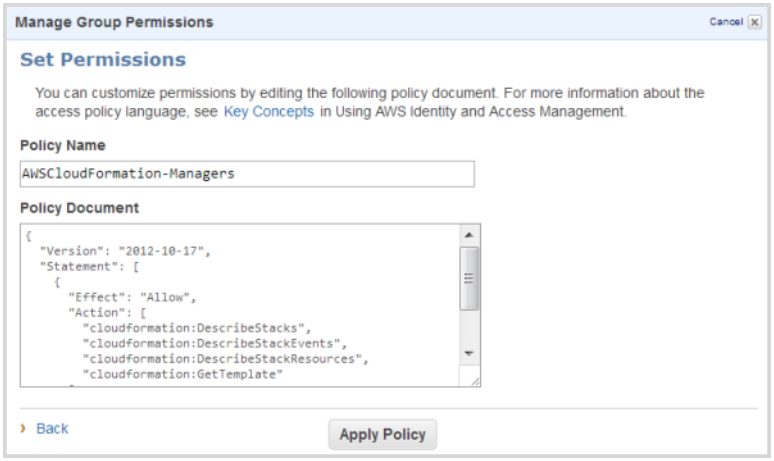
5. How you complete the next step depends on the method you selected to create the policy.

   - If you are using a template to create the policy, review the policy content in the dialog box, then click **Apply Policy**.
   - If you are using the policy generator, select the appropriate **Effect**, **AWS Service**, and **Actions** options, enter the 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 **Continue**. Review the generated output, then click **Apply Policy**.

- If you are using a custom policy, enter a name for the policy under **Policy Name** and write the policy or paste the policy document from your text editor into the **Policy Document** box. When you are finished, click **Apply Policy**.



IAM applies the policy to the group.

Note
There are limitations on policy names and on policy size. For information about policy limitations, see Limitations on IAM Entities (p. 16).

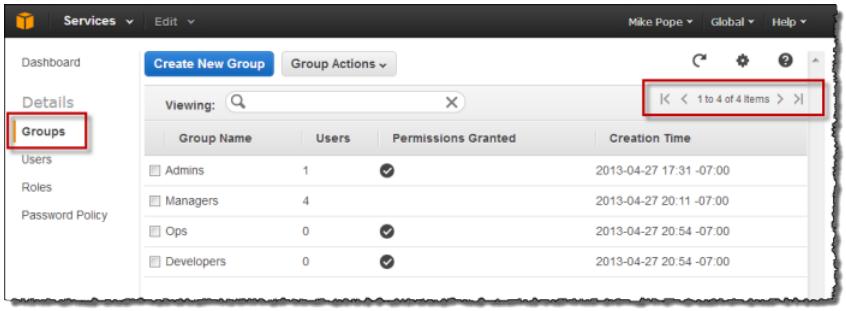
Listing Groups

You can list the groups that a particular user is in, get a list of the groups in the AWS account, or if you are using the IAM API or CLI, you can get a list of the groups with a certain path prefix. How you perform these tasks depends on the interface you use to access IAM.

AWS Management Console

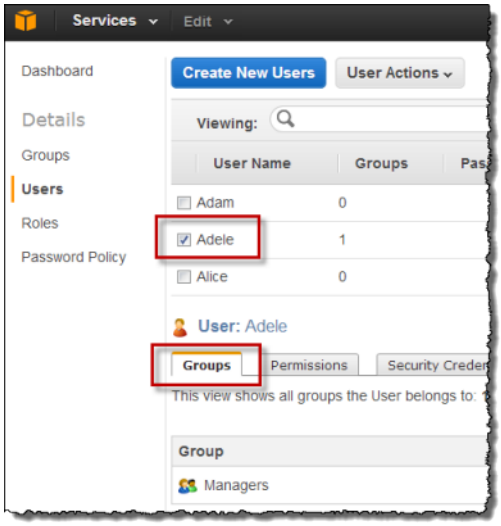
To list all groups in your AWS account

- Click **Groups**. If you have more than 50 groups, you can use the page controls on the right side of the console to advance to the next screen.



To list all the groups a user belongs to

1. In the navigation pane, click **Users**, and then select the user name.
2. To view the groups the user belongs to, select the **Groups** tab.



Command Line Interface

The `iam-grouplistbypath` command lets you list all the groups in the AWS account, or list all the groups with a particular path prefix. To get a list of each of the users in a particular group, use the `iam-grouplistusers` command. The output lists the ARN for each resulting group. For more information about ARNs and paths, see IAM Identifiers (p. 11).

For more information about `iam-grouplistbypath` or `iam-grouplistusers` commands, go to iam-grouplistbypath or iam-grouplistusers in the *AWS Identity and Access Management Command Line Interface Reference*.

API

The `ListGroups` action lets you list all the groups in the AWS account, or list all the groups with a particular path prefix. The response lists the path, ARN, and GUID for each resulting group. For more information about paths, ARNs, and GUIDs, see IAM Identifiers (p. 11).

If you have a large number of groups, you might want to paginate the results. For an example of how to do that, see Listing Users (p. 49).

If you want to get a list of the groups a particular user is in, use `ListGroupsForUser`.

For more information about `ListGroups` and `ListGroupsForUser`, go to AWS Identity and Access Management API Reference, or refer to your SDK's documentation.

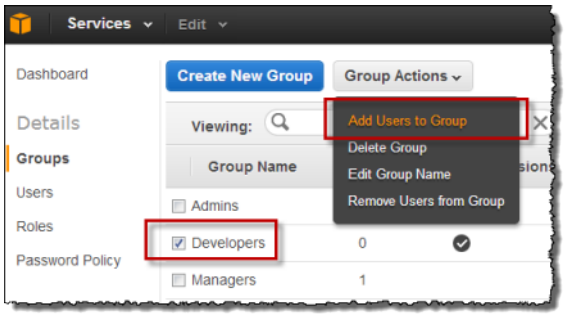
Adding Users to and Removing Users from a Group

You can use the AWS Management Console, the command line interface, or the API to add users to a group, or to remove them.

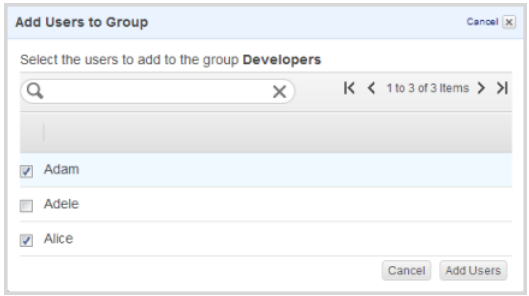
AWS Management Console

To add a user to a group

1. In the navigation pane, click **Groups**, and then select the group name.
2. From the **Group Actions** list, select **Add Users to Group**.

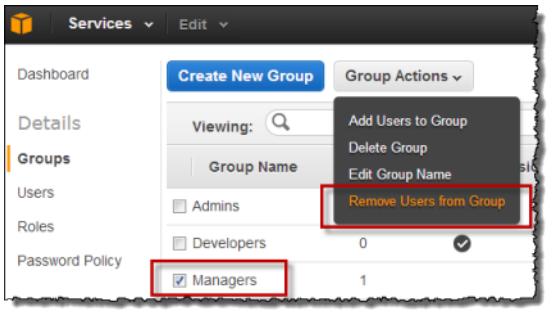


3. Select the users to add to the group, and then click **Add Users**.

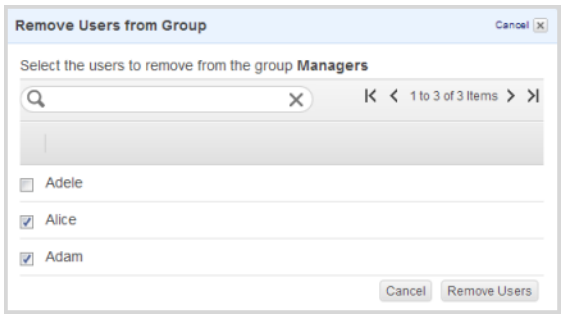


To remove a user from a group

1. In the navigation pane, click **Groups**, and then select the group name.
2. From the **Group Actions** list, select **Remove Users from Group**.



3. Select the users to remove from the group, and then click **Remove Users**.



Command Line Interface

To add a user to a group, use the `iam-groupadduser` command. Use the `iam-groupremoveuser` to remove a user from a group.

API

To add a user to a group, use the `AddUserToGroup` action. To remove a user from a group, use the `RemoveUserFromGroup` action. You can add or remove only a single user to or from a single group with a single API call. For more information about the actions, go to the AWS Identity and Access Management API Reference, or refer to your SDK's documentation.

The response doesn't include an updated list of the users in the group; it just contains the basic information about the group. If you want to get a list of users in the group, use `GetGroup`. For more information about listing users, see Listing Users (p. 49).

Deleting a Group

Topics

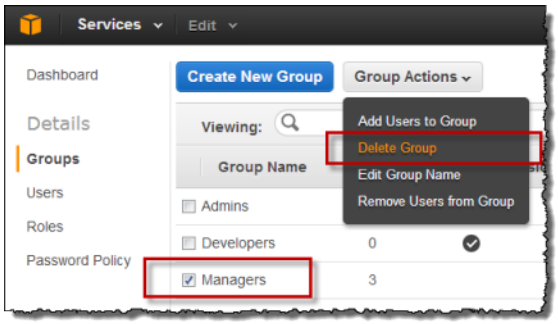
You might want to delete a group that you no longer need. When you use the AWS Management Console to delete a group, IAM deletes the group and any associated policies, but leaves the users intact. If you use the IAM CLI or API to remove the group, you must remove the users and policies before you can remove the group.

AWS Management Console

When you use the console to remove a group, IAM removes any policies associated with the group, and it removes the group. Users are removed from the group, and permissions the users had because they belonged to the group will no longer apply to them.

To use the AWS Management Console to delete a group

1. In the navigation pane, click **Groups**, and then select the group name.
2. From the **Group Actions** list, select **Delete Group**.



3. Review your changes, and then click **Yes, Delete**.

Command Line Interface or API

The following diagram and table describe the general process for deleting a group.

Process for Deleting a Group from Your AWS Account

1	Remove all users from the group.
2	Delete all policies attached to the group.
3	Delete the group.

How you actually execute the tasks in the preceding table depends on which interface you're using to access IAM. The interface-specific details are covered in the sections that follow.

Command Line Interface

If you're using the command line interface to access IAM, you can use a separate command for each of the tasks involved in deleting a group from your AWS account. Or, optionally, you can recursively delete the group and any attached policies by specifying an option with the command. The following table lists the commands to use. For more information about the commands, go to the AWS Identity and Access Management Command Line Interface Reference.

Process for Deleting a Group from Your AWS Account

1	Individually remove all users from the group: `iam-groupremoveuser`
2	Delete the policies attached to the group: `iam-groupdelpolicy`
3	Delete the group:`iam-groupdel` This function works only when the users have been removed from the group, and when the policies are no longer attached. If you want to delete the group and all attached policies without first removing the users and policies, use the `iam-groupdel -r` option. For more information about this command, go to the AWS Identity and Access Management Command Line Interface Reference.

API

If you're programmatically accessing IAM, you use a separate API call for each of the tasks involved in setting up a new user. The following table lists the API actions to use. Before deleting the group, you must delete all users from the group and remove any policies attached to it.

Process for Deleting a User from Your AWS Account

1	Remove all users from the group: `GetGroup` (to get the list of users in the group), and `RemoveUserFromGroup`
2	Delete all policies attached to the group: `ListGroupPolicies` (to get a list of the group's policies), and `DeleteGroupPolicy`
3	Delete the group:`DeleteGroup`

For more information about the actions, go to the AWS Identity and Access Management API Reference, or refer to your SDK's documentation.

Renaming Users and Groups

Topics

This section shows how to rename or change the path for a user or group.

Permission to Rename

Administrators in your AWS account are probably the only types of users who will have permission to rename users and groups (or change their paths). To understand why, think of changing the name or path of a user or group as a "move" operation. Whoever wants to rename the user or group needs to have permission to do it on both sides of the "move."

For example, let's say a user is changing from one division in the company to another. You need to change the user's path from /division_abc/ to /division_efg/. So, you need permission to remove the user from /division_abc/, and you need permission to put the user into /division_efg/. Effectively this means you need permission to call `UpdateUser` on both `arn:aws:iam::123456789012:user/division_abc/*` and `arn:aws:iam::123456789012:user/division_efg/*`. It's possible that the only people within the organization who have that type of permission are administrators.

Changing a User's Name or Path

You must use the IAM CLI or API to change a user's name. 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 GUID for the user remains the same (for more information about GUIDs, see User IDs (p. 15))

In addition, any policies that refer to the user *as the principal* (the user being granted access) are automatically updated to use the new name or path. For example, any queue-based policies in the Amazon SQS system that give the user access to a particular queue are automatically updated to use the new name and path. Amazon S3 bucket policies are also automatically updated.

However, we do 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, let's say Bob has a policy attached to him that lets him manage his security credentials. If an administrator renames Bob to Robert, the admin also needs to update that policy to change the resource from `arn:aws:iam::123456789012:user/division_abc/subdivision_xyz/Bob` to `arn:aws:iam::123456789012:user/division_abc/subdivision_xyz/Robert`. This is also true if the admin changes the path; the admin needs to update the policy to reflect the new path for the user.

Command Line Interface

The `iam-usermod` command lets you change the user's name or path. For more information about the command, go to the AWS Identity and Access Management Command Line Interface Reference.

API

The `UpdateUser` action lets you change the user's name or path. For more information about the action, go to the AWS Identity and Access Management API Reference, or refer to your SDK's documentation.

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.

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

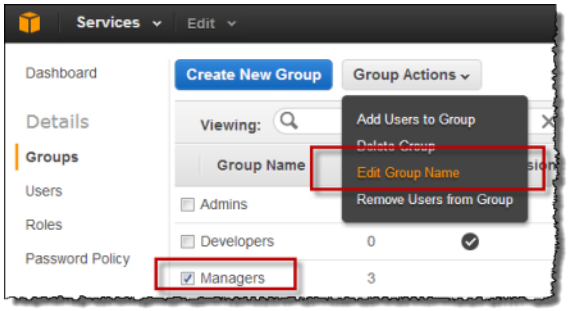
- The group retains all its users under the new name
- The GUID for the group remains the same (for more information about GUIDs, see User IDs (p. 15))

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 him that lets him use `UpdateGroup` specifically 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.

AWS Management Console

To change a group's name

1. In the navigation pane, click **Groups**, and then select the group name.
2. From the **Group Actions** list, select **Edit Group Name**.



3. Enter the new group name, and then click **Yes, Edit**.

Command Line Interface

The `iam-groupmod` command lets you change the group's name or path. For more information about the command, go to the AWS Identity and Access Management Command Line Interface Reference.

API

The `UpdateGroup` action lets you change the group's name or path. For more information about the action, go to the AWS Identity and Access Management API Reference, or refer to your SDK's documentation.

Managing Passwords

Topics

To access your AWS account resources from the AWS Management Console, an IAM user must have a password (formerly referred to as a *login profile*). This topic explains how to create, change, and delete a user's password. It also shows how to change the AWS account password, how you can enable users to change their own passwords, and how to set the password policy for users of your AWS account.

> **Note**
> Users who access your services only through the API or command line interface do not need a password.

For information about how users access your account sign-in page, see IAM and the AWS Management Console (p. 185).

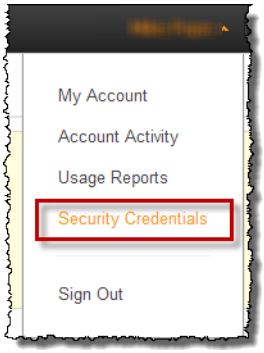
Changing Your AWS Account Password

To change the password for your AWS account

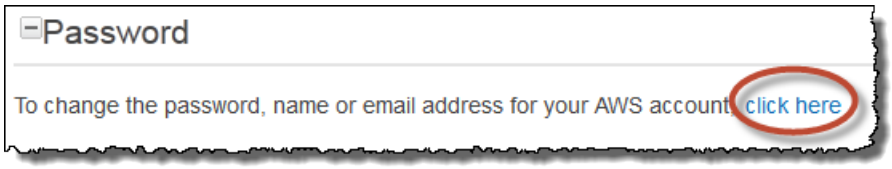
1. Use your AWS account email address and password to sign in to the AWS Management Console.

   > **Note**
   > If you previously signed in to the console with your IAM user credentials, your browser may remember this preference and open your account-specific sign-in page by default. You cannot use this page to sign in with your root credentials (your email address and password). In this case, to access the regular AWS sign-in page, click **Sign in using AWS Account credentials** near the bottom of the 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**.



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

Managing an IAM Password Policy

Topics

You can set a password policy for the passwords used by IAM users. Your policy can specify that passwords must be of a certain length, must include a selection of characters, and so on.

In addition, you can let users manage passwords and credentials in these ways:

- You can let all users change their own passwords. You can do as part of managing the password policy in the IAM console.
- You can create an IAM user or group that has permissions (via a policy) to manage password policies, passwords, credentials, and security certificates for other users.
- You can create a policy that you can apply to an IAM group that allows users in that group to manage their own (but no one else's) passwords, credentials, and certificates.

When you create or change a password policy, the change is enforced immediately when users change their passwords. IAM will not force users to change pre-existing passwords.

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. 76).

Managing a Password Policy (AWS Management Console)

Topics

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

For general information about password policies and how they work, see the overview topic Managing an IAM Password Policy (p. 65).

Displaying, Creating, or Changing the Password Policy

To display, create, or change a password policy

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. To display your current password policy, in the navigation pane of the console, click **Password Policy**. If you do not have a password policy, you will see a message indicating that a password policy hasn't been set for your AWS account.

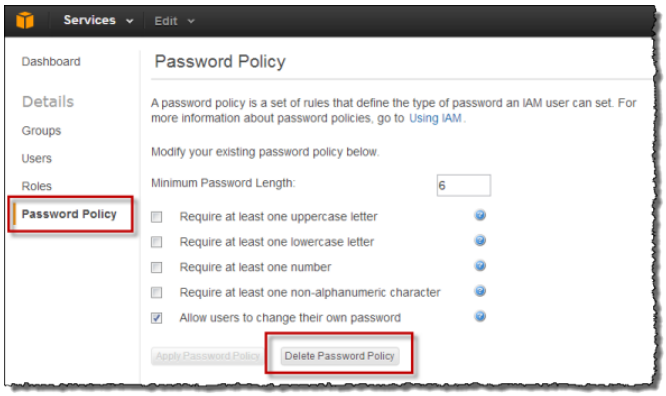
3. To create a new password policy or to change your existing policy, select the check box for the rules you want to apply to your password policy. You can specify the minimum length for the password, and you can require that passwords contain at least one of certain types of characters. If you require that passwords contain non-alphanumeric characters, users must use at least one of the following characters: ! @ # $ % ^ & * ( ) _ + - = [ ] { } | '.
4. To enable all of your IAM users to change their own passwords, make sure **Allow users to change their own password** is checked. When this option is checked, all IAM users in your account can use the IAM console to change their own passwords, as described in see Creating, Changing, or Deleting a Password for an IAM User (p. 70). (They can change their passwords, but they cannot manage their own security credentials or certificates.)

   Alternatively, you can let only some users manage passwords either for themselves or for others. In that case, clear the **Allow users to change their own password** option. For more information about setting policies to limit who can manage credentials, see Granting IAM Users Permission to Change Their Own Password (p. 74).
5. Click **Apply Password Policy**.

Deleting a Password Policy

To delete a password policy

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, click **Password Policy**, and then click **Delete Password Policy**.



Granting IAM Users Permission to Manage Password Policy and Credentials

You can use IAM policies to limit which users can manage passwords and other credentials. There are two scenarios:

- You can create an IAM policy that lets an IAM user or group manage passwords, credentials, and certificates for all IAM users, including themselves. You might do this if you have an administrator user or group and don't want to just let all users manage their own passwords.
- You can create an IAM policy that lets the current user manage his or her own password, credentials, and certificate. An efficient way to do this is to create a policy that includes a policy variable for the current user name. You can then attach the policy to an IAM group that contains the users who should be allowed to manage the credentials.

Using a policy to allow users to manage passwords and credentials gives you fine-grained control over who is allowed to manage these resources. (In contrast, if you enable the **Allow users to change their own password** option when you set password policy, *all* users can manage their own passwords.) Using a policy also lets you specify permissions individually for setting password policy, managing passwords, managing security credentials (access key ID and secret access key), and managing server certificates. Each of these tasks involves a different IAM action, and you can allow or deny access to the actions individually.

Allowing an IAM User or Group to Manage Password Policy and Credentials for All IAM Users

To grant a user or group permission to manage the password policy on your AWS account, you attach the following the following IAM policy to the user or group. This policy allows users to use the AWS Management Console to change the password policy. It also enables users to call the corresponding API actions directly, and to use the corresponding CLI commands.

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

```
  ]
}
```

To allow a user or group to manage credentials (access keys) and certificates for all IAM users, attach the following policy to the user or group. This policy includes permissions that are required in order to use the IAM console.

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iam:*AccessKey*",
        "iam:*SigningCertificate*",
        "iam:ListUsers",
        "iam:GetUser",
        "iam:GetLoginProfile",
        "iam:ListGroupsForUser",
        "iam:ListAccessKeys",
        "iam:ListMFADevices",
        "iam:ListUserPolicies",
        "iam:GetUserPolicy",
        "iam:ListSigningCertificates",
        "iam:ListGroupPolicies",
        "iam:GetGroupPolicy"
      ],
      "Resource": "*"
    }
  ]
}
```

For information about attaching policies to users or groups, see Managing IAM Policies (p. 112).

Allowing IAM Users to Manage Their Own Password and Credentials

The following policy lets the current user use API actions and the CLI to manage their own security credentials and signing certificates. (It doesn't allow them to use the IAM console.) The policy includes a policy variable (`aws:username`), so you can attach the policy to an IAM group but it still limits each user to managing only his or her own access key and certificate. Note that the `Version` element is set to `2012-10-17`, because that's the version of IAM policies that supports policy variables.

```
{
  "Version": "2012-10-17",
  "Statement": [{
      "Effect": "Allow",
      "Action":["iam:*AccessKey*","iam:*SigningCertificate*"],
      "Resource":["arn:aws:iam::123456789012:user/${aws:username}"]
    }
  ]
}
```

The following policy lets users manage their own access keys and signing certificates in the IAM console and by using API actions and the CLI. This policy includes permissions that are required in order to use the IAM console.You must replace *`group-name`* with the name of the IAM group that you're attaching the policy to.

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
   "Action": ["iam:*AccessKey*",
      "iam:*SigningCertificate*",
      "iam:ListUserPolicies",
      "iam:ListGroupPolicies"],
   "Effect": "Allow",
   "Resource": ["arn:aws:iam::123456789012:user/${aws:username}"]
  },
  {
    "Action": ["iam:ListUsers",
          "iam:GetUser",
          "iam:GetLoginProfile",
          "iam:ListGroupsForUser",
          "iam:ListAccessKeys",
          "iam:ListMFADevices",
          "iam:ListUserPolicies",
          "iam:GetUserPolicy",
          "iam:ListSigningCertificates"],
    "Effect": "Allow",
    "Resource": ["arn:aws:iam::123456789012:user/*"]
  },
  {
    "Action": ["iam:ListGroupPolicies","iam:GetGroupPolicy"],
    "Effect": "Allow",
    "Resource": ["arn:aws:iam::123456789012:group/group-name"]
  }
  ]
}
```

Displaying, Creating, Changing, or Deleting a Password Policy (API and CLI)

You can use the IAM API or CLI to manage the password policy for your AWS account.

Using the API to Create, Change, Display, or Delete a Password Policy

Action	Use
`UpdateAccountPasswordPolicy`	Create or change a password policy.
`GetAccountPasswordPolicy`	Display the current password policy.
`DeleteAccountPasswordPolicy`	Delete a password policy.

For more information about the actions, go to the AWS Identity and Access Management API Reference.

Using the CLI to Create, Change, Display, or Delete an IAM User Password

Command	Use
`iam-accountmodpasswordpolicy`	Create or change a password policy.

Command	Use
`iam-accountgetpasswordpolicy`	Display a password policy.
`iam-accountdelpasswordpolicy`	Delete a password policy.

For more information about the commands, go to the AWS Identity and Access Management Command Line Interface Reference.

Creating, Changing, or Deleting a Password for an IAM User

Topics

IAM users who access 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 a user under your AWS account. To complete these tasks, you can use the AWS Management Console, the IAM API, or the command line interface.

After you have created passwords for your users, you can grant them permission to change their own passwords. For more information, see Granting IAM Users Permission to Change Their Own Password (p. 74). For information about how users access your account sign-in page, see IAM and the AWS Management Console (p. 185).

> **Note**
> Users who access your services only through the API or command line interface do not need a password.

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

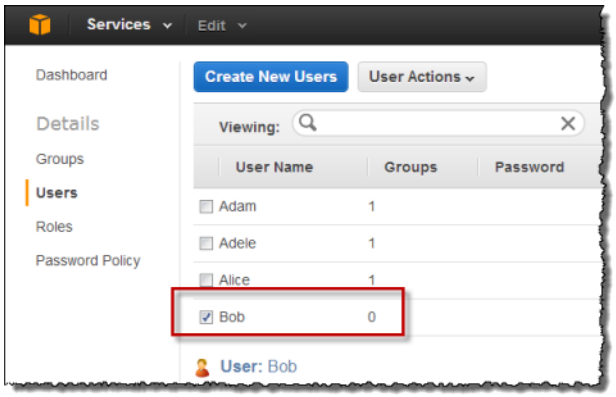
Topics

You can use the AWS Management Console to create, change, or delete a password for an IAM user.

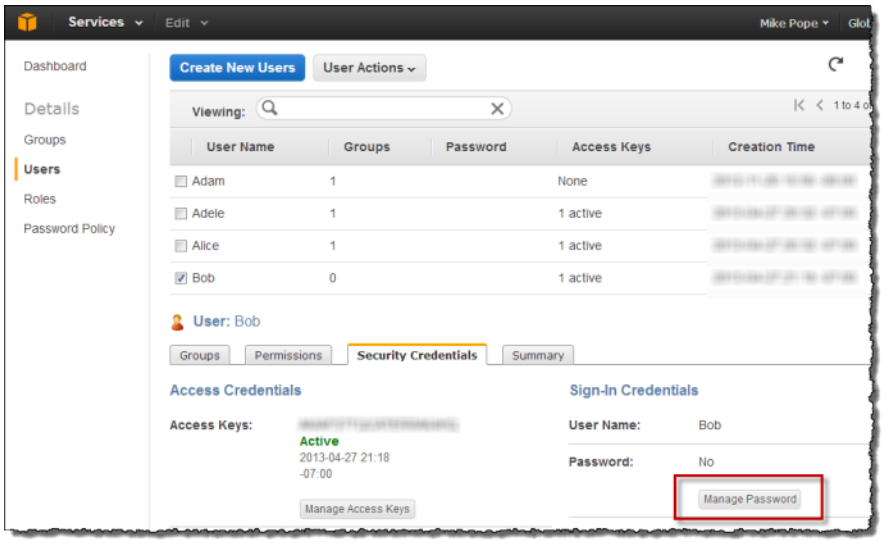
Creating or Changing an IAM User Password

To create or change a password for a user

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, click **Users**. In the **Users** pane, click the user whose password you want to create or change.



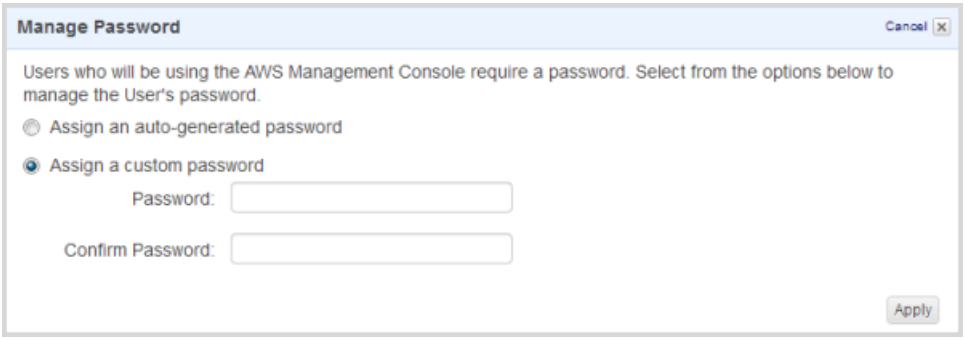
3. In the user details pane, click the **Security Credentials** tab, and then click **Manage Password**.



4. You can create a custom password, or you can have IAM automatically generate a password.

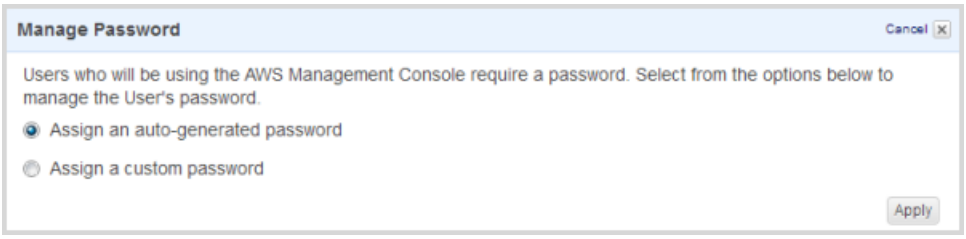
   Important
   When you create the password, you should save it in a safe place. For security reasons, neither you nor the user will be able to retrieve the password after this step.

   - To create a custom password, follow these steps:
     a. In the **Manage Password** dialog box, click **Assign a custom password** (or **Replace existing password with a new custom password** if this is a replacement password). In the boxes provided, enter and confirm the password, and then click **Apply**.

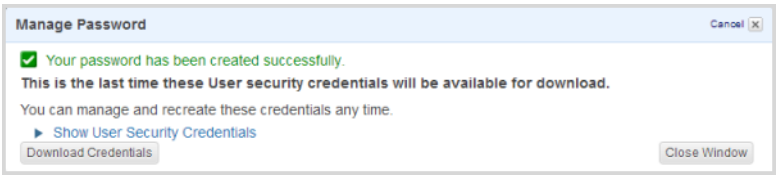


The password must conform to your password policy, if you have one configured. For more information about password policies, see Managing an IAM Password Policy (p. 65).

- To have IAM generate a password, follow these steps:
  a. In the **Manage Password** dialog box, click **Assign an auto-generated password** (or **Replace existing password with new auto-generated password** if this is a replacement password), and then click **Apply**.



  b. Click **Download Credentials** to save the password as a .CSV file to your computer.



After you create the password, you must provide the user with the following information so the user can sign in from your account sign-in page. IAM does not send this information to the user.

- The user's name and password
- The URL to your account sign-in page

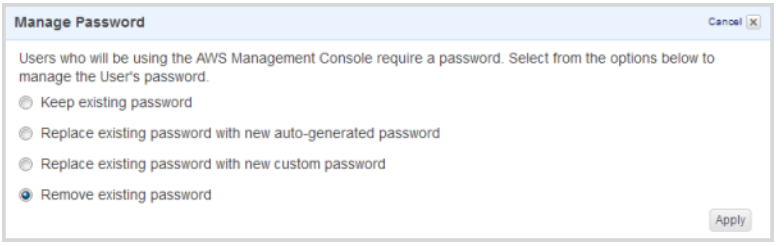
For information on accessing the account sign-in page, see IAM and the AWS Management Console (p. 185).

Note
Even if your users have a password, they still need permission to access your AWS resources. By default, a new 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 Users and Groups (p. 42). For information about using policies to set permissions, see Permissions and Policies (p. 107).

Deleting an IAM User Password

To delete a user's password

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, click **Users**. In the **Users** pane, click the user whose password you want to delete, and then click **Manage Password**.
3. Click **Remove existing password**, and then click **Apply**.



Note
Without a password, a user cannot sign in to AWS website features such as the AWS Management Console or the AWS forums.

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

Topics

You can use the IAM API or CLI to manage user passwords. These topics show the API actions and CLI commands you use to complete these tasks.

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 a User from Your AWS Account (p. 51).

Using the API to Create, Change, or Delete an IAM User Password

Action	Use
`CreateLoginProfile`	Create a password for a user.
`UpdateLoginProfile`	Change a user's password.
`DeleteLoginProfile`	Delete a user's password.

For more information about the actions, go to the AWS Identity and Access Management API Reference.

Using the CLI to Create, Change, or Delete an IAM User Password

Command	Use
`iam-useraddloginprofile`	Create a password for a user.
`iam-usermodloginprofile`	Change a user's password.
`iam-userdelloginprofile`	Delete a user's password.

For more information about the commands, go to the AWS Identity and Access Management Command Line Interface Reference.

Granting IAM Users Permission to Change Their Own Password

When you set a password policy, as part of that process, you can also set an account-wide policy that grants permission to all of your IAM users to change their own password. If you prefer to allow only some users to change their password, you can create a group for your users and then use an IAM policy to grant permission to the group. This topic describes how to use an IAM policy to grant permission to a group of users to change their own password. For information about enabling all of your users to change their own password when setting the account-wide password policy, see Managing an IAM Password Policy (p. 65).

Important
Setting a password policy is recommended because it enforces creation of strong passwords by your users.

To grant a group of users permission to change their password

1. Create a group and add the users whom you want to grant permission to. For more information, see Creating and Listing Groups (p. 54).
2. Create an IAM policy that grants users permission to change their password and apply that policy to the new group.

   Your policy might look like the following policy. This policy enables users to call `ChangePassword` directly, to use the corresponding `iam-userchangepassword` CLI command, and to use the AWS Management Console to change their password themselves. This policy also allows users to call the `GetAccountPasswordPolicy` action, which enables users to view the applicable password policy.

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

   For information about attaching a policy to a group, see Managing IAM Policies (p. 112).
3. Next, create a new password policy or modify your existing password policy, and make sure that the **Allow users to change their own password** check box is *not* checked. For more information about creating or viewing a password policy, see Managing an IAM Password Policy (p. 65).

Note
You can enable users to change their password without setting a password policy, but it is not recommended. To enforce use of strong passwords by your users, you should use a password policy when you enable users to manage their own password.

Related topics

- Managing an IAM Password Policy (p. 65)
- Managing IAM Policies (p. 112)

- How IAM Users Change Their Own Password (p. 75)
- Creating, Changing, or Deleting a Password for an IAM User (p. 70)

How IAM Users Change Their Own Password

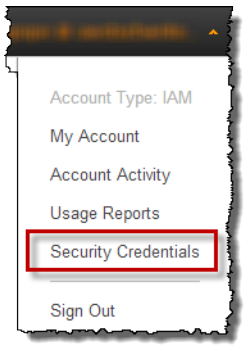
To change their password, users can use the AWS Management Console, the IAM API, or the command line interface. This topic describes how a user uses the AWS Management Console to change a password. The IAM APIs and CLI commands your users need to change their passwords are listed at the end of this topic.

The following procedure shows how users change their own passwords. You should provide this information to your users after you grant them permission to change their passwords. For information about how to grant users permission to change their passwords, see the previous topic, Granting IAM Users Permission to Change Their Own Password (p. 74).

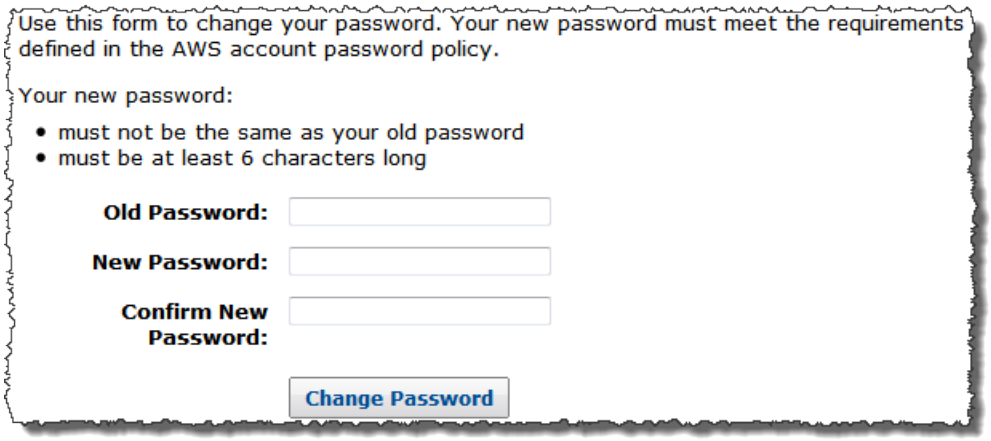
To change your IAM user password

1. Use your user name and password to sign in to your company's AWS Management Console sign-in page.

   To obtain the URL for your company's sign-in page, contact your administrator. For more information about using a sign-in page to access the AWS Management Console, see The AWS Management Console Sign-in Page (p. 185).
2. In the upper right corner of the console, click the arrow next to your user name, and then click **Security Credentials**.



3. On the **My Password** pane, in the **Old Password** box, type your password.



4. In the **New Password** box, type your new password. In the **Confirm New Password** box, retype your new password, and then click **Change Password**.

   Note
   The requirements for your password, such as minimum length and whether your password must contain a number, are defined by the password policy settings selected by your administrator.

Related API Actions and CLI Commands

To change their own password, users can use the `ChangePassword` API action or the `iam-userchangepassword` CLI command. For more information, see ChangePassword in the *AWS Identity and Access Management API Reference*, or iam-userchangepassword in the *AWS Identity and Access Management Command Line Interface Reference*.

Using Multi-Factor Authentication (MFA) Devices with AWS

Topics

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.

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. 76).

To set up MFA-protected API access for IAM users with an enabled MFA device, see Configuring MFA-Protected API Access (p. 89).

Setting Up an MFA Device

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

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 can be a smartphone that has an MFA application installed on it.

   If you want to use a hardware device, you can find information about where to purchase the devices that AWS supports at http://aws.amazon.com/mfa. If you want to use a virtual MFA device, or if you just want to learn more about virtual MFA, see Using a Virtual MFA Device with AWS (p. 78).
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. 78) or Enabling a Hardware MFA Device for Use with AWS (p. 84).
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. For access to MFA-protected APIs, you need access keys and an MFA code.

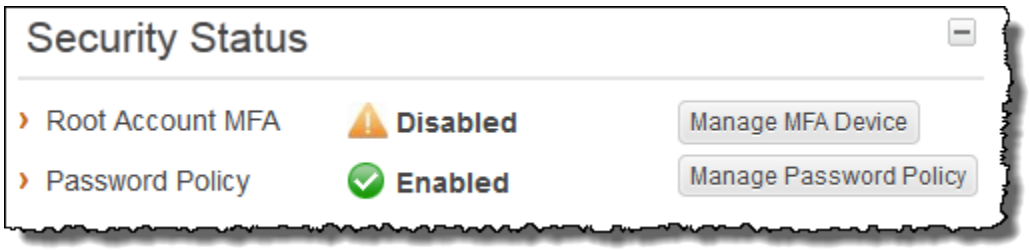
   For information about user passwords, see Managing Passwords (p. 63). For information about using MFA with the AWS Management Console, see MFA Devices and Your IAM-Enabled Sign-in Page (p. 190). For information about the AWS service APIs that use MFA, go to Does AWS MFA affect how I access AWS Service APIs? on the AWS Multi-Factor Authentication FAQs page.

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. Open the **IAM** console at https://console.aws.amazon.com/iam.
2. In the **IAM Dashboard**, check under **Security Status** to see whether MFA is enabled or disabled.



3. Click **Manage MFA Device** to change the current setting.

To check the MFA status of an IAM user

1. Open the **IAM** console at https://console.aws.amazon.com/iam.
2. Select **Users**.
3. Select the user.

   In the **Security Credentials** tab of the user's details pane, 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 a value for a virtual device, such as `arn:aws:iam::123456789012:mfa/user`, or it is the device serial number for a hardware device (usually the number from the back of the device), such as `GAHT12345678`.

4. Click **Manage MFA Device** to change the current setting.

   For virtual device information, see Using a Virtual MFA Device with AWS (p. 78).

   For hardware device information, see Enabling a Hardware MFA Device for Use with AWS (p. 84).

Using a Virtual MFA Device with AWS

Topics

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 any mobile hardware device, including a smartphone. 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.

> **Tip**
> The AWS Virtual MFA application is one example of an TOTP-compatible virtual MFA device. The AWS Virtual MFA application runs on the Android mobile operating system and is available for download from the Amazon Appstore for Android. For more information about downloading and installing the AWS Virtual MFA, see Installing the AWS Virtual MFA Mobile Application (p. 84).

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 Using the IAM CLI or API with Virtual MFA Devices (p. 83).

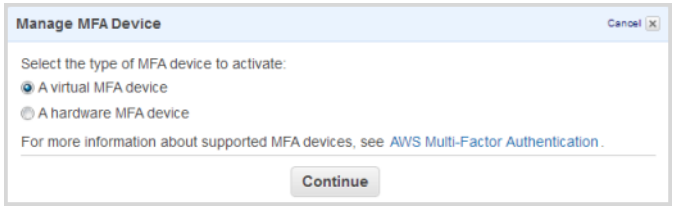
Note
You must have physical access to the user's MFA device in order to configure MFA. For example, if you are configuring MFA for a user who will use a smartphone to generate an OTP, you must have the smartphone available in order to finish the wizard.

The following procedure uses the AWS Virtual MFA mobile application as an example. However, the same general steps apply regardless of the virtual MFA application you use.

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.aws.amazon.com/iam/.
2. In the navigation pane, click **User** and then select the user you want to enable the virtual MFA for.
3. In the user details pane, select **Security Credentials**, and then click **Manage MFA Device**. This starts the **Manage MFA Device** wizard.

4. In the wizard, select **A virtual MFA device** and then click **Continue**.



5. Confirm that a virtual MFA application is installed on the user's mobile device and then click **Continue**. 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. 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, in the AWS Virtual MFA application, tap **Scan account barcode**, and then use the device's camera to scan the code. If you are using a different application on the device, use the equivalent command.

- If you cannot scan the code, enter the configuration information manually by typing the **Secret Configuration Key** value into the application. 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.

7. 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.

8. Click **Activate Virtual 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. 190).

Configuring and Enabling a Virtual MFA Device for Your AWS Account

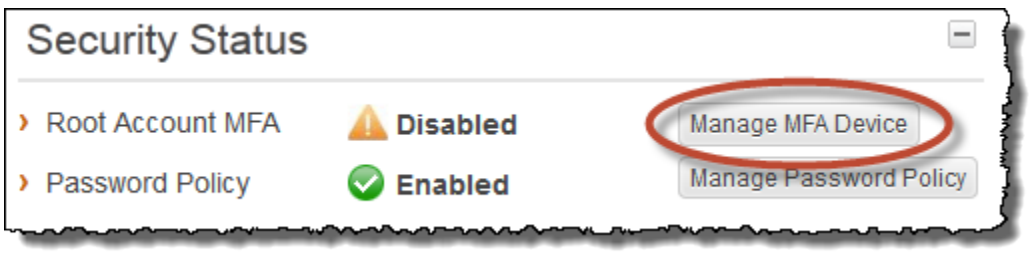
You can use IAM in the AWS Management Console to configure and enable a virtual MFA device for your root account. This section describes how to use the console to complete these tasks. If you prefer to use the CLI or API, see Using the IAM CLI or API with Virtual MFA Devices (p. 83).

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.

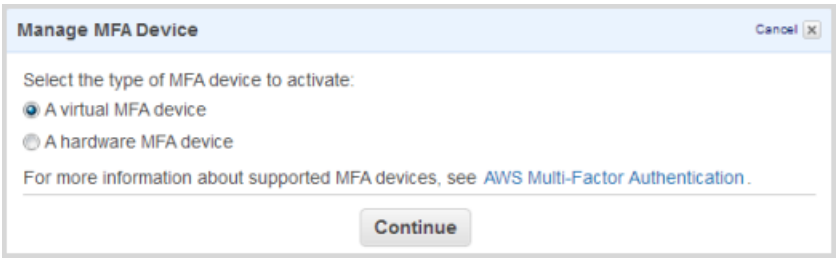
The following procedure uses the AWS Virtual MFA mobile application as an example. However, the same general steps apply regardless of the virtual MFA application you use.

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

1. Use your root credentials to sign in to the AWS Management Console, and then go to the **IAM** console.
2. On the **IAM Dashboard**, click **Manage MFA Device**. This starts the **Manage MFA Device** wizard.



3. In the wizard, select **A virtual MFA device** and then click **Continue**.



4. Confirm that a virtual MFA application is installed on the device, then click **Continue**. IAM generates and displays configuration information for the virtual MFA device, including a QR code similar to the following graphic.

5. 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, in the AWS Virtual MFA application, tap **Scan account barcode**, and then use the device's camera to scan the code. If you are using a different application on the device, use the equivalent command.

- If you cannot scan the code, you can enter the configuration information manually by typing the **Secret Configuration Key** value into the application. To do this in the AWS Virtual MFA application, tap **Manually add account**, 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.

6. In the **IAM Manage MFA Device** wizard, click the link to enable the device.
7. The **Enable AWS Multi-Factor Authentication** page opens on the AWS portal. (You might be prompted to sign in again before you can access this page.) In the **Authentication Code 1** box, type the six-digit number displayed by the MFA device. Wait 30 seconds while the device refreshes, and then type the second number into **Authentication Code 2**.
8. Click **Activate Authentication Device**.

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. 190).

Using the IAM CLI or API with Virtual MFA Devices

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

1. Task: Create the configuration information for the virtual MFA device.

   CLI command: iam-virtualmfadevicecreate

   API action: CreateVirtualMFADevice

2. Task: Enable the device for use with AWS.

   CLI command: iam-userenablemfadevice

   API action: EnableMFADevice

The following list shows additional commands and actions related to virtual MFA.

- Task: Deactivate a device.

  CLI command: iam-userdeactivatemfadevice

  API action: DeactivateMFADevice
- Task: List virtual MFA devices.

  CLI command: iam-virtualmfadevicelist

  API action: ListVirtualMFADevices
- Task: Re-sync an MFA device.

  CLI command: iam-userresyncmfadevice

  API action: ResyncMFADevice
- Task: Delete a virtual MFA device.

  CLI command: iam-virtualmfadevicedel

  API action: 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. 78).

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 Root Account (p. 86).

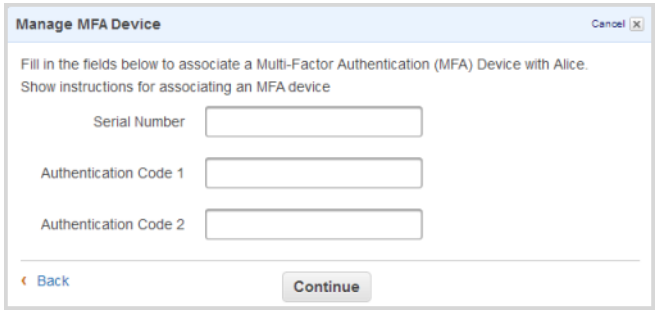
Note
If you want to enable the device from the command line, use `iam-userenablemfadevice`, described in the AWS Identity and Access Management Command Line Interface Reference. To enable the MFA device with the IAM API, use the `EnableMFADevice` action, described in the AWS Identity and Access Management API Reference.

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.aws.amazon.com/iam/.
2. On the Navigation pane, choose **Users**.
3. Select the user you want to enable an MFA device for, and then click **Manage MFA Device**.

4. Enter the device serial number. The serial number is usually on the back of the device.



5. 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.

6. 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.
7. 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. 190).

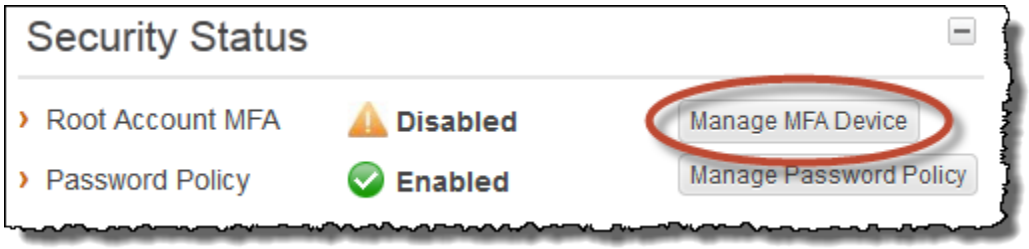
Enabling a Hardware MFA Device for Your AWS Root Account

To enable the MFA device for your AWS account

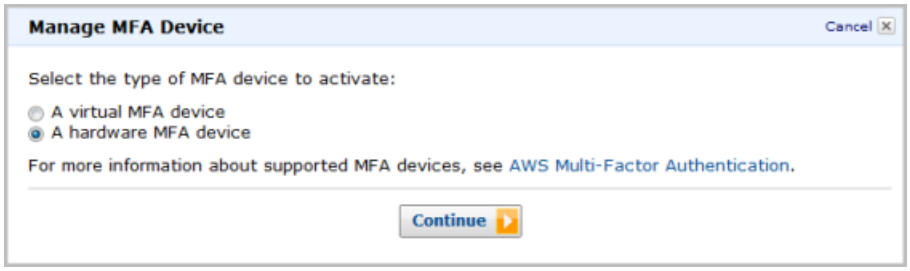
1. Use your root credentials to sign in to the AWS Management Console, then go to the **IAM** console.

   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. From the **IAM Dashboard**, click **Manage MFA Device**. This starts the **Manage MFA Device** wizard.



3. In the wizard, select **A hardware MFA device**, and then click **Continue**.



4. To complete the remainder of this process, you need to go to the AWS portal. To go to the portal, click **Click here to enable your device**.
5. In the **Serial Number** box, enter the serial number displayed on the back of the MFA device. Re-enter the serial number in the **Re-Enter Serial Number** box.

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 **Activate Authentication Device**. The MFA device is now associated with the AWS Account.
9. In the AWS Management Console, click **Close** to close the confirmation dialog box.

   The next time anyone signs in using the AWS account credentials, they 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. 190).

Synchronizing an MFA Device

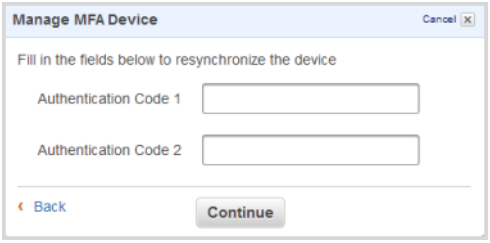
It's possible for an MFA device to get out of synchronization. If the device is not synchronized when the user tries to use it, the user's login will fail. If the user is using the MFA device to sign in to the AWS Management Console, IAM prompts the user to resynchronize the device.

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 `iam-userresyncmfadevice`.

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 AWS Management Console **IAM** console, click **Users**, then select the name of the user you want to synchronize a device for.
2. Click the **Security Credentials** tab, 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 **Update**.

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 `iam-userdeactivatemfadevice` command.

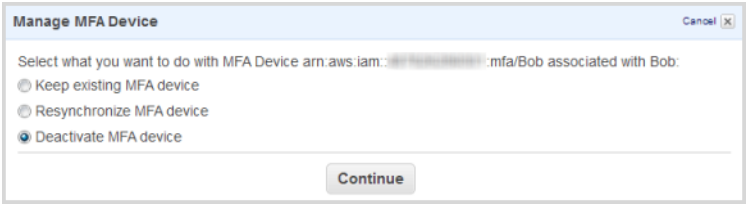
> **Note**
> If you use the API or CLI to delete a user from your AWS account, you must deactivate or delete the user's MFA device as part of the process of removing the user. For more information about deleting users, see Deleting a User from Your AWS Account (p. 51).

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. 88).

Deactivating a User's MFA Device

To deactivate a user's MFA device

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



4. Click **Update**.

The user's device is deactivated.

Deactivating Your AWS Account's MFA Device

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

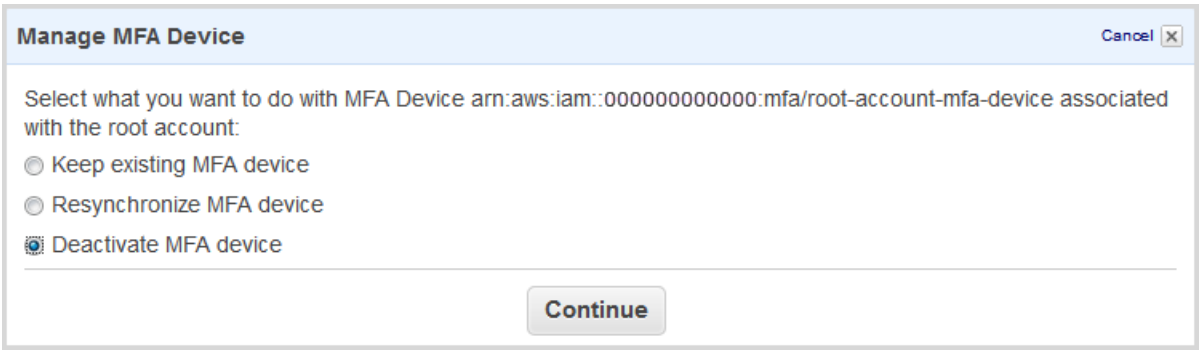
To deactivate the MFA device for your AWS account

1. Use your root credentials to sign in to the AWS Management Console, then go to the **IAM** console.

   > **Note**
   > 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.

2. From the **IAM Dashboard**, click **Manage MFA Device**.

3. Select **Deactivate this device**.



The MFA device is deactivated for the AWS account.

Configuring MFA-Protected API Access

MFA-protected API access is an optional feature that offers an extra layer of security by requiring users to authenticate with an MFA device before they can use APIs you specify in a policy. Users can authenticate through the AWS Management Console or programmatically through the AWS Security Token Service (STS).

MFA-protected API access is available only to services that support temporary security credentials. For a complete list of these services, and for information about using the AWS STS API for requesting temporary security credentials, 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 policy 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, either through the console, or by requesting new temporary security credentials using an MFA code and device serial number.

Note
MFA-protected API access cannot be applied to root accounts accessing their own resources. Federated users cannot be assigned an MFA device for use with AWS services, so they cannot access AWS resources controlled by MFA.

Using MFA-Protected APIs Through the Console

AWS evaluates MFA-protected API policies for actions in the console, such as terminating an Amazon EC2 instance. Set up the IAM user with an MFA device and enable an MFA-protected API policy. The user can then simply log into the console with MFA authentication and is subject to the policies for

MFA-protected APIs. For users who already have an assigned MFA device, the console experience doesn't change (except for optional time limits on certain MFA-protected APIs that require more frequent re-authentication). For more information on setting up an IAM user with an MFA device, see Setting Up an MFA Device (p. 76).

Using MFA-Protected APIs Programmatically

MFA-protected API policies also apply to IAM users who access MFA-protected API programmatically, either through their own application or an application provided to them, specifically:

1. An MFA-protected API policy is attached to a user, group, or resource.
2. The IAM user acquires temporary security credentials using an MFA code and device serial number, programmatically, in a request to the AWS Security Token Service (STS). The temporary security credentials include the MFA authentication status.

   An application can use the STS API and custom code to prompt the user to provide the MFA code and serial number in the application. For information on implementing MFA authentication using STS, see Creating Temporary Security Credentials in *Using Temporary Security Credentials*.
3. When the IAM user tries to use an API directly or through an application, AWS uses the condition in the MFA-protected API policy to check whether the IAM user is allowed access to the API.
4. AWS grants the IAM user access to the MFA-protected API (or prevents access) according to the policy.

Policies for MFA-Protected APIs

MFA-protected API policies are attached to a user, group, or resource, such as an Amazon Simple Storage Service (Amazon S3) bucket, Amazon Simple Queue Service (Amazon SQS) queue, or Amazon Simple Notification Service (Amazon SNS) topic).

MFA-protected API policies include a condition statement (or statements) with the `aws:MultiFactorAuthAge` key. The policy can use the key in two ways:

- Existence—To verify that MFA authentication is present, use an existence check with a `Null` statement to evaluate whether the `aws:MultiFactorAuthAge` key exists (is not null, matching the Boolean value "false") or not (is null, matching the Boolean value "true").
- Duration—For situations that require control over the duration of the MFA authentication, independent of the lifetime of the temporary security credentials, use a numeric condition type to compare the key's age to a value (such as 3600 seconds). If a policy checks for existence, only, the access is available for the entire authentication session (the default is 12 hours). A numeric condition enables the policy to restrict access to APIs for a user who does not have recent MFA authentication, even if the user's authorization session is still valid. If necessary, an IAM user can refresh their MFA authentication, either through the console, or by requesting new temporary security credentials using an MFA code and device serial number.

For more information on the condition types for `aws:MultiFactorAuthAge`, see Existence of Condition Keys (p. 138) and Numeric Conditions (p. 135).

The following policies demonstrate several use-cases that require MFA authentication.

Topics

Example 1: Granting access after MFA authentication

The following example shows a policy attached to a user or group that grants Amazon EC2 access only after valid MFA authentication. Specifically, AWS grants access only when the existence check for a `Null` value evaluates to `false`. In other words, the `aws:MultiFactorAuthAge` key value is not null.

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

Example 2: Granting access after recent MFA authentication

The following example shows a policy attached to a user or group that grants Amazon EC2 access only after checking for a valid MFA authentication within an hour of the request. Specifically, AWS grants access only when the `aws:MultiFactorAuthAge` key value is present and less than 3600 seconds (1 hour).

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

Example 3: Denying access to specific APIs without valid MFA authentication

The following example shows a policy attached to a user or group that grants access to the entire Amazon EC2 API, but restricts access to `StopInstances` and `TerminateInstances` until valid MFA authentication is present. Specifically, AWS denies access to `StopInstances` and `TerminateInstances` when the existence check for a `Null` value evaluates to `true`, meaning the `aws:MultiFactorAuthAge` key value is null.

```
{
  "Version":"2012-10-17",
  "Statement":[
    {
```

```
      "Action":["ec2:*"],
      "Effect":"Allow",
      "Resource":["*"]
    },
    {
      "Action":["ec2:StopInstances","ec2:TerminateInstances"],
      "Effect":"Deny",
      "Resource":["*"],
      "Condition":{
        "Null":{"aws:MultiFactorAuthAge":"true"}
      }
    }
  ]
}
```

Example 4: Denying access to specific APIs without recent valid MFA authentication

The following example shows a policy attached to a user or group that grants access to the entire Amazon EC2 API, but restricts access to `StopInstances` and `TerminateInstances` unless valid MFA authentication occurred within the last hour. Specifically, AWS denies access to `StopInstances` and `TerminateInstances` when the existence check for a `Null` value evaluates to `true`, meaning the `aws:MultiFactorAuthAge` key value is null. Additionally, AWS denies access to `StopInstances` and `TerminateInstances` when `aws:MultiFactorAuthAge` key value is present and greater than 3600 seconds (1 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":[
    {
      "Action":["ec2:*"],
      "Effect":"Allow",
      "Resource":["*"]
    },
    {
      "Action":["ec2:StopInstances","ec2:TerminateInstances"],
      "Effect":"Deny",
      "Resource":["*"],
      "Condition":{
        "Null":{"aws:MultiFactorAuthAge":"true"}
      }
    },
    {
      "Action":["ec2:StopInstances","ec2:TerminateInstances"],
      "Effect":"Deny",
      "Resource":["*"],
      "Condition":{
        "NumericGreaterThan":{"aws:MultiFactorAuthAge":"3600"}
      }
    }
```

```
  ]
}
```

Example 5: Granting access to a resource after MFA authentication

MFA-protected API policies can be attached to a resource, as well, such as Amazon S3 bucket policies. The following policy is attached to an Amazon S3 bucket. This policy grants access to the Amazon S3 actions `PutObject` and `DeleteObject` only after valid MFA authentication for all IAM users with access to the bucket. Specifically, AWS grants access only when the existence check for a `Null` value evaluates to `false`. In other words, the `aws:MultiFactorAuthAge` key value is not null.

```
{
  "Version":"2012-10-17",
  "Statement":[
    {
      "Effect":"Allow",
      "Principal":{
        "AWS": "111122223333"
      },
      "Action":["s3:PutObject","s3:DeleteObject"],
      "Resource": "arn:aws:s3:::myawsbucket/*",
      "Condition":{
        "Null":{"aws:MultiFactorAuthAge":"false"}
      }
    }
  ]
}
```

Note
Amazon S3 offers an MFA Delete feature for *root* account (only) access. 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.

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. 87).

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 you 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 **Sign-In Credentials** on 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. 88).

Managing User Keys and Certificates

Topics

This section describes how to create and manage access keys and signing certificates (also known as X.509 certificates) for IAM users. Each user needs access keys (an access key ID and a secret access key) to make programmatic calls to AWS using the command-line interface (CLI), the AWS SDKs, or direct HTTP calls using the APIs for individual services.

Some services also support the use of signing certificates. For example:

- Amazon EC2 originally supported the SOAP protocol for making service calls; SOAP-based calls use a signing certificate in order to digitally sign the requests. However, support for SOAP in Amazon EC2 is being deprecated, and we recommend that instead you use query requests. For more information, see Making API Requests in the Amazon Elastic Compute Cloud User Guide.
- The command-line interfaces (CLI) for some services support both access keys and certificates. In these cases, we recommend that you configure the CLI using access keys.

For more information about the credentials, see Security Credentials (p. 7) and Adding a New User to Your AWS Account (p. 42).

Each user can have two sets of active keys and two certificates for the purposes of credential rotation. For more information about the number of allowed for IAM entities, see Limitations on IAM Entities (p. 16).

Creating, Modifying, and Viewing User Security Credentials

This section shows how to create, modify, or view a user's security credentials. Security credentials consist of an Access Key ID and a Secret Access Key for a user. If you've performed the steps in Getting Started (p. 21), you've already created security credentials for a user.

By default, when you create a key, its status is `Active`, which means the user can use the key for API calls. At any point, you can disable the key, which means it can't be used for API calls. You might switch

the status of a key if you're rotating keys (for more information, see Rotating Credentials (p. 106)) or to revoke API access for a user.

When you create the key, IAM returns the Secret Access Key and an Access Key ID, which is a public identifier for the key. You need to save these keys and give them to the user.

> **Important**
> To ensure the security of your AWS account, the Secret Access Key is accessible only during key and user creation. You must save the key (for example, in a text file) if you want to be able to access it again. If a secret key is lost, you can delete the access keys for the associated user and then create new keys.

You can give your users permission to list and manage their own keys. For more information, see Adding a Credentials Management Policy for Users (p. 104).

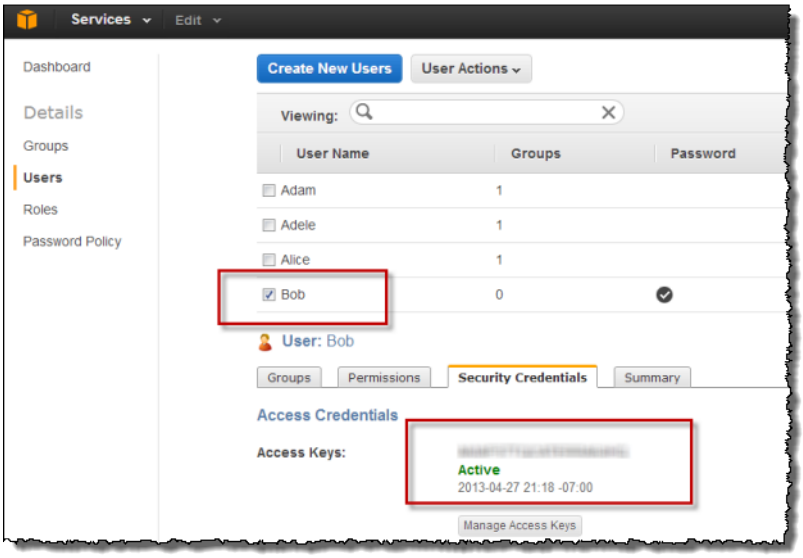
You can delete an access key at any time. However, when you delete an access key, it's gone forever and cannot be retrieved. (You can create new keys.)

The details of how you manage keys depends on whether you're using the IAM console, the command line interface (CLI), or API calls. The following sections provide instructions for each of these approaches.

AWS Management Console

To list a user's access keys

- In the navigation pane, click **Users**, and then select the user.

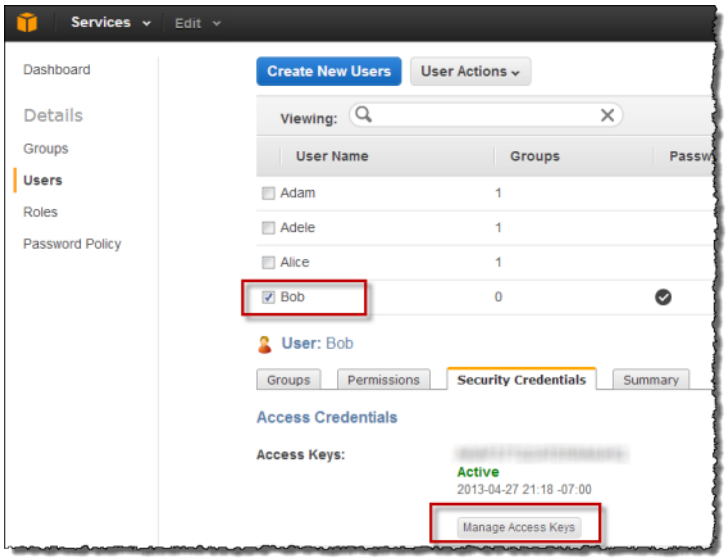


The user's access keys and the status of each key is displayed on the **Security Credentials** tab. Users cannot have more than two sets of access keys.

> **Important**
> The user's Access Key ID is visible. The Secret Access Key cannot be retrieved after you are finished creating it.

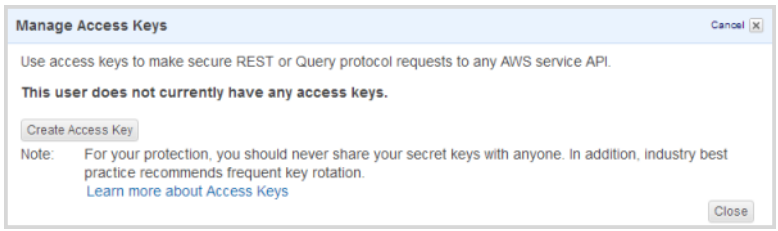
To create, modify, or delete a user's access keys

1. Select the user.

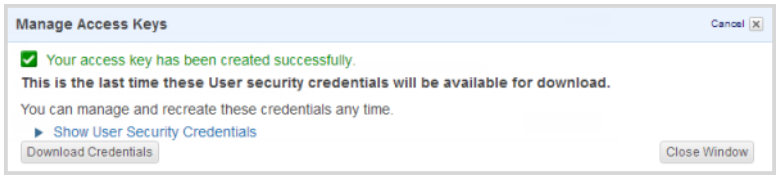


2. Click **Manage Access Keys**.
3. To create an access key, do this:

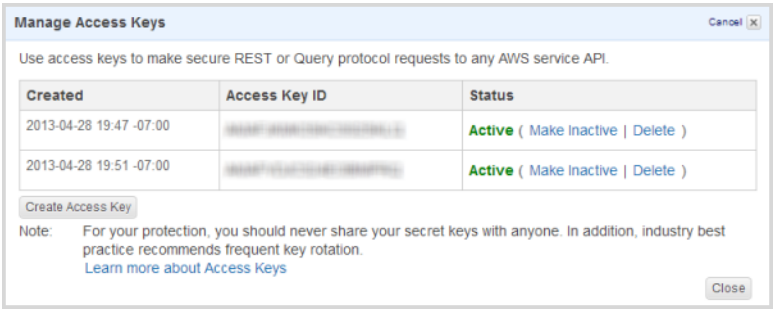
   i. Click **Create Access Key**.



   ii. Click **Download Credentials** to save the Access Key ID and Secret Access Key to a .csv file on your computer. 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.



4. To disable an access key, click **Make Inactive**. To re-enable the key, click **Make Active**.



5. To delete a user's access key, click **Delete**.
6. Click **Close Window**.

Command Line Tools

The following table lists the CLI commands for managing a user's access keys. For more information about the commands, see the AWS Identity and Access Management Command Line Interface Reference.

Task	Command to Use
Create an access key	`iam-useraddkey`
Disable or re-enable an access key	`iam-usermodkey`
List a user's access keys	`iam-userlistkeys`
Delete an access key	`iam-userdelkey`

API

The following table lists the actions for managing a user's access keys. For more information about the actions, go to the AWS Identity and Access Management API Reference.

Task	Action to Use
Create an access key	`CreateAccessKey`
Disable or re-enable an access key	`UpdateAccessKey`
List a user's access keys	`ListAccessKeys`
Delete an access key	`DeleteAccessKey`

Creating and Uploading a User Signing Certificate

If a user needs a signing certificate, you first must obtain a signing certificate and then upload it to the IAM system. IAM doesn't have an API action to create signing certificates, so you must use a third-party tool such as OpenSSL to create the user signing certificate.

Signing certificates are used in only some services. For example, in Amazon EC2 you can use a signing certificate for SOAP requests. However, Amazon EC2 is in the process of deprecating SOAP support, and recommends instead that you the Query API. For more information, see Making API Requests in the Amazon Elastic Compute Cloud User Guide.

In Amazon EC2, Auto Scaling, Elastic Load Balancing, and Amazon CloudWatch, you can use a certificate to configure credentials for the CLI. As an alternative to using a certificate, you can use access credentials (an access key ID and a secret access key) to configure the CLI. For more information, see the instructions in the documentation for individual services that describes how to configure the CLI.

> **Note**
> If you need credentials for connecting to an Amazon EC2 instance, see Connecting to Amazon EC2 Instances in the *Amazon Elastic Compute Cloud User Guide*.

To create a user certificate using OpenSSL, you do the following:

1. Install OpenSSL from the http://www.openssl.org/ site.
2. Configure OpenSSL for your operating system.
3. Create a private key.
4. Generate a certificate using the private key. Because you're creating a user signing certificate and not a server certificate, you don't need to submit a certificate signing request (CSR) to a Certificate Authority (CA). During this process, OpenSSL prompts you for values like your company name. For server certificates, these values are verified and signed by the CA and used by browsers to ensure that the domain is genuine. But for an AWS User certificate, these values are not used and you can leave them blank. The certificate that is generated must be in PEM format so you can copy the certificate contents into the console GUI to upload the user's signing certificate.
5. Upload the certificate.

> **Note**
> Although you can use the security credentials page in the AWS Management Console to create an X.509 certificate, that method is only for the AWS account credentials. You can't upload a certificate generated using the security credentials page for IAM users. Instead, you must use the process described in this section (that is, using a third-party tool like OpenSSL).

As with access keys, each certificate can have a status of either `Active` or `Inactive`. By default, the status is `Active` when you upload the certificate.

When you upload the certificate, it returns a certificate ID that you can save for your records. However, if necessary, you can list the IDs for the user's certificates. You can delete a certificate at any time.

You can give your users permission to list and manage their own certificates. For more information, see Adding a Credentials Management Policy for Users (p. 104).

Install and Configure OpenSSL

Creating and uploading a certificate requires a tool that supports the SSL and TLS protocols. OpenSSL is an open-source tool that provides the basic cryptographic functions necessary to create an RSA token and sign it with your private key. If you don't already have OpenSSL installed, follow these instructions.

To install OpenSSL on Linux and 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**.
4. After you have installed the Microsoft Visual C++ 2008 Redistributables, select the appropriate version of the OpenSSL binaries for your environment, save the file locally, and run it. The **OpenSSL Setup Wizard** launches.
5. Follow the instructions described in the **OpenSSL Setup Wizard**. Save the OpenSSL binaries to a folder in your working directory.

Before you use OpenSSL commands, you must configure the operating system so that it has information about the location of the OpenSSL install point.

To configure OpenSSL on Linux and 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 the OpenSSL installation:

```
export PATH=$PATH:$OpenSSL_HOME/bin
```

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 Path=OpenSSL_HOME\bin;%Path%
```

3. Set the path to the OpenSSL installation:

```
set Path=OpenSSL_HOME\bin;%Path%
```

> **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.

Create a Private Key

You need a unique private key that you use when generating the user signing certificate.

To create a private key

1. At the command line, use the `openssl genrsa` command with the following syntax:

```
openssl genrsa 1024 > private-key.pem
```

For *private-key.pem*, specify your own file name. In the example, `1024` represents 1024-bit encryption. AWS also supports 2048-bit and 4096-bit encryption. We recommend you create a 1024-bit or 2048-bit RSA key.

2. If you will be using the certificate to authenticate CLI commands for Auto Scaling, Amazon CloudWatch, or Elastic Load Balancing, generate the certificate in PKCS8 format using the following command:

```
openssl pkcs8 -topk8 -nocrypt -inform PEM -in private-key.pem -out private-key-in-PCKS8-format.pem
```

Create the User Signing Certificate

You can now create a user signing certificate.

To create a user signing certificate

- Use the `openssl req` command and the following syntax:

```
openssl req -new -x509 -nodes -sha1 -days 365 -key private-key.pem -outform
 PEM > certificate.pem
```

For *private-key.pem*, use the .pem file that you generated in a previous procedure. For *certificate.pem*, use the name of a file into which you want the certificate to be generated. The certificate must be in .pem format.

In this example, the `-days 365` switch specifies that the certificate is good for 365 days. For information about the other switches, enter `openssl req -h` at the command line.

OpenSSL displays a message similar to the following:

```
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.
```

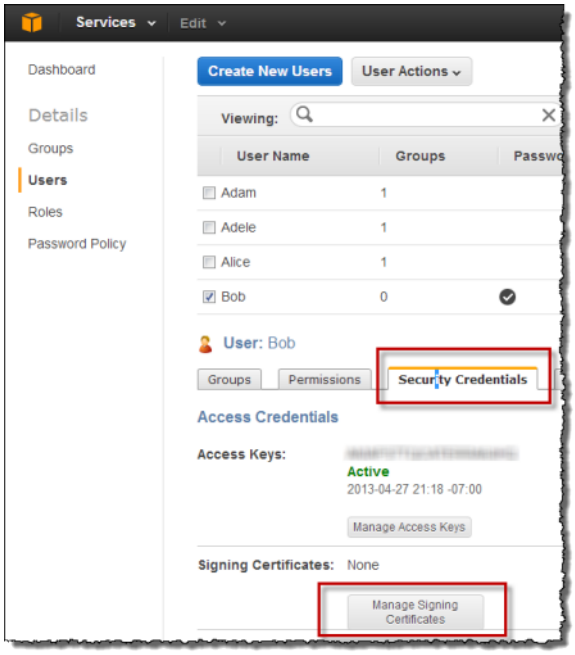
Because you're creating a user signing certificate (not a server certificate), you can leave all the values blank when you're prompted. These values are used by the Certificate Authority (CA) to help authenticate the server certificate. However, because user signing certificates are uploaded in an authenticated session, AWS does not need any information in the certificate for further validation, and requires only the public-private key pair.

The .pem file contains the certificate value that you can copy and paste during the upload procedure that follows.

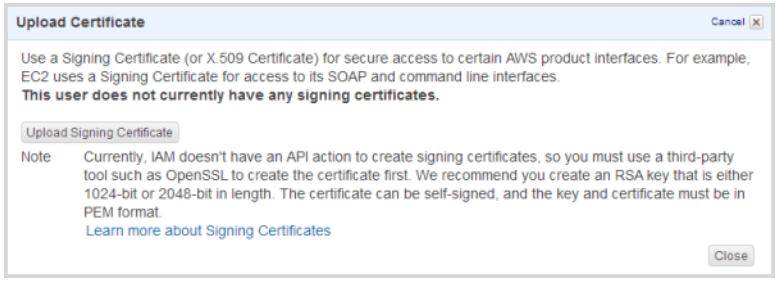
Managing the Certificate Using the AWS Management Console

To upload a signing certificate for a user

1. Select the user, click the **Security Credentials** tab, and then click **Manage Signing Certificates**.



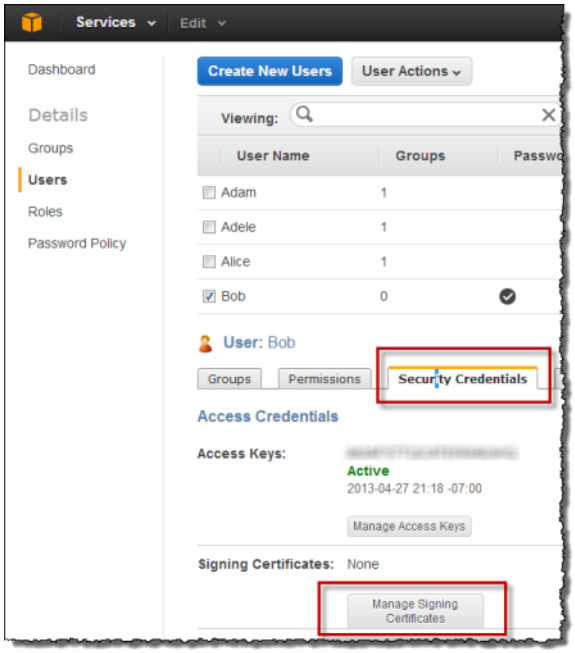
2. Click **Upload Signing Certificate**.



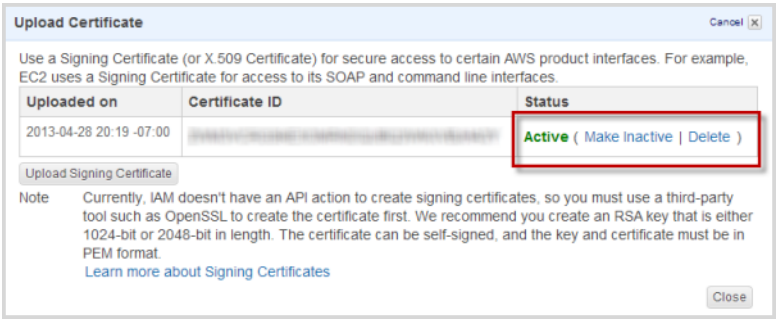
3. Using a text editor, open the .pem file that contains the user signing certificate that you created using OpenSSL, as described earlier in this section, and copy the contents of the .pem file.
4. Paste the contents of the user's signing certificate into the **Certificate Body** field.

To modify or delete a signing certificate for a user

1. Select the user.



2. Click **Manage Signing Certificates**.
3. To disable or re-enable a signing certificate, click **Make Inactive** or **Make Active**.

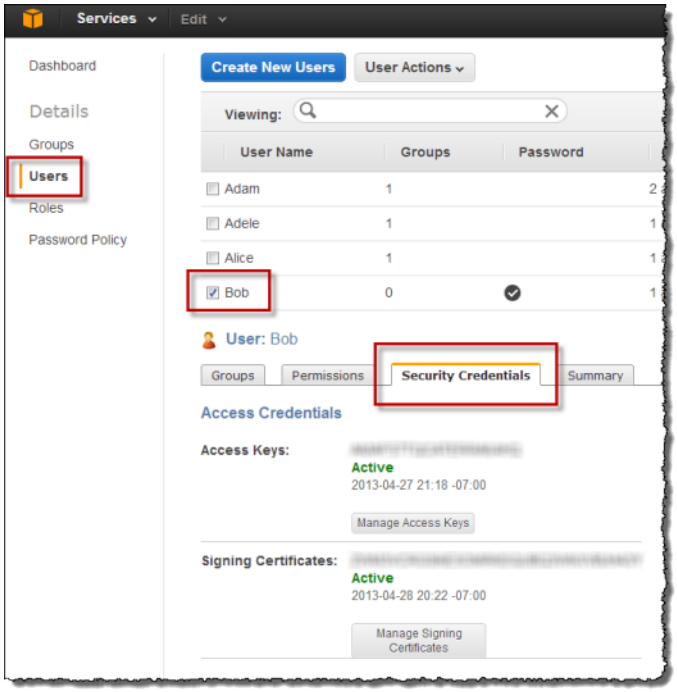


4. To delete a user's signing certificate, click **Delete**.
5. Click **Close**.

   Note
   After the certificate has been uploaded, it cannot be retrieved, its contents cannot be viewed, and it cannot be reused.

To view a list of signing certificates belonging to a user

- In the navigation pane, click **Users**, and then select the name of the user whose signing certificate you want to view.



Signing certificates and the status of certificate is displayed on the **Security Credentials** tab. Users cannot have more than two signing certificates.

Command Line Tools

The following table lists which command to use for each task you want to perform with a signing certificate. For more information about the commands, go to the AWS Identity and Access Management Command Line Interface Reference.

Task	Command to Use
Upload a certificate	`iam-useraddcert`
Disable or re-enable a certificate	`iam-usermodcert`
List a user's certificates	`iam-userlistcerts`
Delete a certificate	`iam-userdelcert`

API

The following table lists which actions to use for each task you want to perform with a signing certificate. For more information about the actions, go to the AWS Identity and Access Management API Reference, or refer to your SDK's documentation.

Task	Action to Use
Upload a certificate	`UploadSigningCertificate`
Disable or re-enable a certificate	`UpdateSigningCertificate`
List a user's certificates	`ListSigningCertificates`
Delete a certificate	`DeleteSigningCertificate`

Note
Use a POST request when uploading a signing certificate because of the certificate's size.

Adding a Credentials Management Policy for Users

This section shows how to add a policy that lets a user create and manage his or her own credentials. You can take two approaches:

- Create an individual policy for each user who should be allowed to manage his or her own credentials. You might do this if there are a very limited number of users who need these permissions.
- Create a group and add all users to the group who should have these permissions. You then attach a policy to the group that uses a policy variable to represent any user in that group. This is usually more practical if there are more than just a few users who need these permissions.

You could give the user access to only some of the actions related to the credentials (perhaps just listing them), or to a wider set if you want the user to be able to create and rotate the credentials.

Note
Users can have two sets of keys; for more information, see Limitations on IAM Entities (p. 16)). Typically, the only reason a user might have more than one active at a time is for the purpose of rotating credentials. For more information, see Rotating Credentials (p. 106).

Creating an Individual Policy

The following example policy lets the user named Bob perform any of the credential-related actions on his own credentials.

```
{
   "Version":"2012-10-17",
   "Statement":[{
      "Effect":"Allow",
      "Action":["iam:*AccessKey*","iam:*SigningCertificate*"],
      "Resource":"arn:aws:iam::123456789012:user/Bob"
      }
   ]
}
```

Notice that the resource in the policy is the Amazon Resource Name (ARN) for Bob. If you want to let a different user named Jane manage her own credentials, you would replace the ARN for Bob with the ARN for Jane in the policy, and then attach the policy to Jane. You must explicitly name the user in the resource.

The preceding policy uses wildcards (`"Action":"iam:*AccessKey*"`, where the * matches zero or multiple characters). You could instead explicitly list each of the IAM actions related to access keys and certificates (for example, `CreateAccessKey`, `UploadSigningCertificate`, and so on). If you use the wildcards in your policy, be aware that if we add any new IAM actions that include the string `AccessKey` or `SigningCertificate` in their names, this policy would automatically give Bob access to those new actions.

For information about uploading and managing policies, see Managing IAM Policies (p. 112).

Creating a Group Policy

The following policy might be attached to an IAM group. It gives members of the group permission to programmatically manage their own access keys and certificates.

```
{
  "Version": "2012-10-17",
  "Statement": [{
      "Effect": "Allow",
      "Action":["iam:*AccessKey*","iam:*SigningCertificate*"],
      "Resource":["arn:aws:iam::123456789012:user/${aws:username}"]
    }
  ]
}
```

The policy uses a policy variable (p. 139) (`${aws:username}`) that is evaluated at run time and contains the friendly name (p. 11) of the IAM user who made the request.

As in the policy for the individual user, this policy uses wildcards to specify the IAM actions related to access keys and certificates. If any new IAM actions are added that include the string `AccessKey` or `SigningCertificate` in their names, this policy would automatically give users access to those new actions.

Rotating Credentials

For the purposes of security, we recommend that you, an administrator, or your users regularly rotate the users' credentials (we recommend a frequency of every 90 days).

> **Note**
> If you're using the AWS account's credentials on a regular basis, we recommend you also regularly rotate those. Users can't manage credentials for the AWS account, so you must use the AWS account's credentials (and not a user's) when making the required API calls to rotate the credentials.

The following steps describe the general process for rotating either a key or a certificate without interrupting your applications.

1. While the first set of credentials is still active, create a second set of credentials (or upload a second one, in the case of a certificate), which will be active by default. At this point, the user has two active sets of credentials.

   Related action: `CreateAccessKey` or `UploadSigningCertificate`

   Related command: `iam-useraddkey` or `iam-useraddcert`
2. Update all applications to use the new credentials.
3. Change the state of the first set of credentials to `Inactive`.

   Related action: `UpdateAccessKey` or `UpdateSigningCertificate`

   Related command: `iam-usermodkey` or `iam-usermodcert`
4. Using only the new credentials, confirm that your applications are working well. If you need to, you can revert to using the original set of credentials by switching its state back to `Active`.
5. Delete the first set of credentials.

   Related action: `DeleteAccessKey` or `DeleteSigningCertificate`

   Related command: `iam-userdelkey` or `iam-userdelcert`

For information about creating and enabling or disabling access keys in the AWS Management Console, see Creating, Modifying, and Viewing User Security Credentials (p. 94). For information about uploading and enabling or disabling signing certificates in the console, see Creating and Uploading a User Signing Certificate (p. 97).

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, by default they don't have access to any resources at all. You must explicitly grant them permissions to access the resources that they need for their work.

This section describes permissions and the policies that define them. Permissions are rights that you grant to a user or group to let the user perform tasks in AWS. To define permissions, you use policies, which are documents in JSON format.

To learn more, we recommend you read the following sections:

- Overview of Permissions (p. 107) — This section discusses types of permissions, how to grant them, and how to manage permissions.
- Overview of Policies (p. 109) — 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 IAM Policies (p. 112) — This section describes how to create and manage policies by using the console, the IAM CLI, and the IAM API.
- Policy Reference (p. 121) — This section describes the elements, variables, and evaluation logic used in policies.

Overview of Permissions

Permissions let you specify who has access to AWS resource, 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.

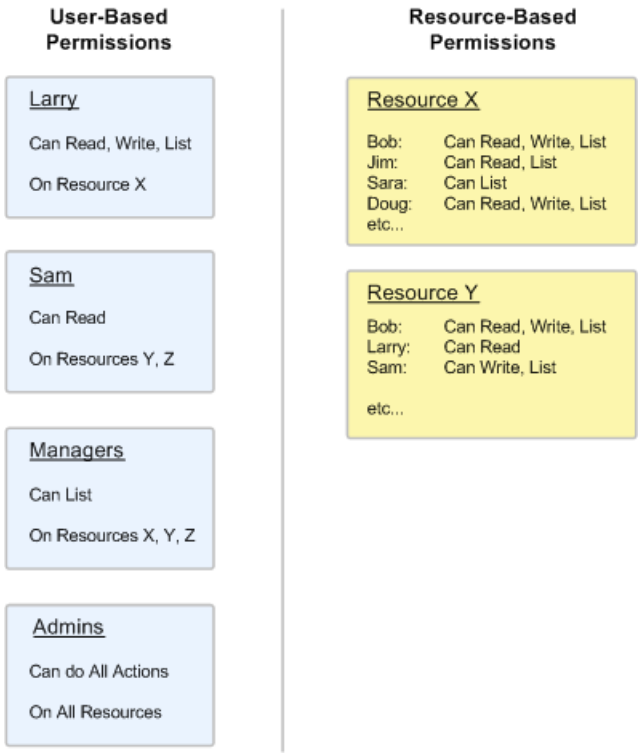
When you give permissions to a group, all users in that group get those permissions. For example, you can give the Admins group permission to perform any of the IAM actions on any of the AWS account resources. Another example: You can give the Managers group permission to describe the AWS account's Amazon 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 like an Amazon S3 bucket or an Amazon SNS topic. They let you specify who has access to the resource and what actions they can perform on it.

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. 145).

Note

Amazon S3 supports policies for IAM users and for resources (referred to in Amazon S3 as *bucket policies*. In addition, Amazon S3 supports a permission mechanism known as *ACLs* that's independent of IAM policies and permissions, but that can interact with it. 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 specifies includes permissions but that isn't associated with a specific user. Users from other accounts can then *assume* the role and access resources according to the permissions you've assigned to the role. For more information, see Enabling Cross-Account API Access (p. 155).

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 Amazon Elastic MapReduce, users need permission to access Elastic MapReduce, and must also be granted permission to manage EC2 instances. (For details, see Configure User Permissions with IAM in the *Amazon Elastic MapReduce Developer Guide*.)
- 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 Policies

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 user 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 lets IAM user Bob in account 123456789012 perform any actions in `mybucket`.

```
{
  "Version":"2012-10-17",
  "Id": "S3-Account-Permissions",
  "Statement": [
    {
      "Sid": "1",
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws:iam::123456789012:user/Bob"
        ]
      },
      "Action": "s3:*",
      "Resource": "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 IAM Policies (p. 116), AWS Services that Support IAM (p. 200), and the AWS Policy Examples page in the *AWS Sample Code & Libraries* section of the AWS website.

IAM policy templates and the policy generator are available from the IAM console of the AWS Management Console. For more information about creating policies in the console, see Managing IAM Policies (p. 112). Also, you can use the AWS Policy Generator online to create policies for AWS products without accessing the console.

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.

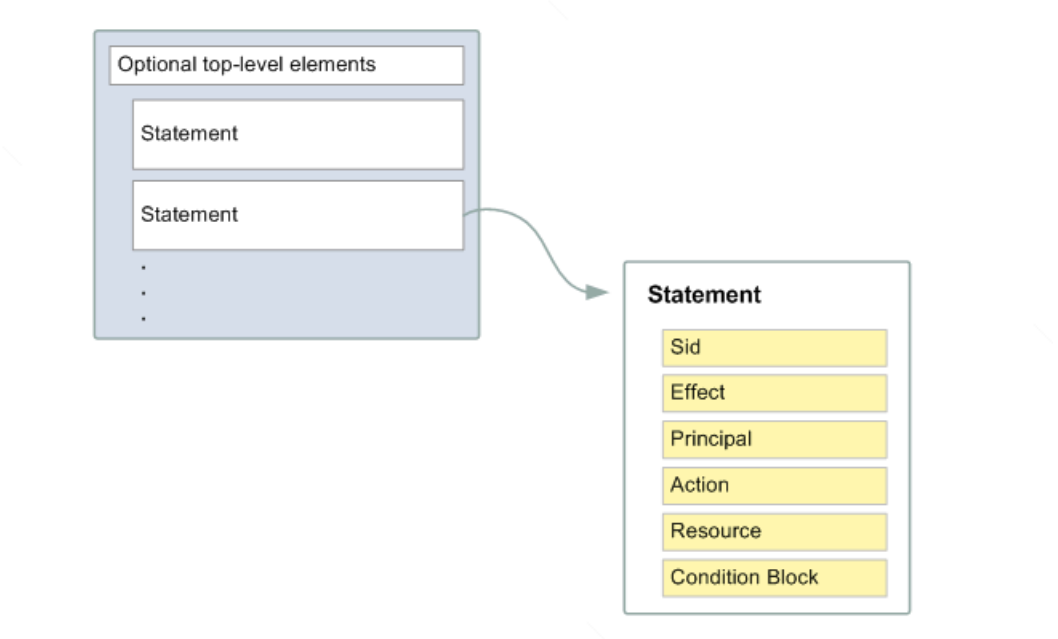
Users often have multiple policies that apply to them (but aren't necessarily *attached* to them). For example, IAM user Bob could have policies attached to him, and other policies attached to the groups he's in. In addition, he might be accessing an Amazon S3 bucket that has its own bucket policy (resource-based 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. 145).

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, we apply a logical `OR` across the statements at evaluation time. If multiple policies are applicable to a request, we apply 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 Elements (p. 121).

Managing IAM Policies

Topics

This section describes how create and manage IAM policies. For general information about what policies are and how they work, see Overview of Policies (p. 109).

When you want to add a permission to an IAM entity such as a user, group, or role, you create a policy in the access policy language that contains the permission, and then attach the policy to the entity. You can attach multiple policies to an entity. Or, instead of creating a new policy, you can edit an existing policy to add the permission. Policy names must be unique to the entity the policy is attached to. For information about how permissions are evaluated when permissions are applied to an IAM entity, see IAM Policy Evaluation Logic (p. 145).

> **Note**
> There are limits on the cumulative size of the policy and on the policy name. For more information, see Limitations on IAM Entities (p. 16).

You can get a list of the policies attached to a user, group, or role. But, there's no single IAM action that gets a list of all the policies you've created for all entities in the AWS account.

You can delete a policy at any time. If you use the API or CLI to delete a user, group, or role, you must delete the attached policies separately. For more information, see Deleting a User from Your AWS Account (p. 51), Deleting a Group (p. 59), or Deleting Roles or Instance Profiles (p. 182).

How you create, view, list, or delete a policy depends on which interface you're using to access IAM. The interface-specific details are covered in the following sections.

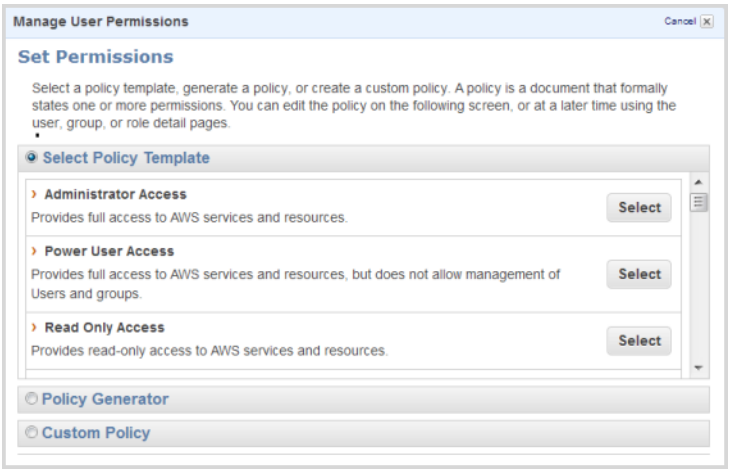
> **Tip**
> Policy templates and the policy generator are available in the AWS Management Console. You can use the AWS Policy Generator online to create policies for AWS products without accessing the console. For more information about the access policy language, see Overview of Policies (p. 109).

Managing Policies (AWS Management Console)

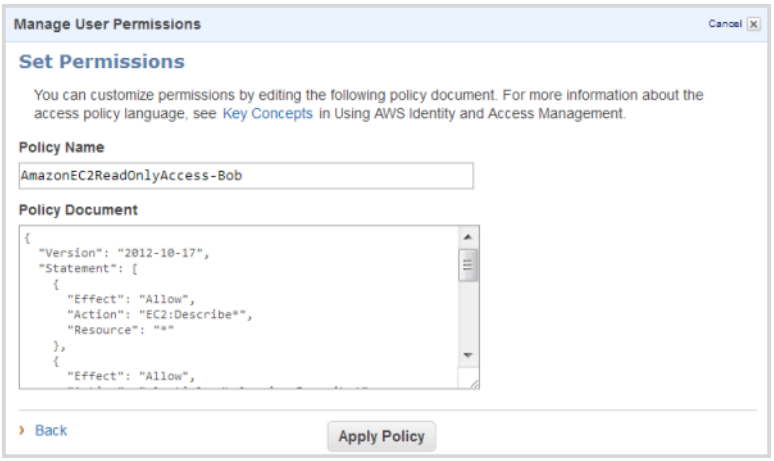
To create a policy and attach it to a group, user, or role

1. In the navigation pane, click **Groups**, **Users**, or **Roles**.
2. Select the group, user, or role you want to create a policy for, and then select the **Permissions** tab.
3. If you are creating a policy for a user, click **Attach User Policy**. If you are creating a policy for a group or role, click **Attach Policy**.

4. Choose the method for creating the policy document by clicking either **Select Policy Template**, **Policy Generator**, or **Custom Policy**. Then click **Select**.

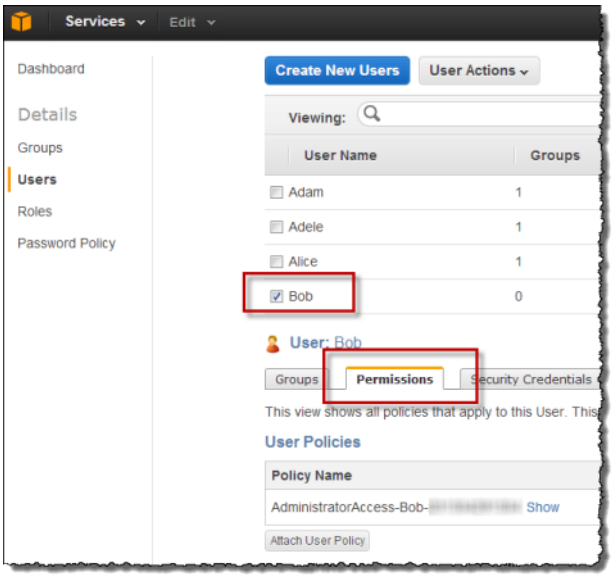


5. In the **Edit Permissions** screen, you can name the policy and create or edit your policy document. Or if you are using the policy generator to create your policy, select the appropriate **Effect**, **AWS Service**, and **Actions** options, enter the ARN (if applicable), and add any conditions you 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 **Continue**.
6. When you are satisfied with the policy, click **Apply Policy**. IAM applies the policy to the selected entity.



To view a policy or a list of policies associated with a user, group, or role

- In the navigation pane, click **Users**, **Groups**, or **Roles**, and then select the name of the entity whose policies you want to view.



IAM lists all policies associated with the entity on the **Permissions** tab. If you want to see the contents of a policy, click **Show**.

Note
For users, under **User Policies**, you can see any policies attached to the user directly. IAM lists policies attached to any groups that the user belongs to under **Group Policies**. You cannot modify group policies from a user's **Permissions** tab. To create or modify group permissions, you must go to the **Permissions** tab for the group whose permissions you want to change.

Note
Currently, permission to assume a role is limited only to Amazon EC2 instances in your AWS account. The permissions described in this step control only what the entity who assumes the role can do.

To edit or delete a policy for a group, user, or role

1. In the navigation pane, click **Groups**, **Users**, or **Roles**.
2. Select the group, user, or role you want to modify a policy for, and then select the **Permissions** tab.
3. If you're editing a policy, click **Manage Policy**. You can edit the policy in the **Edit Permissions** screen (shown in the preceding task), or, if you prefer to work from a policy template, click **View Policy Templates** and select a template. You can also use the policy generator to modify a policy.
4. To delete a policy, click **Remove Policy**. The policy is deleted and permissions associated with the policy no longer apply to the selected group, user, or role.

Managing Policies (CLI)

The following table lists which CLI command to use for each task you want to perform with a policy. For more information about the commands, go to the AWS Identity and Access Management Command Line Interface Reference.

For an example of attaching a policy and listing policies, see Adding a Policy to the Group (p. 28).

Task	Command to Use

Attach a simple policy by passing in the contents of the policy as parameters in the command	`iam-groupaddpolicy`
	`iam-useraddpolicy`
	`iam-roleaddpolicy`

Attach a JSON policy document you've written	`iam-groupuploadpolicy`
	`iam-useruploadpolicy`
	`iam-roleuploadpolicy`

List policies	`iam-grouplistpolicies`
	`iam-userlistpolicies`
	`iam-rolelistpolicies`

Delete a policy	`iam-groupdelpolicy`
	`iam-userdelpolicy`
	`iam-roledelpolicy`

Managing Policies (API)

The following table lists which API actions to use for each task you want to perform with a policy. For more information about the actions, go to the AWS Identity and Access Management API Reference, or refer to your SDK's documentation.

For an example of attaching a policy and listing policies, see Adding a Policy to the Group (p. 28).

Task	Action to Use

Attach a JSON policy document you've written	`PutGroupPolicy`
	`PutUserPolicy`
	`PutRolePolicy`

List policies	`ListGroupPolicies`
	`ListUserPolicies`
	`ListRolePolicies`

Get the contents of a policy	`GetGroupPolicy`
	`GetUserPolicy`
	`GetRolePolicy`

Delete a policy	`DeleteGroupPolicy`
	`DeleteUserPolicy`
	`DeleteRolePolicy`

Example IAM Policies

This section shows some examples of policies that control access for your IAM resources.

Topics

Allow users to access a "home directory" in Amazon S3

The following policy can be attached to an IAM group. It lets the IAM user in that group programmatically access a "home directory" in Amazon S3. Each user can perform all Amazon S3 `GetObject`, `ListBucket`, `PutObject` actions on the contents of the folder inside the `mybucket` bucket—that is, the user can read objects from the folder and write objects to the folder.

> **Note**
> In the following policy, you need to replace `myBucket` with the name of a bucket under which you have created a `home` folder and folders for individual users.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": ["s3:*"],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::mybucket"],
      "Condition":{"StringLike":{"s3:prefix":["home/${aws:username}/*"]}}
    },
    {
       "Action":["s3:*"],
       "Effect":"Allow",
       "Resource": ["arn:aws:s3:::mybucket/home/${aws:username}/*"]
    }
  ]
}
```

The policy uses a policy variable (p. 139) (`${aws:username}`) that is evaluated at run time and contains the friendly name (p. 11) of the IAM user who made the request.

The following policy expands on the previous one by allowing the user to additionally use the Amazon S3 console to manage the "home directory."

> **Note**
> In the following policy, you need to replace `myBucket` with the name of a bucket under which you have created a `home` folder and folders for individual users.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": ["s3:ListAllMyBuckets", "s3:GetBucketLocation"],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::*"]
    },
    {
      "Action": ["s3:ListBucket"],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::myBucket"],
      "Condition":{"StringEquals":{"s3:prefix":["","home/"],"s3:delim
iter":["/"]}}
     },
    {
      "Action": ["s3:ListBucket"],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::myBucket"],
      "Condition":{"StringLike":{"s3:prefix":["home/${aws:username}/*"]}}
    },
    {
       "Action":["s3:*"],
       "Effect":"Allow",
       "Resource": ["arn:aws:s3:::myBucket/home/${aws:username}",
      "arn:aws:s3:::myBucket/home/${aws:username}/*"]
    }
  ]
}
```

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 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 to work only with a specific set of AWS products and resources

This policy is for an IAM group that all users in a company belong to. The policy gives users access to only the following:

- The AWS account's Amazon SimpleDB domain called mySDBDomain (which exists only in the us-east-1 Region)
- The AWS account's corporate Amazon S3 bucket called `my_corporate_bucket` and all the objects it contains. 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":["sdb:*","s3:*"],
    "Resource":["arn:aws:sdb:us-east-1:123456789012:domain/mySDBDomain",
      "arn:aws:s3:::my_corporate_bucket",
      "arn:aws:s3:::my_corporate_bucket/*"]
    },
    {
    "Effect":"Deny",
    "Action":["sdb:*","s3:*"],
    "NotResource":["arn:aws:sdb:us-east-1:123456789012:domain/mySDBDomain",
      "arn:aws:s3:::my_corporate_bucket",
      "arn:aws:s3:::my_corporate_bucket/*"]
    }
  ]
}
```

The explicit deny, in conjunction with the `NotResource` element, helps to ensure that the users can't use any other AWS products or resources than those specified in the policy.

Allow a user all actions on an Amazon DynamoDB table whose name matches the user name

The following policy gives a user permission to programmatically access an Amazon DynamoDB table that has their name. For example, user Bob can perform any Amazon 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 Amazon DynamoDB table.

```
{
  "Version": "2012-10-17",
  "Statement": [{
      "Effect": "Allow",
      "Action": ["dynamodb:*"],
     "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/${aws:username}"

    }
  ]
}
```

The policy uses a policy variable (p. 139) (`${aws:username}`) that is evaluated at run time and contains the friendly name (p. 11) of the IAM user who made the request.

Allow a user to manage his or her own security credentials

The following policy might be attached to an IAM group. It gives members of the group permission to programmatically manage their own access keys and certificates.

```
{
  "Version": "2012-10-17",
  "Statement": [{
      "Effect": "Allow",
      "Action":["iam:*AccessKey*","iam:*SigningCertificate*"],
      "Resource":["arn:aws:iam::123456789012:user/${aws:username}"]
```

```
    }
  ]
}
```

The policy uses wildcards to specify all the IAM actions related to access keys and certificates (the * matches zero or multiple characters). You could instead list each action explicitly. The benefit of using wildcards (*) instead of explicitly listing each action is that if any new IAM actions were to be added that include the string `AccessKey` or `SigningCertificate` in their names, this policy would automatically give users access to those new actions to use with his own credentials.

The policy uses a policy variable (p. 139) (`${aws:username}`) that is evaluated at run time and contains the friendly name (p. 11) of the IAM user who made the request.

Allow a user to manage a group's membership

This policy lets the IAM user that it's attached to update the membership of the group called *MarketingGroup*.

```
{
  "Version": "2012-10-17",
  "Statement":[{
    "Effect":"Allow",
    "Action":[
       "iam:AddUserToGroup",
       "iam:RemoveUserFromGroup",
       "iam:GetGroup"
    ],
    "Resource":"arn:aws:iam::123456789012:group/MarketingGroup"
    }
  ]
}
```

Allow requests only if they come from a certain IP address or range

This policy is for an IAM group that all users in a company belong to. The policy denies access to all actions in the account unless the request comes from 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 the company are within the specified ranges.) A typical use is for Amazon VPC, where you might expect all your users' requests to originate from a particular IP address, and so you want to deny requests from any other address.

```
{
  "Version": "2012-10-17",
  "Statement":[{
    "Effect":"Deny",
    "Action":"*",
    "Resource":"*",
    "Condition":{
       "NotIpAddress":{
          "aws:SourceIp":["192.0.2.0/24", "203.0.113.0/24"]
          }
       }
    }
```

```
  ]
}
```

Allow a user to list the AWS account's groups and users for reporting purposes

This policy allows the user it is attached to call any IAM action that starts with the literal string `Get` or `List`. You probably don't want the user reading the access key IDs of other users, so the policy includes a statement that denies the user access to the `ListAccessKeys` action.

```
{
  "Version": "2012-10-17",
  "Statement":[{
    "Effect":"Allow",
    "Action":[
      "iam:Get*",
      "iam:List*"
      ],
    "Resource":"*"
    },
    {
    "Effect":"Deny",
    "Action":
      [
        "iam:ListAccessKeys",
        "iam:GetUserPolicy",
        "iam:GetGroupPolicy",
        "iam:GetRolePolicy"
      ],
    "Resource":"*"
    }
  ]
}
```

The benefit of using `List*` for the action is that if new types of security credentials were added to IAM in the future, the access granted in the policy to all `List*` actions would automatically allow the user to list those new credentials.

Allow users signed in using Login with Amazon to access their own Amazon S3 bucket

Web identity federation lets you provide access to AWS resources for users who have signed in using a third-party identity provider like Amazon, Facebook, or Google instead of using an IAM user identity. To configure web identity federation, you create a role that determines what permissions the federated user will have. If you want to support more than one identity provider, you must create multiple roles, one per provider. For example, if you want to support Login with Amazon, Facebook, and Google, you must create three roles, and for each role indicate the provider that the role is for. Before you can create a role for web identity federation, you must register your application with the identity provider, who will give you an application ID.

The following example shows a policy that might be used for a mobile app that uses web identify federation (specifically, Login with Amazon). The condition makes sure that the user has access to objects in the Amazon S3 `myBucket` bucket only if the object's name includes a provider name (here, `amazon`), the

friendly name of the application (`mynumbersgame`), and the federated user's ID. You need additional, though similar, policies to grant permissions to users who are logged in using Facebook and Google. For more information about web identity federation, see Creating Temporary Security Credentials for Mobile Apps Using Third-Party Identity Providers.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["s3:ListBucket"],
      "Resource": ["arn:aws:s3:::myBucket"],
      "Condition": {
        "StringLike": {
          "s3:prefix": ["amazon/mynumbersgame/"]
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObject"
      ],
      "Resource": [
        "arn:aws:s3:::myBucket/amazon/mynumbersgame/${www.amazon.com:user_id}",

        "arn:aws:s3:::myBucket/amazon/mynumbersgame/${www.amazon.com:user_id}/*"

      ]
    }
  ]
}
```

Policy Reference

This section presents detailed syntax, descriptions, and examples of the elements, variables, and evaluation logic of policies. It includes the following sections.

- Elements (p. 121) — This section describes each of the elements that you can use when creating a policy. It includes additional policy examples and describes conditions, supported data types, and how they are used in various services.
- Policy Variables (p. 139) — This section describes placeholders that you can specify in a policy that are replaced during policy evaluation with values from the request.
- IAM Policy Evaluation Logic (p. 145) — This section describes AWS requests, how they are authenticated, and how AWS uses policies to determine access to resources.

Elements

This section describes the elements that you can use in a 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. 200).

Version

The `Version` element specifies the access 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. 139), 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 IAM policies, the service automatically sets the policy's `Id` value to the policy name that you create. If you attempt to set the `Id` element, the policy will rejected. The `Id` value for an IAM policy might look like the following example.

```
"Id":"Admin_Policy"
```

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 buckets 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. 140).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:ListAllMyBuckets"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:s3:::*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject"
      ],
      "Resource": "arn:aws:s3:::mybucket/home/${aws:username}/*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket"
      ],
      "Resource": "arn:aws:s3:::mybucket",
      "Condition": {
        "StringLike": {
          "s3:prefix": [
```

```
            "/${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 you want the statement to 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. 145).

Principal

The `Principal` element specifies the user, account, service, or other entity that is allowed or denied access to a resource.

For IAM users and groups, you do not specify a principal. In those cases, the principal is the user whose credentials are used to make the request. (If the policy is attached to a group, the principal is the member of the group that's making the request.) For IAM roles, principals are used to specify who can assume the role. In other services, principals are used in policies that are attached to a resource, such as an Amazon S3 bucket or an Amazon SQS queue.

You specify a principal using the *Amazon Resource Name* (ARN) of the account, IAM user, or role. (Note that IAM groups cannot be specified as principals.) For more information about the format of ARNs, see Amazon Resource Names (ARNs) and AWS Service Namespaces.

In IAM, you can use wildcards (*) to indicate all possible users. As a shortcut, if you're specifying an account ID, you can use a shortened form that consists of the `AWS:` prefix followed by the ID.

The following examples show various ways in which principals can be specified.

```
<!-- Everyone (anonymous users) -->
"Principal":"AWS":"*.*"

<!-- Specific account or accounts -->
"Principal":{"AWS":"arn:aws:iam::123456789012:root" }
"Principal":{"AWS":"123456789012"}
"Princip
al":{"AWS":["arn:aws:iam::123456789012:root","arn:aws:iam::999999999999:root"]}


<!-- Individual user -->
"Principal":"AWS":"arn:aws:iam::123456789012:user/Username"

<!-- Federated user (using web identity federation)
"Principal":{"Federated":"www.amazon.com"}
"Principal":{"Federated":"graph.facebook.com"}
"Principal":{"Federated":"accounts.google.com"}

<!-- Specific role -->
"Principal":{"AWS":["arn:aws:iam::123456789012:role/MyRole"]}

<!-- Specific service -->
"Principal":{"Service":["ec2.amazonaws.com"]}
```

Some services support additional options for specifying a principal. For example, Amazon S3 lets you use a canonical user in a format like this:

```
<!-- Individual user in S3 -->
"Principal":{"CanonicalUser":"79a59df900b949e55d96a1e698fbaced
fd6e09d98eacf8f8d5218e7cd47ef2be"}
```

For more information, see the following:

- Creating Temporary Security Credentials for Mobile Apps Using Third-Party Identity Providers in the *Using Temporary Security Credentials* guide.
- Amazon SQS Policy Examples in the *Amazon Simple Queue Service Developer Guide*
- Example Cases for Amazon S3 Bucket Policies in the *Amazon Simple Storage Service Developer Guide*
- Account Identifiers in the *AWS General Reference*

NotPrincipal

The `NotPrincipal` element lets you specify an exception to a list of principals. For example, you can use this to prevent all AWS accounts *except* a specific account from accessing a resource. Conversely, you can deny access to all principals except the one named in the `NotPrincipal` element. As with `Principal`, you specify the user or account that should be allowed or denied permission; the difference is that `NotPrincipal` translates to everyone *except* that person or account.

In the following example, all users are denied access to a resource except for the user named Bob in the AWS account 123456789012.

```
"Effect": "Deny",
"NotPrincipal": {
```

```
    "AWS":"arn:aws:iam::123456789012:user/Bob"
}
```

Action

The `Action` element describes the type of access that should be allowed or denied (for example, read, write, list, delete, and so on). 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.

You specify a value using a namespace that identifies a service (`iam`, `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.

```
<!-- SQS action -->
"Action":"sqs:SendMessage"

<!-- EC2 action -->
"Action":"ec2:StartInstances"

<!-- IAM action -->
"Action":"iam:ChangePassword"

<!-- 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 (* or ?) 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`.

```
"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.

The following example refers to a specific Amazon SQS queue.

```
"Resource":"arn:aws:sqs:us-west-2:111122223333:queue1"
```

The following example refers to the IAM user named Bob in an the AWS account 123456789012.

```
"Resource":"arn:aws:iam::123456789012: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::123456789012:user/acounting/*"
```

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 Amazon DynamoDB tables.

```
"Resource":[
    "arn:aws:dynamodb:us-west-2:123456789012:table/books_table",
```

```
    "arn:aws:dynamodb:us-west-2:123456789012:table/maagazines_table"
]
```

In the `Resource` element, you can use policy variables (p. 139) 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-1:123456789012:table/${aws:username}"

    }
  ]
}
```

For more information about policy variables, see Policy Variables (p. 139).

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 examples 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.

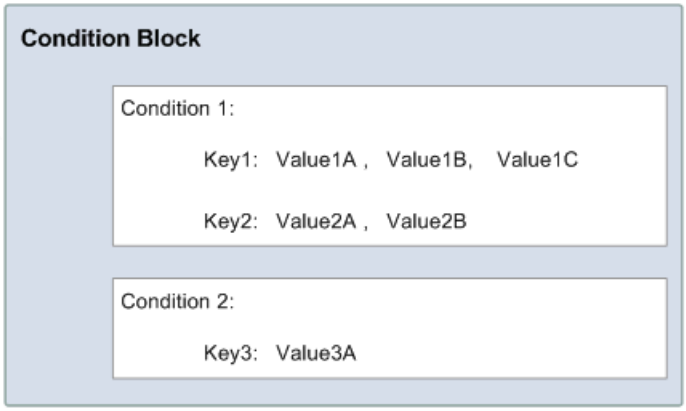
Condition

The `Condition` element 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 also let you additional values in conditions; for example,

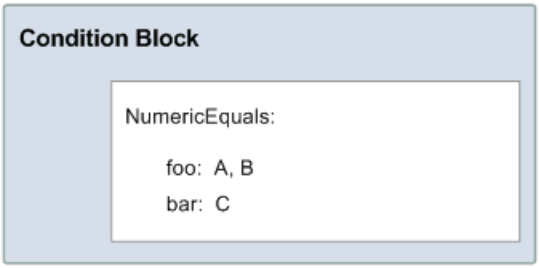
Amazon S3 lets you write a condition using the `s3:VersionId` key, which is unique to that service. (For more information about writing conditions in policies for Amazon S3, see Using IAM Policies in the Amazon Simple Storage Service Developer Guide.)

The Condition Block

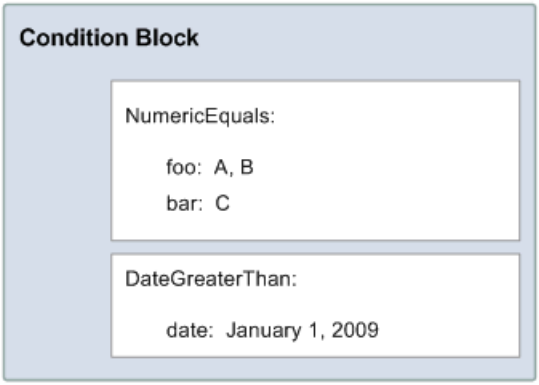
A `Condition` element, or *condition block*, can contain multiple conditions, and each condition can contain multiple key-value pairs. The following figure illustrates this. Unless otherwise specified, all keys can have multiple values.



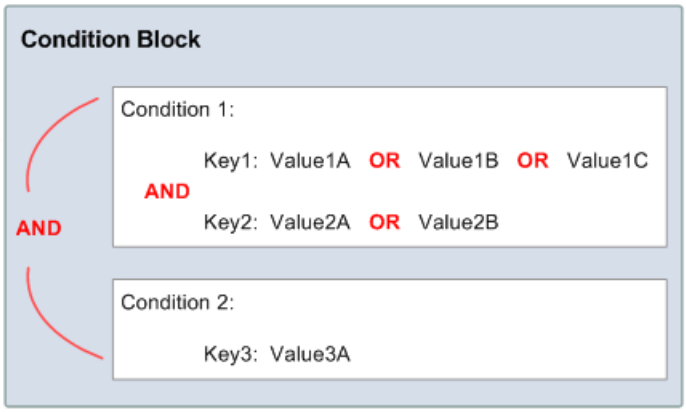
An example of a condition is `NumericEquals`. Let's say you want to let John use a resource only if a numeric value *foo* equals either A or B, and another numeric value *bar* equals C. You would create a condition block that looks like the following figure.



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"]
      }
}
```

Available Keys

AWS provides the following predefined keys for all AWS services that support the access policy language for access control:

- `aws:CurrentTime`–To check for date/time conditions (see Date Conditions (p. 135)).
- `aws:EpochTime`–To check for date/time conditions using a date in epoch or UNIX time (see Date Conditions (p. 135)).
- `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. 138), Numeric Conditions (p. 135) and Using Multi-Factor Authentication (MFA) Devices with AWS (p. 76).
- `aws:principaltype`–To check the type of principal (user, account, federated user, etc.) for the current request (see String Conditions (p. 132)).
- `aws:SecureTransport`–To check whether the request was sent using SSL (see Boolean Conditions (p. 136)).
- `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. 137)). Note that if you use `aws:SourceIp`, and the request comes from an Amazon EC2 instance, the instance's public IP address is evaluated.
- `aws:UserAgent`–To check the requester's client application (see String Conditions (p. 132)).
- `aws:userid`–To check the requester's user ID (see String Conditions (p. 132)).
- `aws:username`–To check the requester's user name (see String Conditions (p. 132)).

> **Note**
> Key names are case sensitive.

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 keys that are available in other services, see the documentation for the individual services.

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 (such as Amazon, Google, or Facebook), additional 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 specific to a specific registered app and to a specific federated user.

Federated Provider

The `aws:FederatedUser` key identifies which of the identity providers was used to authenticate the user. For example, 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:FederatedUser` key as a policy variable in the ARN of a resource. The policy allows any user who has been authenticated using an identity provider to get objects out of a folder in an Amazon S3 bucket that's specific to their identity provider.

```
{
  "Statement": [
    {
      "Version": "2012-10-17",
      "Principal": "*",
      "Effect": "Allow",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::mybucket/${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 identity provider-specific names:

- 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).

For more information about web identity federation, see Creating Temporary Security Credentials for Mobile Apps Using Third-Party Identity Providers.

Condition Types

These are the types of conditions you can specify:

- String
- Numeric
- Date and time
- Boolean
- 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. Short version: `streq`
`StringNotEquals`	Negated matching Short version: `strneq`
`StringEqualsIgnoreCase`	Exact matching, ignoring case Short version: `streqi`
`StringNotEqualsIgnoreCase`	Negated matching, ignoring case Short version: `strneqi`
`StringLike`	Case-sensitive matching. The values can include a multi-character match wildcard (*) or a single-character match wildcard (?) anywhere in the string. Short version: `strl`
`StringNotLike`	Negated case-insensitive matching. The values can include a multi-character match wildcard (*) or a single-character match wildcard (?) anywhere in the string. Short version: `strnl`

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::123456789012:user/*",
      "Condition":{
         "StringEquals":{
            "aws:UserAgent":"Example Corp Java Client"
            }
         }
      }
   ]
}
```

The following example uses string matching with a policy variable (p. 139) 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": [
    {
      "Action": [
```

```
        "s3:ListAllMyBuckets",
        "s3:GetBucketLocation"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:s3:::*"
      ]
    },
    {
      "Action": [
        "s3:ListBucket"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:s3:::myBucket"
      ],
      "Condition": {
        "StringEquals": {
          "s3:prefix": [
            "",
            "home/"
          ],
          "s3:delimiter": [
            "/"
          ]
        }
      }
    },
    {
      "Action": [
        "s3:ListBucket"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:s3:::myBucket"
      ],
      "Condition": {
        "StringLike": {
          "s3:prefix": [
            "home/${aws:username}/*"
          ]
        }
      }
    },
    {
      "Action": [
        "s3:*"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:s3:::myBucket/home/${aws:username}",
        "arn:aws:s3:::myBucket/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 federation identity, see Allow users signed in using Login with Amazon to access their own Amazon S3 bucket (p. 120).

Numeric Conditions

Numeric conditions let you restrict access based on comparing a key to an integer or decimal value. (Fractional or irrational values are not supported.)

Condition	Description
`NumericEquals`	Matching Short version: `numeq`
`NumericNotEquals`	Negated matching Short version: `numneq`
`NumericLessThan`	"Less than" matching Short version: `numlt`
`NumericLessThanEquals`	"Less than or equals" matching Short version: `numlteq`
`NumericGreaterThan`	"Greater than" matching Short version: `numgt`
`NumericGreaterThanEquals`	"Greater than or equals" matching Short version: `numgteq`

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 Short version: `dateeq`
`DateNotEquals`	Negated matching Short version: `dateneq`
`DateLessThan`	A point in time at which a key stops taking effect Short version: `datelt`
`DateLessThanEquals`	A point in time at which a key stops taking effect Short version: `datelteq`
`DateGreaterThan`	A point in time at which a key starts taking effect Short version: `dategt`
`DateGreaterThanEquals`	A point in time at which a key starts taking effect Short version: `dategteq`

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::123456789012: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::123456789012:user/*",
    "Condition":{
      "Bool":{
         "aws:SecureTransport":"true"
      }
    }
  }]
}
```

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::123456789012:user/*",
    "Condition":{
      "IpAddress":{
         "aws:SourceIp":"203.0.113.0/24"
      }
    }
  }]
}
```

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.)

Condition	Description
`ArnEquals`	Matching for ARN
`ArnNotEquals`	Negated matching for ARN

Condition	Description
`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": "2008-10-17",
  "Id": "arn:aws:sqs:us-west-2:123456789012:user_queue1/SQSDefaultPolicy",
  "Statement": [
    {
      "Sid": "Sid1234567890123",
      "Effect": "Allow",
      "Principal": {
        "AWS": "123456789012"
      },
      "Action": ["SQS:SendMessage"],
      "Resource": "arn:aws:sqs:us-west-2:336924118301:your_queue_1",
      "Condition": {
        "ArnEquals": {
          "aws:SourceArn": "arn:aws:sns:us-east-1:123456789012:your_topic_1"
        }
      }
    }
  ]
}
```

...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."

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 has authenticated with MFA (multi-factor authentication). The following example shows a condition that states that MFA authentication must exist (be *not null*) for the user to use the Amazon EC2 API.

```
{
  "Version": "2012-10-17",
  "Statement":[{
      "Action":["ec2:*"],
```

```
      "Effect":"Allow",
      "Resource":["*"],
      "Condition":{
         "Null":{"aws:MultiFactorAuthAge":"false"}
      }
    }
  ]
}
```

Supported Data Types

This section lists the set of data types the access policy language supports. The 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

Policy Variables

Topics

Introduction

In IAM policies, you provide a name for the specific resources that you 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 IAM 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*, which 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. 11) 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 in 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, `${aws:username}` and other variables are treated as literal strings in the policy.

You can use policy variables in a similar way to allow each IAM user to be able 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::123456789012:user/David"]
    }
  ]
}
```

If this policy is attached to IAM 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::123456789012: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 an IAM group. It gives each of the users in the IAM group full programmatic access to a user-specific object (their own "home directory") in Amazon S3.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": ["s3:*"],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::mybucket"],
      "Condition":{"StringLike":{"s3:prefix":["home/${aws:username}/*"]}}
    },
    {
       "Action":["s3:*"],
       "Effect":"Allow",
       "Resource":["arn:aws:s3:::mybucket/home/${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 User IDs (p. 15).

The following policy might be used in an IAM group policy. 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 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.). The following Amazon SNS topic policy gives

users the ability to manage (perform all actions for) the topic only if the URL matches their AWS user name.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sns:*",
      "Principal": {"AWS": "*"},
      "Condition": {
        "StringLike": {
          "sns:endpoint": "https://example.com/${aws:username}/*"
        }
      }
    }
  ]
}
```

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. 146).

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: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. 11) 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. 15)
  - `aws:principaltype`: `User`

- **Federated user**
  - `aws:username`: (not present)
  - `aws:userid`: *`account`*:*`caller-specified-name`*
  - `aws:principaltype`: `FederatedUser`
- **Web federated user**

  Note
  For information about policy keys that are available when you use web identity federation, see Identifyhing 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`
- **Anonymous caller** (Amazon SQS Amazon SNS and Amazon S3 only)
  - `aws:username`: (not present)
  - `aws:userid`: (not present)
  - `aws:principaltype`: `Anonymous`
- **Assumed role**
  - `aws:username`: (not present)
  - `aws:userid`: *`role-id`*:*`caller-specified-role-name`*
  - `aws:principaltype`: `Assumed role`

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.
- *`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.

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. This proxy application first authenticates individual users using the corporate network; the proxy application then uses its IAM identity to get temporary security credentials for individual users and gives them to the user's local copy of the corporate application. The user's local copy of the corporate application can use these temporary credentials to call AWS.

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 Amazon, Google, or Facebook. The app can then use the user's authentication information from these providers to get temporary security credentials for accessing AWS resources.

For more information about federation and getting temporary security credentials, see 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 Getting Started Guide*.

For More Information

For more information about policies, see the following:

- Overview of Permissions (p. 107)
- Example IAM Policies (p. 116)
- Elements (p. 121)
- IAM Policy Evaluation Logic (p. 145)
- Creating Temporary Security Credentials for Mobile Apps Using Third-Party Identity Providers in the *Using Temporary Security Credentials* guide.

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 Using ACLs and Bucket Policies Together 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 Amazon 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}/*"
      ]
```

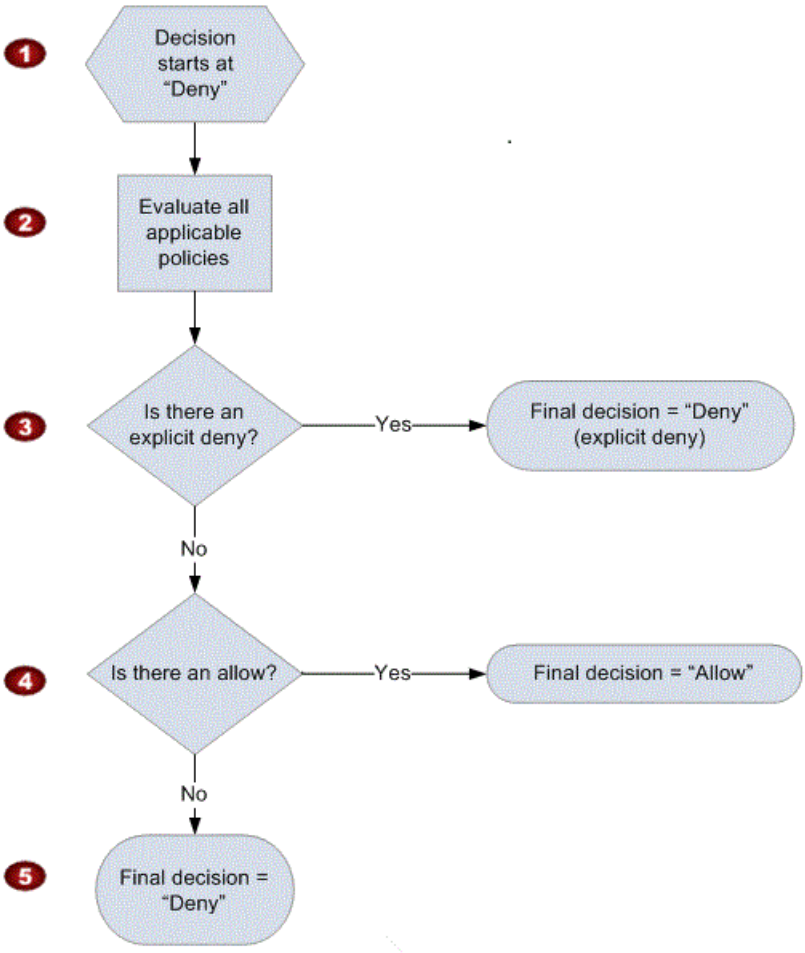
Determining Whether a Request is Allowed or Denied

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. 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."

Example : Explicit Deny Overriding an Allow

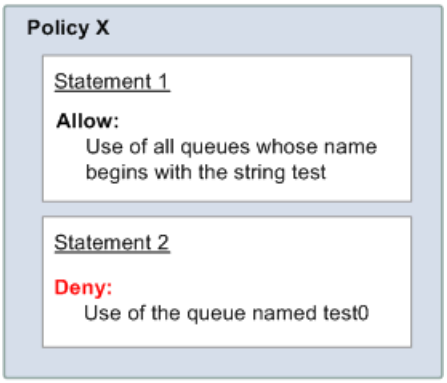
This example illustrates how you can use an explicit deny to override a broad policy that allows access to a wide set of resources. Let's say that you give an IAM group permissions to use any Amazon SQS queues in your AWS account whose names begin with the string `test`.

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 just add a statement to the existing policy) that explicitly denies the group's access to just that 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.



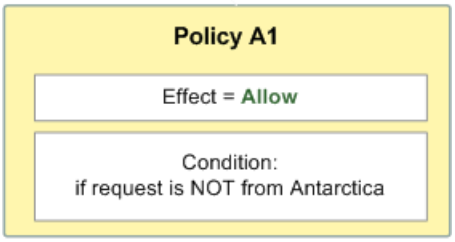
When any user in the group makes a request to use the queue test0, the explicit deny overrides the allow, and the user is denied access to the queue.

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.

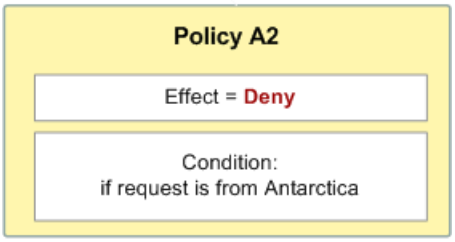
A policy also results in a deny if a condition in a statement isn't met. If all conditions in the statement are met, the policy results in either an allow or an explicit deny, based on the value of the `Effect` element in the policy. Policies don't specify what to do if a condition isn't met, so the 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.

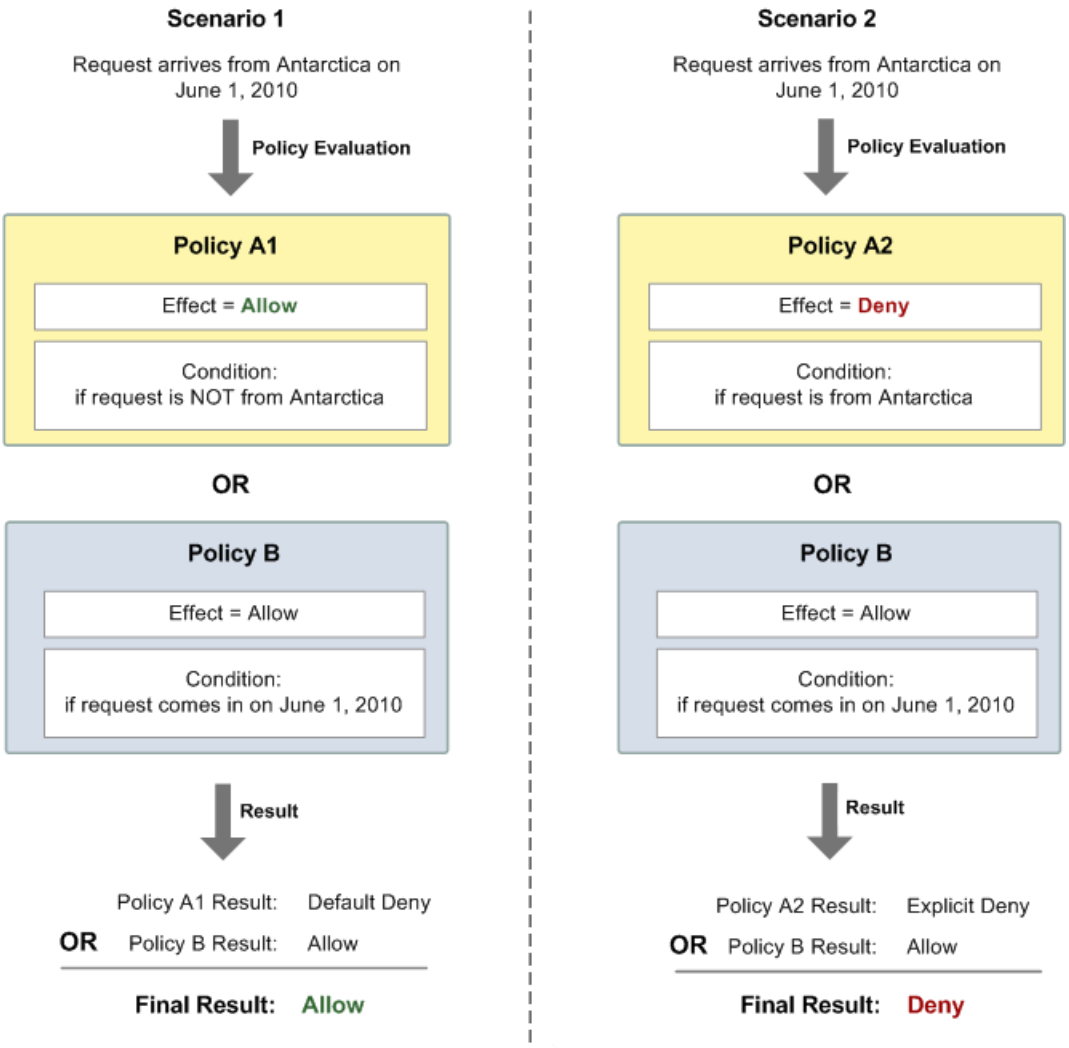


If this policy applies to a user who sends a request from Antarctica, the condition is met, and the policy's result is therefore a deny.

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.



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.

Roles

IAM roles allow you to delegate access to users or services that normally don't have access to your organization's AWS resources. IAM users or AWS services can assume a role to obtain temporary security credentials that can be used to make AWS API calls. Consequently, you don't have to share long-term credentials or define permissions for each entity that requires access to a resource. The following scenarios highlight some of the challenges that you can address by delegating access:

Granting applications that run on Amazon EC2 instances access to AWS resources
: To grant applications on an Amazon EC2 instance access to AWS resources, developers might distribute their credentials to each instance. Applications can then use those credentials to access resources such as Amazon S3 buckets or Amazon DynamoDB data. However, distributing long-term credentials to each instance is challenging to manage and a potential security risk.

Cross-account access
: To control or manage access to resources, such as isolating a development environment from a production environment, you might have multiple AWS accounts. However, in some cases, users from one account might need to access resources on the other account. For example, a user from the development environment might require access to the production environment to promote an update. Therefore, users must have credentials for each account, but managing multiple credentials for multiple accounts makes identity management difficult.

Instead of creating and distributing multiple credentials, you can delegate API access by using IAM roles. You create a role in the AWS account where you want to delegate access. When you create the role, you specify two policies: one that defines who can assume the role (the trusted entities) and another that defines what actions can be done on which resources. You can specify account IDs or AWS services as trusted entities.

For IAM users to assume the role, they must be granted permission to call `sts:AssumeRole` for that specific role. After assuming the role, the user can use the role's temporary security credentials to access resources that are defined in the role. With the temporary security credentials, the user acts as the role and has only the permissions that are associated with the role. Callers can further restrict permissions by passing an additional policy when they assume a role.

For details about how you can use roles to delegate API access, see the following scenarios:

- Granting Applications that Run on Amazon EC2 Instances Access to AWS Resources (p. 152)
- Enabling Cross-Account API Access (p. 155)

Granting Applications that Run on Amazon EC2 Instances Access to AWS Resources

Imagine that you are an administrator who manages your organization's AWS resources. Developers in your organization have applications that run on Amazon EC2 instances. These applications require access to other AWS resources—for example, making updates to Amazon S3 buckets.

Applications that run on an Amazon EC2 instance must sign their AWS API requests with AWS credentials. One way to do this is for developers to pass their AWS credentials to the Amazon EC2 instance, allowing applications to use the credentials to sign requests.

However, when AWS credentials are rotated, developers have to update each Amazon EC2 instance that uses their credentials.

IAM Roles Provide a Solution

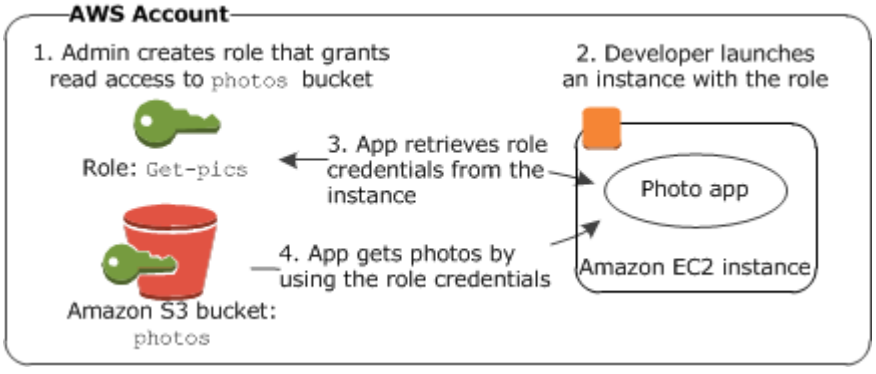
With IAM roles, no one has to distribute AWS credentials to Amazon EC2 instances. Instead, an administrator can create a role with the permissions that applications require when the application makes calls to other AWS resources. When developers launch an Amazon EC2 instance, they can specify a role to associate with the instance. All applications that run on the instance can use the role credentials to sign requests. Because role credentials are temporary and rotated automatically, you don't have to worry about long-term security risks.

Also, if you use a single role for multiple instances, you can make a change to the role and that change is propagated to all the instances, simplifying credential management.

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, which grants read permissions for the bucket, and allows the developer to launch it 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 credentials.

Application on an instance accessing an AWS resource



1. An administrator uses IAM to create the **`Get-pics`** role. In the role, the administrator defines that only Amazon EC2 instances can assume the role and allows read permissions for the photos bucket.

2. A developer launches an Amazon EC2 instance and associates the `Get-pics` role with that instance. Now, any application on the instance can assume the `Get-pics` role to obtain read access to the photos bucket.

   Important
   Because roles are temporary security credentials, you must ensure that applications that use roles on Amazon EC2 instances support temporary security credentials. Otherwise, the application might be denied access.

3. When the application runs, it uses the role credentials that are available on the Amazon EC2 instance. The application retrieves credentials from the instance metadata.
4. Using the role credentials, the application accesses the photo bucket with read-only permissions.

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",
      "Resource":"*"
    },
    {
      "Effect":"Allow",
      "Action":"iam:ListInstanceProfiles",
      "Resource":"*"
    },
    {
      "Effect":"Allow",
      "Action":"ec2:*",
      "Resource":"*"
    }]
}
```

Important
Users could indirectly escalate their privileges by launching an Amazon EC2 instance with a role that has more permissions assigned to it than the user. After launching the instance, users can access the role if they have access to the launched instance. You can control which roles users can use when launching Amazon EC2 instances by limiting permissions on the `PassRole` action so that users can launch instances with certain roles and not any role.

The following sample policy allows developers to use the Amazon EC2 API to launch an instance with a role. The `Resource` element specifies an Amazon Resource Name (ARN). By specifying the ARN, the policy grants the user to pass only the `Get-pics` role.

Example Policy that grants a user permission to launch an instance with a role by using the Amazon EC2 API

```
{
  "Version": "2012-10-17",
  "Statement": [{
      "Effect":"Allow",
      "Action":"ec2:RunInstances",
      "Resource":"*"
    },
    {
      "Effect":"Allow",
      "Action":"iam:PassRole",
      "Resource":"arn:aws:iam::123456789012:role/Get-pics"
    }]
}
```

How Do I Get Started?

To understand how roles work with EC2 instances, you need to use the IAM console to create a role, launch an EC2 instance that uses that role, and then examine the running instance to see how it's using the role credentials. You cannot directly see the role being used. However, you can see how the role credentials are made available to an instance, 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 PHP in the *AWS SDK for PHP 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, view the following information:

- For more information about using IAM roles with Amazon EC2 instances, go to the *Amazon Elastic Compute Cloud User Guide*.

- To create a role, view Creating a Role (p. 170)
- 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. 155).
- 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 a role. To associate a role to an Amazon EC2 instance, you must use the instance profile name.

If you use the AWS Management Console to create and manage roles, instance profiles are automatically managed for you. If you use the IAM API or CLI to create and manage roles, you must create instance profiles for each role. A role can be associated with many instance profiles, but an instance profile can be associated with only one role.

When you use the Amazon EC2 console to launch an instance with an IAM role, you use the **IAM Role** list. The **IAM Role** list is populated with instance profile names. If you use the AWS Management Console to create a role, the instance profile is created automatically and its name is the same as the name of the role it corresponds to. However, if you use the API, the CLI, or a third-party tool to create roles and instance profiles, the names of your instance profiles might not correspond to the names of your roles. In this case, you need 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.

You can use the following commands to list all the instance profiles in a AWS account, or all the instance profiles with a particular path prefix. The output lists the Amazon Resource Name (ARN) for each instance profile.

- For the CLI, enter `iam-instanceprofilelistbypath`.
- For the API, call `ListInstanceProfiles`.

Use the following commands to list the ARNs of the instance profiles associated with a particular role:

- For the CLI, enter `iam-instanceprofilelistforrole`.
- For the API, call `ListInstanceProfilesForRole`.

Enabling Cross-Account API Access

You are an administrator who uses IAM to control access to your organization's AWS resources. Occasionally, your organization might have resources that users must access across multiple AWS accounts. To allow this access, you can establish trust between accounts so that users from one account can access resources in another account. Using IAM roles, you define trusted entities and the actions they are permitted to do.

Suppose your organization has multiple AWS accounts that you use to isolate different environments. For example, you might isolate a production account from a development account. In the Development account, you let developers work freely with all AWS resources. But in the Production account, you limit developers to read and write access to specific resources.

Because the environments are separated by two different AWS accounts, you can tightly control who can access the Production account and what actions they can do. However, with multiple accounts, you must manage multiple users on multiple accounts, increasing overhead for you and your users.

To avoid this overhead, you can create roles for cross-account API access, and you won't need to create multiple IAM users for each person to access resources in another account. For example, if you want IAM users in the Development account to be able to access resources in the Production account, you can create a role in the Production account that IAM users in the Development account can assume.

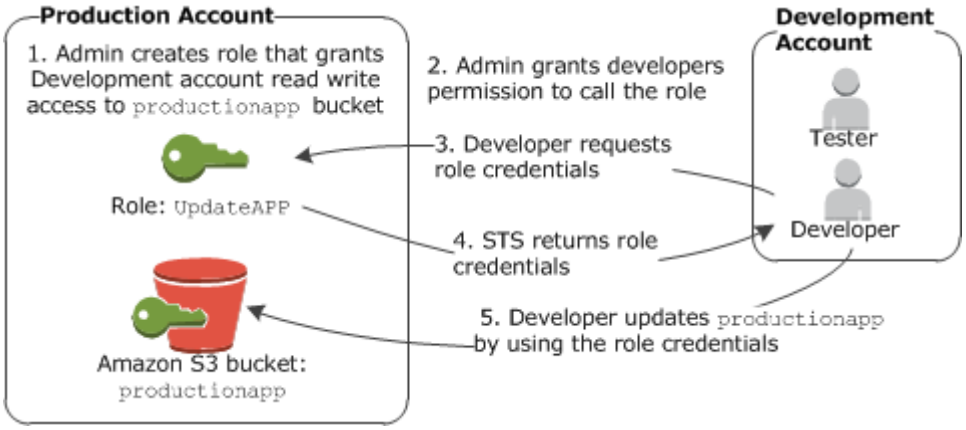
When you create the role, you specify an AWS account from which IAM users can assume the role. You also define the permissions that are available with the role, such as read and write permissions to a specific Amazon S3 bucket.

Important
Only IAM users can assume a role. If you use AWS account credentials, access is denied.

How Does Cross-Account Access with Roles Work?

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.

Cross-account API access



1. An administrator in the Production account uses IAM to create the `UpdateAPP` role. The administrator defines the Development account as a trusted entity, meaning authorized users from the Development account can assume the `UpdateAPP` role. The administrator also defines a policy for the role that specifies that the role has read and write permissions for the `productionapp` bucket.

   The administrator then shares the Amazon Resource Name (ARN) of the role with anyone who needs to assume it. An ARN is automatically associated with a role when a role is created. `arn:aws:iam::123456789012:role/UpdateAPP` is a sample ARN for the `UpdateAPP` role in account number 123456789012.
2. Because only developers are required to update the `productionapp` bucket, the administrator grants the Developer group permission to call the role. The administrator of the Development account must grant the Developer group permissions to call the AWS Security Token Service (AWS STS) `AssumeRole` API for the `UpdateAPP` role. By default, Developers and Testers don't have permission to call `AssumeRole`.
3. A user in the Developer group of the Development account calls `AssumeRole` to obtain the `UpdateAPP` role credentials. The user specifies the `UpdateAPP` ARN 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. AWS STS verifies the request against the role's trust policy to ensure that it's from a trusted entity (Development account). After verification, AWS STS returns temporary security credentials to the developer.
5. The developer uses the temporary security credentials to update the `productionapp` bucket. With the temporary security credentials, the developer can only read and write to the `productionapp` bucket and cannot access any other resource in the Production account.

Related Information

To view a detailed walkthrough of cross-account access, see Enabling Cross-Account API Access Walkthrough (p. 157).

To see a list of services that can use roles, see Using Temporary Security Credentials to Access AWS in *Using Temporary Security Credentials*.

Enabling Cross-Account API Access Walkthrough

In the following walkthrough, you will learn how to delegate access to AWS accounts that you own so that you don't need to manage multiple credentials for each user on different AWS accounts.

Before You Begin

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 each account, application information is stored in Amazon S3 buckets.

The walkthrough assumes that you are an administrator of both accounts. You manage IAM users in the Development account, where you have two IAM groups: Developer and Tester with power-user access (permissions to all actions except IAM). A developer, David, is in the Developer group, and a tester, Teresa, is in the Tester group.

Also, you have an existing Amazon S3 bucket called `productionapp` in the Production account that developers from the Development account will access. The walkthrough shows you how you can use IAM roles to provide cross-account API access to that `productionapp` bucket.

For more information about IAM users and groups or Amazon S3 buckets, see the following:

- Users and Groups (p. 42)
- Create a Bucket with Amazon Simple Storage Service in the *Amazon Simple Storage Service Getting Started Guide*.

Overview

From time to time, David must update the live applications in the Production account; however, you don't want to create another IAM user for David in the Production account. As your organization grows, creating multiple IAM users for each developer will be costly to manage. Instead, you can delegate API 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.

What You Will Accomplish

By the end of the walkthrough, you will have a role in the Production account that allows users from the Development account to access the `productionapp` bucket in the Production account. After assuming the role, David uses the assumed role credentials to access the `productionapp` bucket in the Production account. A similar call by Teresa to assume the role fails.

The following list outlines the steps that you will complete:

1. Creating a Role (p. 158)

   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 will define the Development account as a trusted entity and specify a role policy that allows trusted users to only update the `productionapp` bucket.
2. Modifying Group Permissions (p. 161)

   Modify the IAM group policy so that testers are denied access to the UpdateAPP role. In this walkthrough, the permissions provided by power-user access allows Developers to assume the role. Testers, who have similar permissions, will be denied permission to assume the role.
3. Assuming a Role (p. 163)

   As the IAM user David, assume the `UpdateAPP` role. Using the assumed role credentials, you can update the `productionapp` bucket in the Production account.

Creating a Role

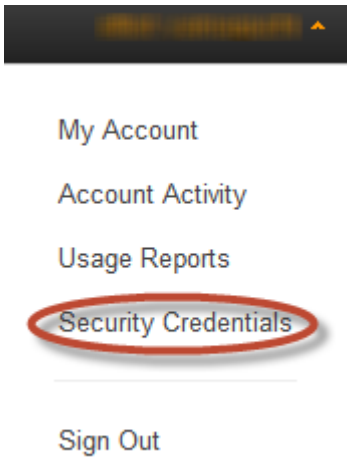
To allow users from one AWS account to access resources on another AWS account, create a role that defines who can access the account and what permissions are available on the role.

You create the role on 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. As long as David is granted permission to assume the role, he can read and write to the `productionapp` bucket by using the assumed role's temporary security credentials.

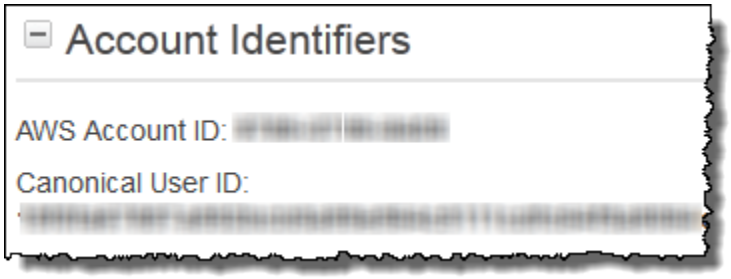
Before you can create a role, you need the account ID of the Development account. The account ID is a unique identifier that is unique to each AWS account.

To obtain the Development account ID

1. From the AWS website, click **My Account/Console** and then sign in to the AWS Management Console for the Development account.
2. In the banner of the console, click your user name and then select **Security Credentials**.



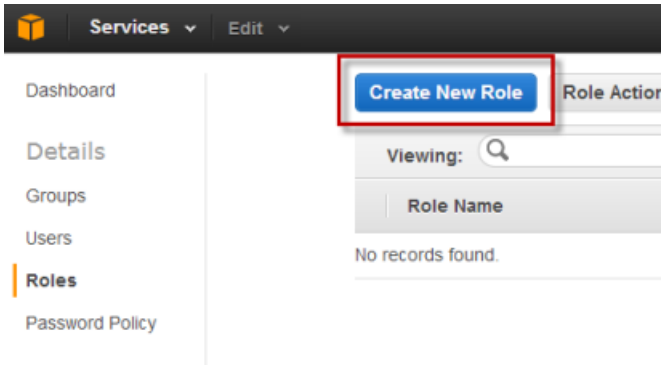
3. On the Security Credentials page, expand the **Account Identifiers** section to view your account ID.



The account ID is a 12-digit number. For this scenario, the Development account ID is 123456789012; however, you should use a valid account ID to avoid any errors.

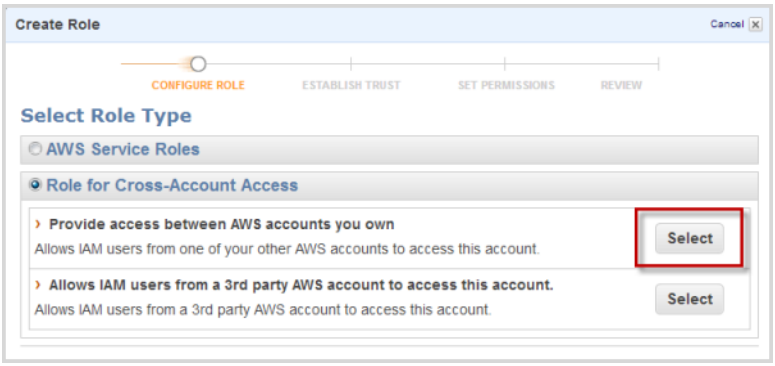
To create a role

1. Sign in to the AWS Management Console as an administrator of the Production account.
2. Navigate to the IAM console.
3. In the navigation pane, click **Roles**.
4. Click **Create New Role**.



5. Enter **`UpdateAPP`** for the role name, and then click **Continue**.

6. Click **Roles for Cross-Account Access**, and then select **Provide access between AWS accounts you own**.



7. Enter the Development account ID.

   For this walkthrough, we're using the example account ID **`123456789012`** for the Development account. However, you should use a valid account ID. (If you use **`123456789012`** as the account ID, IAM will not let you create the new role.)
8. Click **Continue** to set the permissions that will be associated with the role.
9. Select **Custom Policy** to create read and write permissions for the `productionapp` bucket.

   You want to set read write access to the `productionapp` bucket. Although IAM provides some Amazon S3 policy templates, one is not provided for read write access to an Amazon S3 bucket, so you can create your own policy instead.
10. Enter **`read-write-app-bucket`** for the policy name.
11. Add the following permissions to the policy document:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket"
      ],
      "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 and update the contents within the `productionapp` bucket.

12. Click **Continue** to review the role before it's created.
13. After reviewing the role, click **Create Role**.

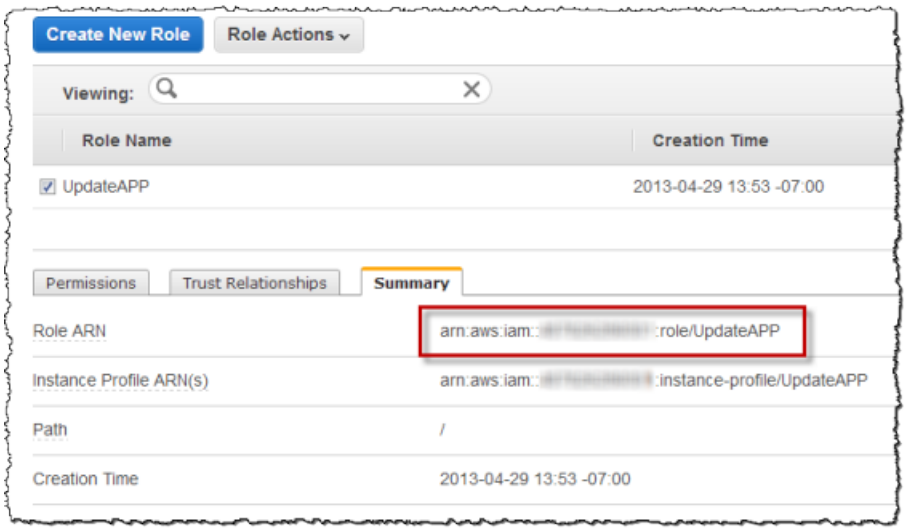
    The `UpdateAPP` role is displayed in the list of roles.

Now, you must obtain the role's Amazon Resource Name (ARN), which is a unique identifier for the role. When you modify the Tester group's policy, you will specify the role's ARN to deny testers from assuming this role.

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** tab for the `UpdateAPP` role, record the Role ARN value.

   The Production account has an account ID of 123123123123, so the role ARN is arn:aws:iam::123123123123:role/UpdateAPP.



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. You also defined what users from the Development account can do if they are allowed to assume the `UpdateAPP` role.

Next, modify the permissions for the Tester group.

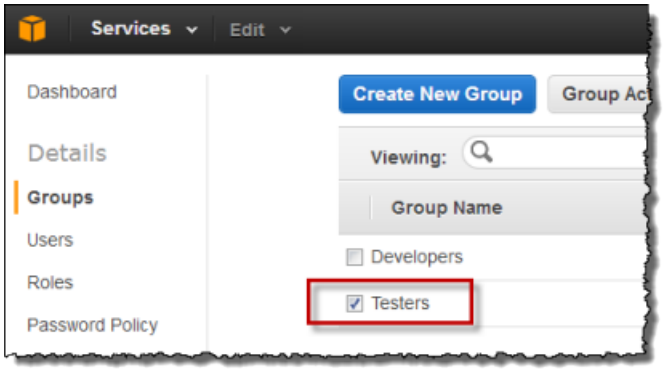
Modifying Group Permissions

Because testers and developers have power-user access, they can freely test applications in the Development account. They have permission to all actions except for IAM actions so that they cannot create or modify permissions. With power-user access, the Developer group has the necessary permissions

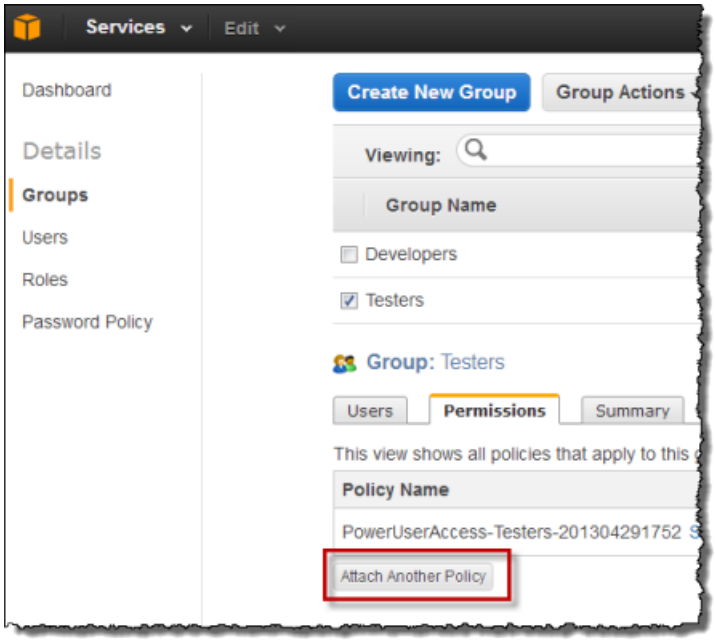
to assume the `UpdateAPP` role. However, the Testers group's policy must be modified so that it doesn't have access to the `UpdateAPP` role.

To modify the Testers group

1. Log in as an administrator in the Development account.
2. Navigate to the IAM console.
3. Click **Groups**.
4. From the list of groups, click **Testers**.



5. Click the **Permissions** tab for the Testers group.
6. Click **Attach Another Policy**.



7. Select **Custom Policy**.
8. (Optional) You can rename the policy by editing the **Policy Name** field.
9. Add the following policy statement to deny the assume role action on the `UpdateAPP` role:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": "sts:AssumeRole",
      "Resource": "arn:aws:iam::123123123123:role/UpdateAPP"
    }
  ]
}
```

The deny permission explicitly denies the Testers group access to the `UpdateAPP` role. Any tester who tries to access the role will get an access denied message. However, because you specified a specific role ARN, testers can still assume other roles.

10. Click **Apply Policy** to add the policy to the Tester group.

Summary and what's next?

The Developer group has access to assume the `UpdateAPP` role. The Testers group had a similar policy, but you modified the Testers group policy by adding a deny permission that prevents them from accessing the `UpdateAPP` role.

Next, you'll learn how David can access the `productionapp` bucket in the Production account by using the `AssumeRole` API call.

Assuming a Role

When David needs to make an update in the Production account, he makes an `AssumeRole` call to assume the `UpdateAPP` role. AWS Security Token Service (STS) returns temporary credentials that he can use to access the `productionapp` bucket in the Production account. With those credentials, David can update the `productionapp` bucket but cannot access any other resources in the Production account, even though he has power-user privileges in the Development account.

Important
Only IAM users can assume a role. If you use AWS account credentials, access is denied.

To assume a role

1. David calls `AssumeRole` as part of an application. He must specify the `UpdateAPP` ARN: `arn:aws:iam::123123123123:role/UpdateAPP`.

   The response from the `AssumeRole` call includes the temporary credentials. A similar call by Teresa would fail.
2. With the temporary credentials, David makes an `s3:PutObject` call to update the `productionapp` bucket.

   Because the temporary role credentials have only read-write access to the `productionapp` bucket, any other actions in the Production account are denied.

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 do. Then, you modified a group policy

to limit 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.

Delegating API Access to Third Parties

When third parties require access to your organization's AWS resources, you can use roles to delegate API 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.

> **Note**
> When you grant third parties access to your AWS resources, they can access any resource that you give them permissions to. In addition, their use of your resources is billed to you. Make sure that you limit their use of your resources appropriately.

Third parties must provide you the following information for you to create a role that they can assume:

- The AWS account ID that the third party's IAM users use to assume your role. You specify their AWS account ID when you define the trusted entity for the role.
- An external ID that the third party can associate you with your role. You specify the ID that was provided by the third party when you define the trusted entity for the role.
- The permissions that the third party requires in order to work with your AWS resources. You specify these permissions when defining the role's permission policy. This policy defines what action they can do 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 assume the role.

Related Information

- For more information about the external ID, see About the External ID.
- For a detailed walkthrough about creating a role with the AWS Management Console, see Creating a Role (p. 158).

Cross-Account Access Using Resource-Based Policies

Topics

You can enable trusted entities outside of your AWS account to access certain resources within your AWS account. In addition, when the trusted entity is another AWS account, that account can delegate access to its AWS Identity and Access Management (IAM) users. This process is referred to as *cross-account access*. Cross-account access enables you to share access to your resources with users under other AWS accounts.

Cross-account access is a two-step process. First you give another AWS account access to specific resources, and then that AWS account delegates access to its users. An AWS account can delegate access only to the extent that it has been granted access. This section links to information about enabling access to particular AWS account resources, and then describes how to use IAM to delegate access to users within an account.

Access to AWS Resources

Certain AWS services enable you to grant access to your AWS resources to trusted entities outside of your AWS account. For example, you can use Amazon S3 ACLs and policies to grant other AWS accounts access to your Amazon S3 buckets.

The following table shows the AWS services that use policies to control access to resources, lists the resources available, and provides links to detailed information about how to use policies with each service.

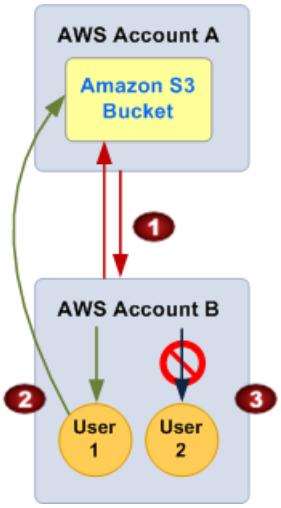
Note
You can also enable cross-account access by using IAM roles. With a single role, you can manage access and permissions to multiple AWS services. IAM roles also supports additional AWS services that are not available with resource-based policies. For more information about using roles for cross-account access, see Enabling Cross-Account API Access (p. 155). For a list of services that support roles, see Using Temporary Security Credentials to Access AWS in *Using Temporary Security Credentials*.

AWS service	Resource	Reference
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 Appendix: Managing Access to Your Amazon SNS Topics in the *Amazon Simple Storage Service Getting Started Guide*.
Amazon SQS	queues	For more information, go to Appendix: The Access Policy Language in the *Amazon Simple Queue Service Developer Guide*.

Important
Give access only to entities you trust, and give the minimum amount of access necessary. Granting access to an AWS account enables it to delegate access to its users. An AWS account can delegate access only to the extent that it has access.

Delegating AWS Account Permissions to IAM Users

An AWS account with access to another AWS account's resources can use an IAM policy to delegate access to the users under its account, as described in the following figure and table. (For information about permissions, policies, and the access policy language you use to write policies, see Permissions and Policies (p. 107).)



1. Account A gives Account B full access to Account A's Amazon S3 bucket. As a result, Account B can perform any action on Account A's bucket, and Account B can grant access to users under Account B.
2. Account B gives User 1 read 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 it has.
3. Account B does not give access to User 2. Because User 2 has not been granted access to Account A's Amazon S3 bucket by Account B, by default, User 2 cannot access the bucket or the objects in the bucket.

> **Important**
> In the preceding example, be aware that if Account B had used wildcards (*) to give a user full access to all its resources, that user would automatically have access to any resources that Account B has access to, including access granted by another account to its own resources. In this case, the user would have access to Account A's resources even though Account B did not specifically apply the permission.
> Policies are late-binding. This means that 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.

How to Delegate AWS Account Permissions

To delegate permissions to users under your AWS account, you attach a policy to the user or group that you want to delegate your permissions to. You can delegate permissions equivalent to, or less than, the permissions you have.

For example, if your account has full access to the resources of another AWS account, you can delegate full access, list access, or any other partial access to users under your AWS account. If you have been granted list access only, you can delegate only list access. This means that where you have list access only, but you grant users under your AWS account full access, your users will have list access only. (For information about attaching a policy to a user or group, see Managing IAM Policies (p. 112).)

Topics

Example: Use a policy to delegate access to another AWS account's Amazon S3 bucket

In this example, Account A uses a 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 access to Account A's bucket to one of the users under Account B.

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" : {
      "Effect":"Allow",
      "Sid":"AccountBAccess1",
      "Principal" : {
          "AWS":"111122223333"
      },
      "Action":"s3:*",
      "Resource":"arn:aws:s3:::mybucket/*"
   }
}
```

Note that, alternatively, Account A could 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 this case, the only thing that would change is how Account A grants access to Account B. Account B would still use a policy to delegate access to a user under 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 under 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) under Account B.

```
{
   "Version": "2012-10-17",
   "Statement":{
      "Effect":"Allow",
      "Action":"s3:List*",
      "Resource":"arn:aws:s3:::mybucket/*"
   }
}
```

Example: Delegate access to another AWS account's Amazon SQS queue

In this example, Account A has an Amazon SQS queue and Account A uses a queue policy to grant queue access to Account B. Then, Account B uses an IAM user policy to delegate access to a user under Account B.

The following example queue policy gives Account B `SendMessage` and `ReceiveMessage` permission for Account A's queue named *1234-5678-9012/queue1*, but only between noon and 3:00 p.m. on November

30, 2011. 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",
      "Action": ["sqs:SendMessage","sqs:ReceiveMessage"],
      "Principal": {
        "AWS": "111122223333"
      },
      "Resource": "arn:aws:sqs:*:/123456789012:queue1",
      "Condition": {
        "DateGreaterThan": {
          "aws:CurrentTime": "2011-11-30T12:00Z"
        },
        "DateLessThan": {
          "aws:CurrentTime": "2011-11-30T15:00Z"
        }
      }
   }
}
```

Account B's policy delegating access to a user under 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 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, 2011, and the user can only perform `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 under Account B access to Account A's bucket. The explicit deny applied to Account A's Amazon S3 bucket propagates to the users under Account B and overrides the IAM user policy granting access to the user under Account B. (For detailed information how permissions are evaluated, see IAM Policy Evaluation Logic (p. 145).)

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" : {
      "Effect":"Deny",
      "Sid":"AccountBDeny",
      "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 under Account B.

```
{
   "Version": "2012-10-17",
   "Statement":{
      "Effect":"Allow",
      "Action":"s3:*",
      "Resource":"arn:aws:s3:::mybucket/*"
   }
}
```

Because Account A's bucket policy explicitly denies Account B access to *mybucket*, the denial propagates to the user under Account B, and Account B's IAM user policy granting Account B's user access to Account A's bucket has no effect. The user cannot access Account A's bucket.

Delegating API Access Within an AWS Account

For mission critical permissions that IAM users might not frequently use, you can separate those permissions from their normal day-to-day permissions by using roles. Users would then have to actively assume a role, which can prevent them from accidentally performing disruptive actions.

For example, you might have Amazon EC2 instances that are critical to your organization. Instead of directly granting administrators permission to terminate the instances, you can create a role with those privileges and allow administrators to assume the role. Administrators won't have permission to terminate those instances; they must first assume a role. By using a role, administrators must take an additional step to assume a role before they can stop an instance that's critical to your organization.

How Do I Get Started?

When you create a role, specify the AWS account ID for the account that you are creating the role in so that IAM users within the account can assume the role. Then, you must grant assume role permissions to IAM users who can assume the role.

Related Information

For more information about creating and assuming roles, see the following topics:

- For a detailed walkthrough about creating a role with the AWS Management Console, see Creating a Role (p. 158).
- To assume a role, see Assuming a Role (p. 181).

Creating a Role

You can create a role by using the AWS Management Console, CLI, or the API. If you use the CLI or API to create roles for Amazon EC2 instances, you must also create an instance profile and add the role to the instance profile.

Topics

> **Note**
> Role names have character limitations. The number of roles and policy size are also limited. For more information, see Limitations on IAM Entities (p. 16). After you create a role, you cannot rename it.

Create a Role (AWS Management Console)

If you use the AWS Management Console to work with IAM, you can create a role for an AWS service, for an AWS account, or for a federated user by using the **Create New Role** wizard.

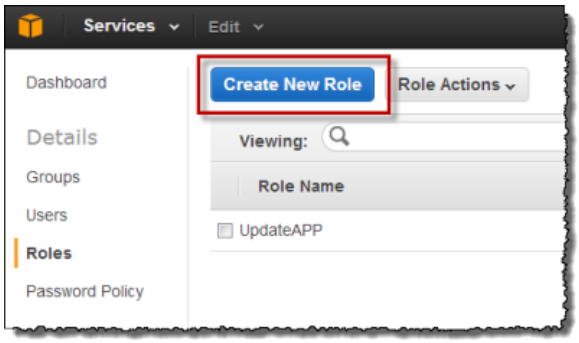
Topics

Creating a Role for an AWS Service

You create a role for an AWS service when you are working with an AWS service like Amazon EC2, AWS Data Pipeline, Amazon Elastic Transcoder, or AWS OpsWorks. These services manage other AWS resources, so you create a role that determines what the service is allowed to do with those resources. For more scenarios, you can select a policy template that contains predefined permissions. However, if you have requirements that are not covered by a policy template, you can create a custom policy (or edit a predefined one).

To create a role for an AWS service

1. In the navigation pane, 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.

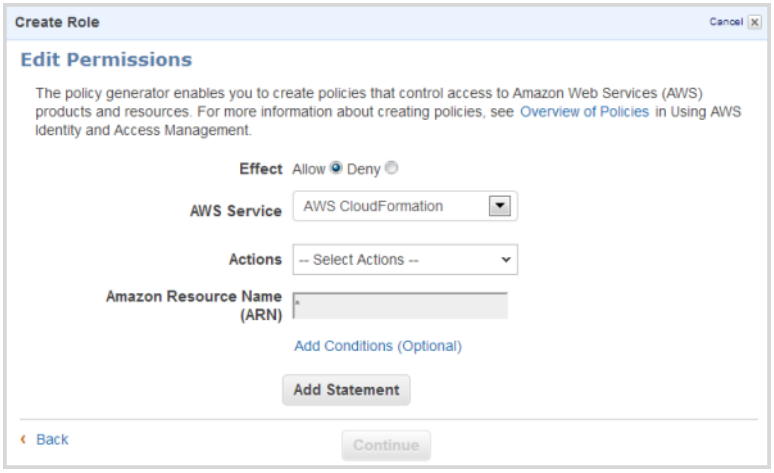
   Because various entities might reference the role, you cannot edit the name of the role after it has been created.
3. Click **AWS Service Roles**, and then select the service that you will allow to assume this role.
4. Depending on the role that you selected, review the predefined policy or create a policy.

   a. If the role included a predefined policy, you can modify the policy name or policy document, and then click **Continue** to review the role.
   b. If the role that you selected doesn't include a predefined policy, select a method for creating the policy document by clicking **Select Policy Template**, **Policy Generator**, or **Custom Policy**.

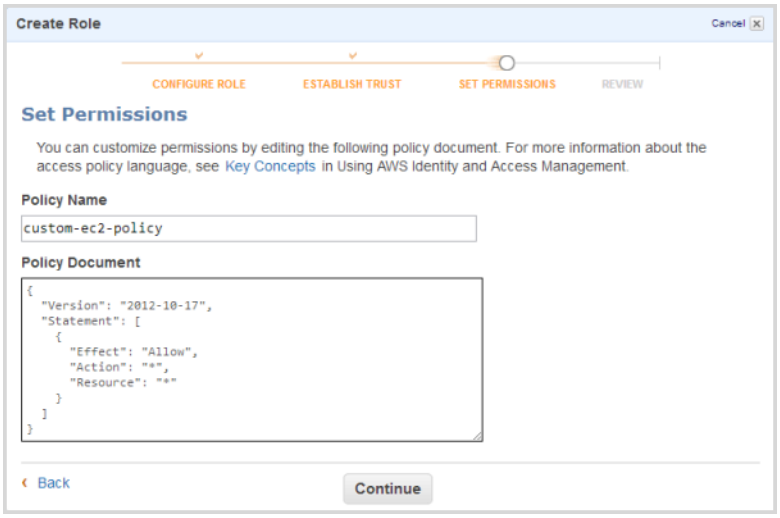
      - Policy templates are predefined policies that have one or more permissions specified. If you are specifying permissions that match or are related to a template, select the template and then make any modifications on the next screen.
      - The policy generator helps you create permissions for a policy by providing drop-down menus where you can select services, actions, conditions, and keys. The generator creates the policy document for you.
      - If you want to fully write and customize a policy, select **Custom Policy**.

   c. How you complete the next step depends on the method you selected to create the policy.

      - If you are using a template to create the policy, review the policy content in the dialog box.
      - If you are using the policy generator, select the appropriate **Effect**, **AWS Service**, and **Actions** options, enter the 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 **Continue**.



- If you are using a custom policy, enter a name for the policy under **Policy Name** and write the policy or paste the policy document from your text editor into the **Policy Document** box.



Note
There are limitations on policy names and on policy size. For information about policy limitations, see Limitations on IAM Entities (p. 16).

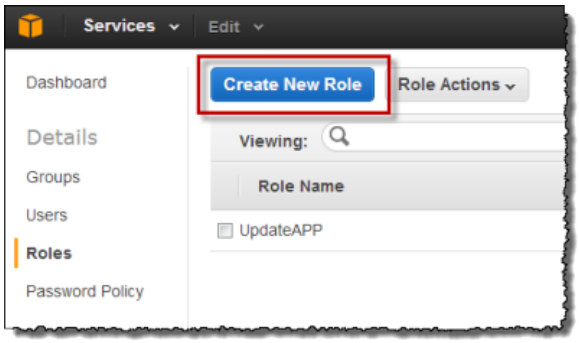
5. Click **Continue** to review the role.
6. After reviewing the role, click **Create Role** to complete the wizard.

Creating a Role That IAM Users Can Assume

You create a role that IAM users can assume when you want to configure cross-account access, where an IAM user in one account is allowed to access resources that belong to a different account.

To create a role that IAM users can assume

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.

   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.
4. Specify an AWS account ID that you want to establish trust with.

   Any IAM user from the trusted AWS account can assume this role, as long as they are granted permission to call the role.
5. If you selected a role for a third party, enter the external ID provided by the third party.

   For more information about the external ID, see About the External ID.
6. Click **Continue**.
7. Set the permissions for the role to specify what actions can be done on specific resources (similar to setting permissions for IAM groups).

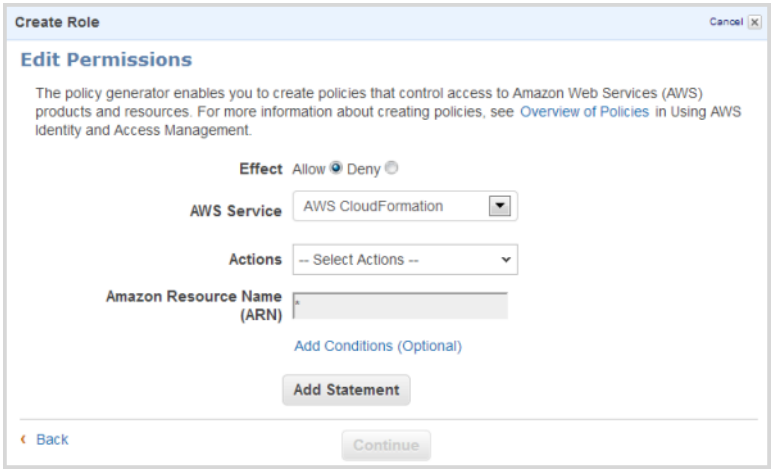
   The permissions that you specify are available to entities that assumes the role. By default, roles have no permissions.

   a. Choose the method for creating the policy document by clicking **Select Policy Template**, **Policy Generator**, or **Custom Policy**, and then click **Select**.

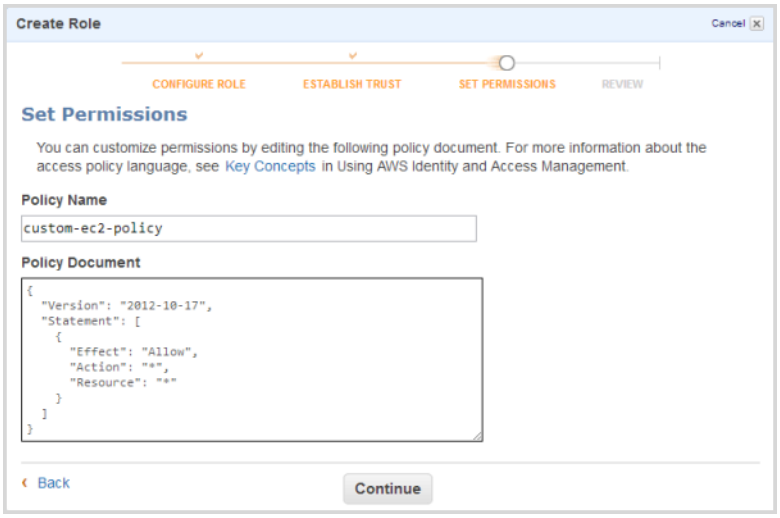
      - Policy templates are predefined policies that have one or more permissions specified. If you are specifying permissions that match or are related to a template, select the template and then make any modifications on the next screen.
      - The policy generator helps you create permissions for a policy by providing drop-down menus where you can select services, actions, conditions, and keys. The generator creates the policy document for you.
      - If you want to fully write and customize a policy, select **Custom Policy**.

   b. How you complete the next step depends on the method you selected to create the policy.

      - If you are using a template to create the policy, review the policy content in the dialog box.
      - If you are using the policy generator, select the appropriate **Effect**, **AWS Service**, and **Actions** options, enter the 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 **Continue**.



- If you are using a custom policy, enter a name for the policy under **Policy Name** and write the policy or paste the policy document from your text editor into the **Policy Document** box.



Note
There are limitations on policy names and on policy size. For information about policy limitations, see Limitations on IAM Entities (p. 16).

c. Click **Continue** to review the role.

8. After reviewing the role, click **Create Role** to complete the wizard.

Creating a Role for Web Identity Federation

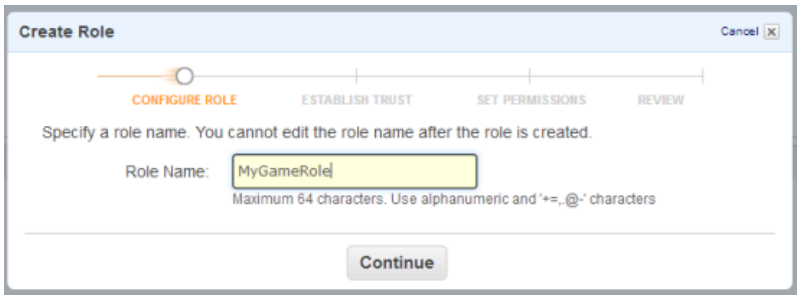
Web identity federation lets you provide access to AWS resources for users who have signed in using a third-party identity provider like Amazon, Facebook, or Google. To configure web identity federation, you create a role that determines what permissions the federated user will have.

If you want to support more than one identity provider, you must create multiple roles, one per provider. For example, if you want to support Login with Amazon, Facebook, and Google, you must create three roles, and for each role indicate the provider that the role is for.

Before you can create a role for web identity federation, you must register your application with the identity provider, who will give you an application ID. For more information, see Creating Temporary Security Credentials for Mobile Apps Using Third-Party Identity Providers in the *Using Temporary Security Credentials* guide.

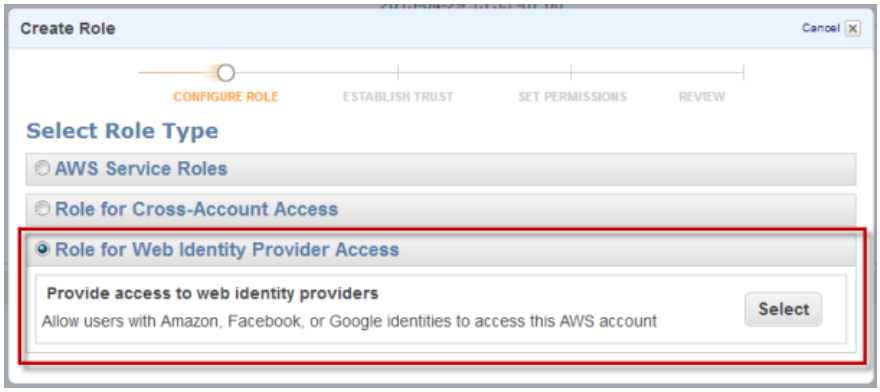
To create a role for web identify federation

1. In the navigation pane, 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.

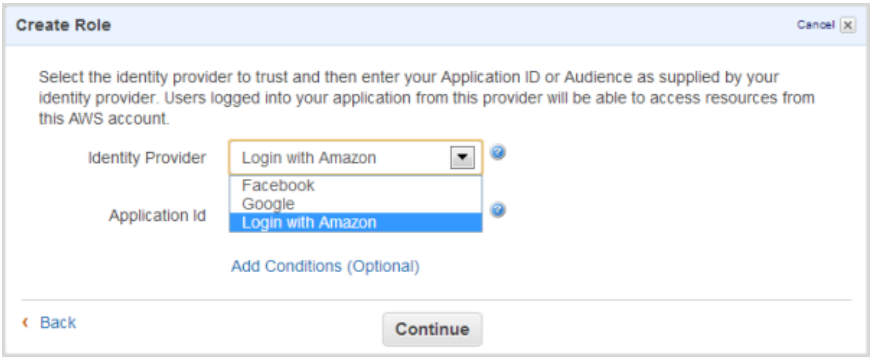


Because various entities might reference the role, you cannot edit the name of the role after it has been created.

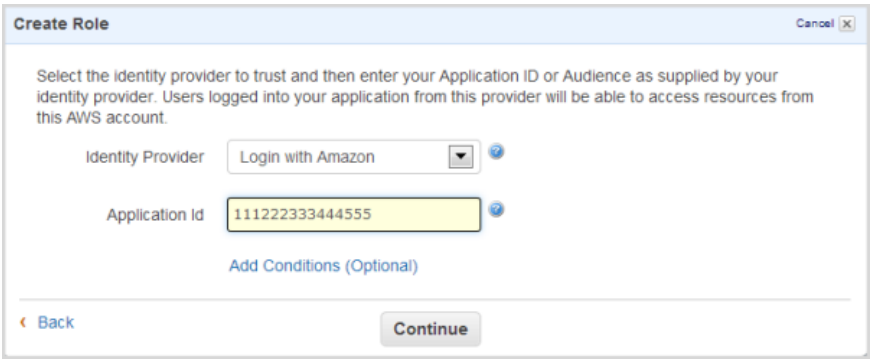
3. Click **Roles for Web Identity Provider Access**, and then click **Select**.



4. In the **Identity Provider** list, select the identity provider that you're creating the role for. Remember that you must create a separate role for each identity provider that you want to support.



5. In the **Application ID** box, enter the ID that the provider gave you when registered your application with the provider.



   Important
   If you do not enter an application ID, any user who is logged into the provider will be able to assume the role and get access to the resources that are allowed by the role's policy.

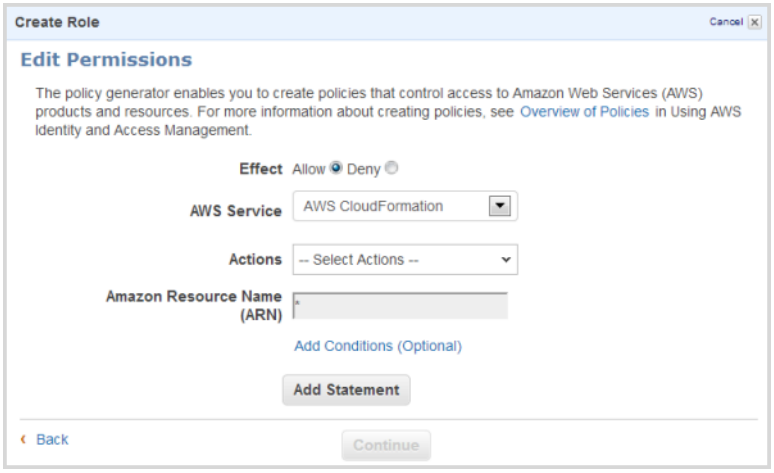
6. Optionally, click **Add Conditions** to create additional conditions for permissions granted by the role. For example, you can add a condition that grants access to AWS resources only for a specific user ID. (The condition to check the application ID is generated automatically as part of the policy template.)
7. Click **Continue**.
8. Set the permissions that determine what the federated user will be allowed to do. By default, roles have no permissions.

   a. Choose the method for creating the policy document by clicking **Policy Generator** or **Custom Policy**, and then click **Select**.

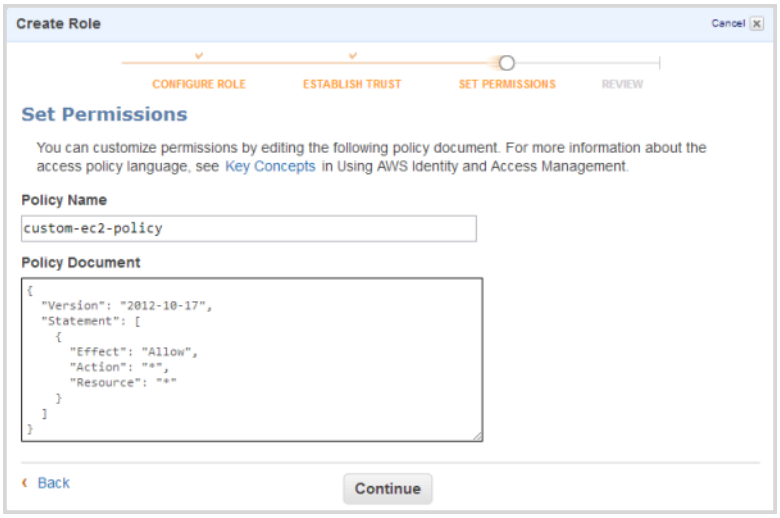
      - The policy generator helps you create permissions for a policy by providing drop-down menus where you can select services, actions, conditions, and keys. The generator creates the policy document for you.
      - If you want to fully write and customize a policy, select **Custom Policy**.

   b. How you complete the next step depends on the method you selected to create the policy.

      - If you are using the policy generator, select the appropriate **Effect**, **AWS Service**, and **Actions** options, enter the 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 **Continue**.



- If you are using a custom policy, enter a name for the policy under **Policy Name** and write the policy or paste the policy document from your text editor into the **Policy Document** box.



Note
There are limitations on policy names and on policy size. For information about policy limitations, see Limitations on IAM Entities (p. 16).

c. Click **Continue** to review the role.

9. After reviewing the role, click **Create Role** to complete the wizard.

Create a Role (CLI)

This topic shows the process for using the IAM CLI to create a role.

To create a role with the CLI

1. Create a role by entering the following command: `iam-rolecreate -r role_name [-f policy_document_file | -s service_endpoint`.

   For the role's trust policy, you can specify a file location or a service endpoint. The policy document file can contain multiple entities that can assume the role and any conditions for assuming the role.
2. Associate a policy to the role by entering the following command: `iam-roleaddpolicy -r role_name -p policy_name -e Allow_or_Deny {-a action ...} {-c ARN ...}`.
3. If you are launching an Amazon EC2 instance with this role, create and add the role to an instance profile by entering the following command: `iam-instanceprofilecreate -r role_name -s instance_profile_name`.

   An instance profile is a container for a role. Each instance profile can contain only one role. For more information about instance profiles, see Instance Profiles (p. 155).

Create a Role (API)

This topic shows the process for using the IAM API to create a role.

To create a role with the API

1. Create a role by calling `CreateRole` with the role name and role trust policy.
2. Associate a policy to the role by calling `PutRolePolicy` with the policy document, policy name, and role name.
3. If you are launching an Amazon EC2 with this role, you must complete the following steps:

   a. Create an instance profile by calling `CreateInstanceProfile` with the instance profile name.
   b. Add the role to the instance profile by calling `AddRoleToInstanceProfile` with the role name and instance profile name.

      An instance profile is a container for a role. Each instance profile can contain only one role. For more information about instance profiles, see Instance Profiles (p. 155).

Related Information

For information about how to launch an instance with the role you just created, see Using IAM roles with Amazon EC2 instances in the *Amazon Elastic Compute Cloud User Guide*.

For information about cross-account access with roles, see Enabling Cross-Account API Access (p. 155).

For information about managing policies, see Managing IAM Policies (p. 112).

For more information about IAM CLI commands or API actions, see *AWS Identity and Access Management Command Line Interface Reference* or *AWS Identity and Access Management API Reference*.

Modifying a Role

You can edit who can access a role and the permissions associated with the role by using the AWS Management Console, CLI, or API. Any changes to the role are propagated in near realtime.

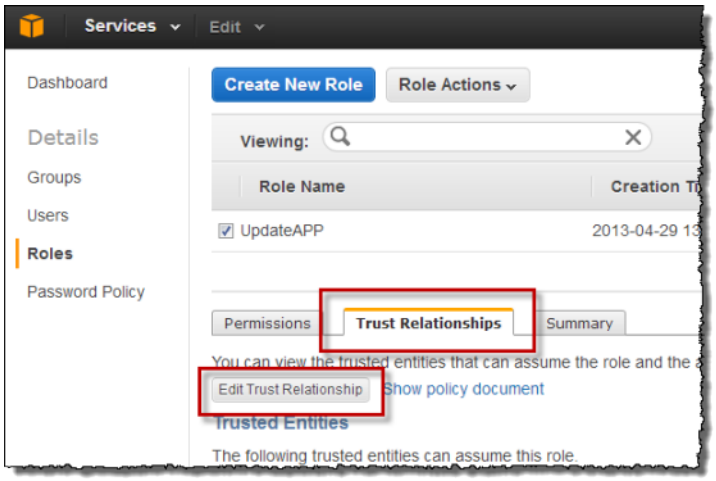
Topics

Modify a Role (AWS Management Console)

To edit who can access the role

1. In the navigation pane of the IAM console, click **Roles**.

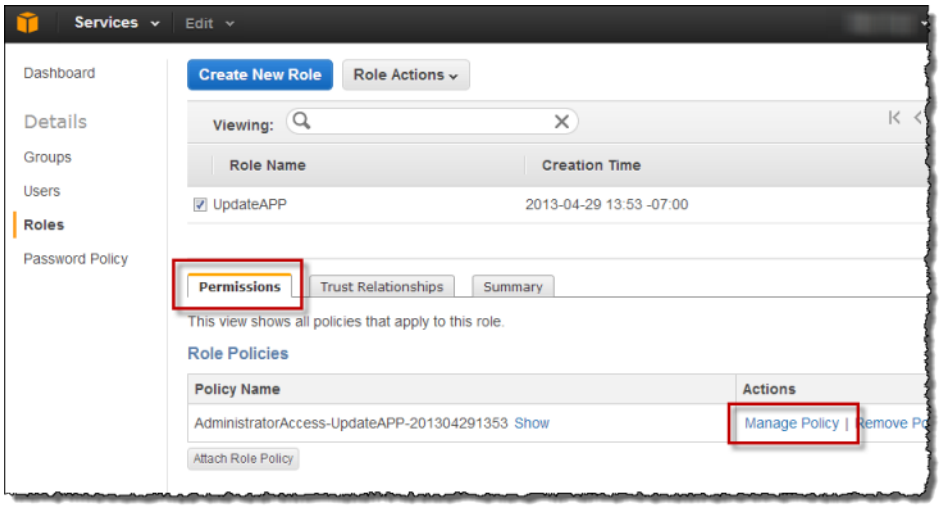
   A list of roles for your account is displayed in the **Roles** pane.
2. Select the role that you want to modify, and then in the properties pane for the role, click the **Trust Relationship** tab.
3. In the **Trust Relationship** tab, click **Edit Trust Relationship**.



4. Edit, add, or remove trusted entities, and then click **Continue**.
5. Edit, add, or remove conditions for assuming the role, and then click **Continue**.
6. Review your changes, and then click **Save Role**.

To edit the permissions associated with a role

1. In the navigation pane of the console, click **Roles**.
2. Select the role that you want to modify, and in the properties pane for the role, click the **Permissions** tab.
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**.

Modify a Role (CLI)

To edit who can access the role

1. If you don't know the name of the role that you want to modify, list the roles in your account by entering the following command: **`iam-rolelistbypath`**.

   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 had 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, enter the following command: **`iam-rolegetattributes -r myrole`**.
3. Create a text file with the updated trust 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.
4. To update the trust policy, enter the following command: **`iam-roleupdateassumepolicy -r myrole -f policy_document_file`**.

   Changes to the role are propagated in near real time. You can verify that the role has been updated by entering the following command: **`iam-rolegetattributes -r myrole`**.

To edit the permissions associated with a role

1. If you don't know the name of the role that you want to modify, list the roles in your account by entering the following command: **`iam-rolelistbypath`**.

   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 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, enter the following command: **`iam-rolelistpolicies -r myrole -v`**.
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, enter the following command: **`iam-roleuploadpolicy -r myrole -p policy_name -f policy_document_file`**.

   Changes to the role are propagated in near real time. You can verify that the role has been updated by entering the following command: **`iam-rolelistpolicies -r myrole`**.

Modify a Role (API)

To edit who can access the role

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 realtime. You can verify that role has been updated by calling **`GetRole`**.

To edit the permissions associated with a role

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.

Assuming a Role

If your AWS account is a trusted entity for a role, you can assume that role as an IAM user to access AWS resources that you might not have explicit permissions to. When you assume a role, you receive temporary security credentials that you can use to sign requests to access AWS resources. The permissions of the temporary security credentials are defined in the policy associated with the role.

> **Important**
> Only IAM users can assume a role. If you use AWS account credentials, access is denied.

Before you can assume a role, you must ensure that role includes your AWS account as a trusted entity and that you have permission to call the AWS Security Token Service (STS) AssumeRole API. You must also have the Amazon Resource Name (ARN) for the role.

To assume a role

- Call `AssumeRole` and pass the role ARN.

  Depending on the conditions that are set in the role, you might also need to pass additional parameters, such as an external ID.

  You can also include a supplemental policy that narrows the permissions of the role's temporary security credentials. By narrowing the permissions, you can maintain the least amount of privileges.

In the response for `AssumeRole`, you receive temporary security credentials that you can use to access AWS resources that are defined by role.

Related Information

- For a detailed scenario about using roles to delegate access, see Enabling Cross-Account API Access Walkthrough (p. 157).
- To see a list of services that can use roles, see Using Temporary Security Credentials to Access AWS in the *Using Temporary Security Credentials* guide.
- For more information about the external ID, see About the External ID.

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.

You can also remove roles from instance profiles.

> **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.

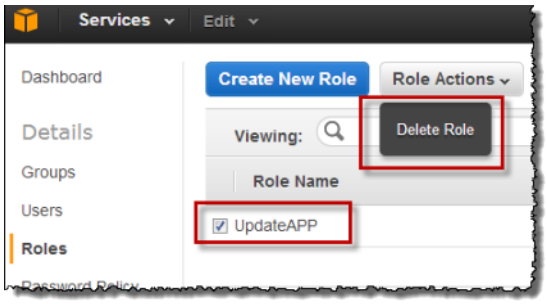
Topics

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 the instance profile that contains the role.

To delete a role

1. In the navigation pane of the IAM Dashboard, click **Roles**.
2. Select the role 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 Delete a Role (CLI and API) (p. 183).

Delete a Role (CLI and API)

When you use the IAM CLI or 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

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: **`iam-rolelistbypath`**.

   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 had 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: **`iam-instanceprofilelistforrole -r myrole`**.
   b. To remove the role from each instance profile, for each instance profile, enter the following command: **`iam-instanceprofileremoverole -r myrole -s instance_profile_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: **`iam-rolelistpolicies -r myrole`**.
   b. To delete each policy from the role, for each policy, enter the following command: **`iam-roledelpolicy -r myrole -p policy_name`**.

4. Delete the role by entering the following command: **`iam-roledel -r myrole`**.
5. If you are not using the instance profiles that were associated with the role, you can delete them by entering the following command: **`iam-instanceprofiledel -s instance_profile_name`**.

To delete a role

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 using 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. 155).

IAM and the AWS Management Console

Topics

The AWS Management Console provides a web-based interface for many AWS product APIs (such as the API for Amazon S3, Amazon EC2, and so on). Users can make requests directly to an API, or users can use the AWS Management Console to make requests to a service's API. Users who work with your AWS resources through the AWS Management Console will use your AWS account's IAM-enabled sign-in page and the passwords you create for them to access the console.

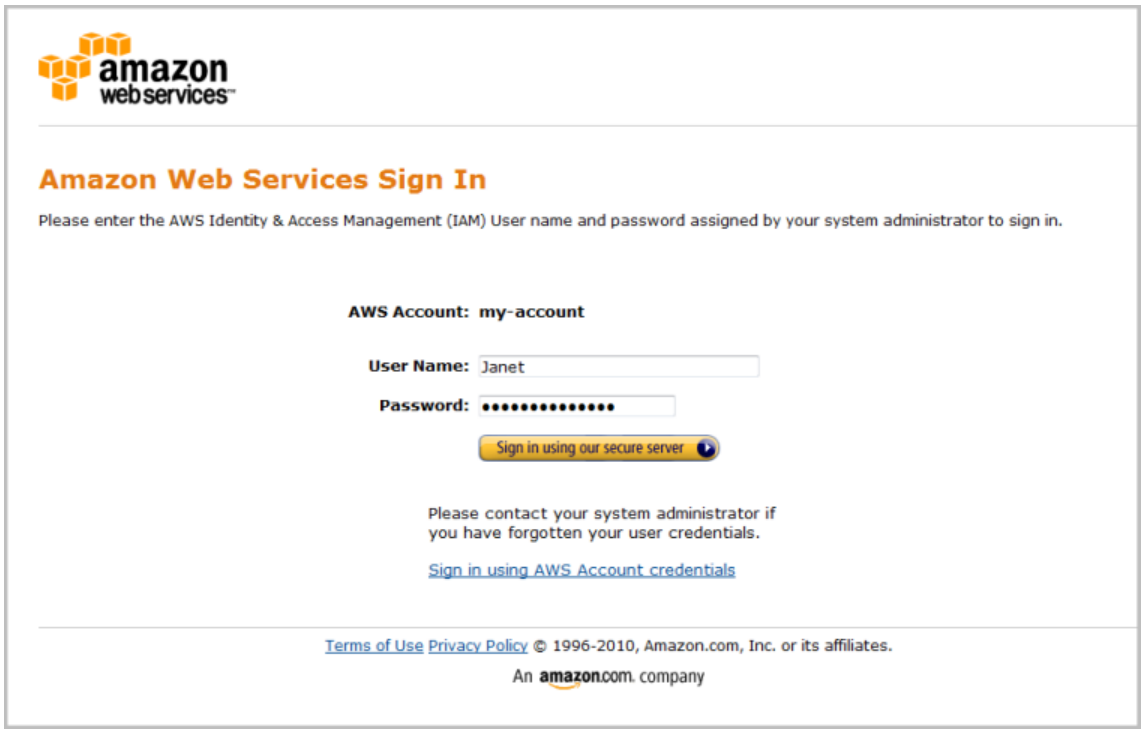
This section provides information about the IAM-enabled AWS Management Console sign-in page and explains how to create an AWS account alias for your sign-in page. For information about creating user passwords, see Managing Passwords (p. 63).

> **Note**
> Some AWS products are not supported on the AWS Management Console. For a list of AWS products that are supported on the console, go to AWS Management Console.

The AWS Management Console Sign-in Page

Users who use the AWS Management Console must sign in to your AWS account through the IAM-enabled sign-in page. You provide your users with the URL they need to access the sign-in page. You might consider sending the link through email to the users who need it, or you might create a link to the sign-in page from a page in your company intranet.

Your IAM-enabled AWS Management Console sign-in page will look similar to the following page.



Important
In addition to providing users with a URL to your IAM-enabled sign-in page, for users to 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 Managing Passwords (p. 63) and Using Multi-Factor Authentication (MFA) Devices with AWS (p. 76).

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

```
https://your_AWS_Account_ID.signin.aws.amazon.com/console/ec2
```

Note
Your AWS account ID is the same as your account number, but without hyphens. To locate your AWS account number, go to the AWS Manage Your Account page. Your account number is located near the top right corner of the page.

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. For more information about AWS account ID aliases, see Using an Alias for Your AWS Account ID (p. 188).

Note
When a user signs in, the console opens to the console referenced by the abbreviation at the end of your URL. In the preceding sample, signing in opens the Amazon EC2 console. If the URL ended with *s3*, then the AWS Management Console would open to the Amazon S3 console.

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

When users sign in to your AWS account, they sign in via an IAM-enabled user 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 calls the IAM-enabled user sign-in page by default.

If you want to sign in to the AWS Management Console under your AWS account credentials instead of as an AWS account user, from the user sign-in page, click **Sign in using AWS account credentials**. The Amazon Web Services sign-on 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 table 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
IAM is a feature of your AWS account. If you are already signed up for a product that is integrated with IAM, you don't need to do anything else to sign up for IAM, nor will you be charged extra for using it.

AWS feature	User access
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. 185). For information about creating passwords, see Managing Passwords (p. 63).
Your AWS resources, such as Amazon EC2, Amazon S3, 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 assign 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 that group permissions under a single policy. For information about creating users and groups, see Users and Groups (p. 42). For information about using policies to set permissions, see Permissions and Policies (p. 107).
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 User Access to Your AWS Account Billing Information
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.

Using an Alias for Your AWS Account ID

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.aws.amazon.com/console/ec2
```

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.aws.amazon.com/console/ec2
```

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

Creating, Deleting, and Listing an AWS Account Alias

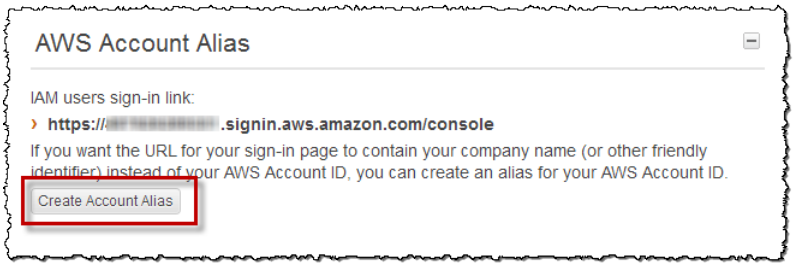
You can use the AWS Management Console, the IAM API, or the command line interface to create or delete your AWS account alias.

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 an account alias

1. On the **Navigation** pane, select **IAM Dashboard**.
2. Under **AWS Account Alias**, click **Create Account Alias**.



3. Enter the name you want to use for your alias, then click **Yes, Create**.

To remove an account alias

1. On the **Navigation** pane, select **IAM Dashboard**.
2. Under **AWS Account Alias**, click **Remove Account Alias**.

3. Review your change, and then click **Yes, Delete**.

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: `iam-accountaliascreate` API: `CreateAccountAlias`
Delete an AWS account ID alias	CLI: `iam-accountaliasdelete` API: `DeleteAccountAlias`
List your AWS account ID alias	CLI: `iam-accountaliaslist` API: `ListAccountAliases`

> **Note**
> The alias must be unique across all Amazon Web Services products, and the number of characters it can contain is limited. For more information on limitations on AWS account entities, see Limitations on IAM Entities (p. 16).

For more information on the IAM API actions or CLI commands, go to the AWS Identity and Access Management API Reference or AWS Identity and Access Management Command Line Interface Reference.

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. 87).

Managing Server Certificates

Topics

This section lists the IAM server certificate actions, describes what you need to know about renaming server certificates, and describes how to create and upload server certificates.

> **Note**
> If you want to create user signing certificate (not a server certificate) that you can use to sign SOAP requests in EC2, see Creating and Uploading a User Signing Certificate (p. 97)

Currently, Amazon Elastic Load Balancing is the only service to support the use of server certificates with IAM. For more information about using server certificates with Elastic Load Balancing, see the Elastic Load Balancing Developer Guide.

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	iam-servercertdel
Get server certificate information	GetServerCertificate	iam-servercertgetattributes
List server certificates	ListServerCertificates	iam-servercertlistbypath
Update server certificates	UpdateServerCertificate	iam-servercertmod
Upload server certificates	Upload ServerCertificate	iam-servercertupload

For more information about these actions, see the AWS Identity and Access Management API Reference or the AWS Identity and Access Management Command Line Interface Reference.

Renaming Server Certificates

When you rename a server certificate, the GUID for the server certificate remains the same (for more information about GUIDs, see *GUIDs* in IAM Identifiers (p. 11)). 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.

Creating, Uploading, and Deleting Server Certificates

This section describes the process of generating a digital server certificate and preparing it to use with AWS products through IAM. To create a signed certificate, you perform a series of tasks as described by the following topics.

> **Note**
> If you want to create user signing certificate (not a server certificate) that you can use to sign SOAP requests in EC2, see Creating and Uploading a User Signing Certificate (p. 97)

Topics

Install and Configure OpenSSL

Creating and uploading a certificate requires a tool that supports the SSL and TLS protocols. OpenSSL is an open-source tool that provides the basic cryptographic functions necessary to create an RSA token and sign it with your private key. If you don't already have OpenSSL installed, follow the instructions in this section.

To install OpenSSL on Linux and 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**.
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. The **OpenSSL Setup Wizard** launches.
5. Follow the instructions described in the **OpenSSL Setup Wizard**. Save the OpenSSL binaries to a folder in your working directory.

Before you use OpenSSL commands, you must configure the operating system so that it has information about the location of the OpenSSL install point.

To configure OpenSSL on Linux and 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 the OpenSSL installation:

```
export PATH=$PATH:$OpenSSL_HOME/bin
```

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 Path=OpenSSL_HOME\bin;%Path%
```

3. Set the path to the OpenSSL installation:

```
set Path=OpenSSL_HOME\bin;%Path%
```

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.

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 1024 > private-key.pem
```

For *`private-key.pem`*, specify your own file name. In the example, `1024` represents 1024-bit encryption. AWS also supports 2048-bit and 4096-bit encryption. We recommend you create an RSA key that is 1024-bit or 2048-bit in length.

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 digital 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.yourdomain.com
Email address	The server administrator's email address	someone@yourdomain.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.company.com" or "company.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 digital 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 any one CA. For a partial listing of available CAs, see Third-Party Certificate Authorities.

Upload the Signed Certificate

When you receive your digital server certificate from the certificate authority (CA), you can upload the certificate along with the private certificate and, optionally, a certificate chain to IAM. After you upload the certificates to IAM, the certificates are available for other AWS services to use.

To upload certificates on IAM, you use the IAM command line interface (IAM CLI). For more information about installing the IAM command line toolkit, refer to Getting the Command Line Tools in the *AWS Identity and Access Management Command Line Interface Reference*.

Note
A certificate authority might return a signed digital 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 a signed certificate

Your digital server certificate can include a chain. A chain contains a list of certificates that is used to build a trust path to a trusted Certificate Authority. If your signed certificate does not require a chain, omit the *-c* parameter.

- Use the `iam-servercertupload` command to upload a signed certificate:
  - On Linux and UNIX computers, enter the following command:

```
& ./iam-servercertupload -b public_key_certificate_file  -c certific
ate_chain_file -k privatekey.pem -s certificate_object_name
```

  - On Windows computers, enter the following command:

```
c:\ iam-servercertupload -b public_key_certificate_file  -c certific
ate_chain_file -k privatekey.pem -s certificate_object_name
```

You assign your own name to the certificate (the *certificate_object_name* parameter in the preceding command). For information about limitations on server certificate names, see Limitations on IAM Entities (p. 16).

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 digital server 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. 197)).
- The private key cannot be encrypted with a password.
- The SSL certificate chain must include all of your CA's intermediary certificates that lead to the root certificate. The chain starts with the public key SSL certificate that was generated by your CA 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. 197). 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 results 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. 197).

Verify the Certificate Object

After the digital 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 GUID. 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 GUID

- Use the `iam-servercertgetattributes` command to verify the certificate object:
  - On Linux and UNIX computers, enter the following command:

    ```
    & ./iam-servercertgetattributes -s certificate_object_name
    ```

  - On Windows computers, enter the following command:

    ```
    c:\ iam-servercertgetattributes -s certificate_object_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 a load balancer using Amazon ELB's HTTPS support, see the command line interface (CLI) examples in the How to Set Up a LoadBalancer 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 `iam-servercertdel` command to remove an individual certificate.
  - On Linux and UNIX computers, enter the following command:

    ```
    & ./iam-servercertdel -s certificate_object_name
    ```

  - On Windows computers, enter the following command:

    ```
    c:\ iam-servercertdel -s certificate_object_name
    ```

  If the command is successful, no output is displayed.

Sample Certificates

The following certificates show the valid format that IAM accepts for digital server certificates and their associated private key and certificate chain.

The digital server certificate associates your public key with your identity. When you submit your Certificate Signing Request (CSR) to a Certificate Authority (CA), a digital server certificate is returned to you by the CA. The following figure is a sample digital server certificate:

Sample digital server certificate

```
-----BEGIN CERTIFICATE-----
MIIE+TCCA+GgAwIBAgIQU306HIX4KsioTW1s2A2krTANBgkqhkiG9w0BAQUFADCB
tTELMAkGA1UEBhMCVVMxFzAVBgNVBAoTDlZlcmlTaWduLCBJbmMuMR8wHQYDVQQL
ExZWZXJpU2lnbiBUcnVzdCBOZXR3b3JrMTswOQYDVQQLEzJUZXJtcyBvZiB1c2Ug
YXQgaHR0cHM6Ly93d3cudmVyaXNpZ24uY29tL3JwYSAoYykwOTEvMC0GA1UEAxMm
VmVyaVNpZ24gQ2xhc3MgMyBTZWN1cmUgU2VydmVyIENBIC0gRzIwHhcNMTAxMDA4
MDAwMDAwWhcNMTMxMDA3MjM1OTU5WjBqMQswCQYDVQQGEwJVUzETMBEGA1UECBMK
V2FzaGluZ3RvbjEQMA4GA1UEBxQHU2VhdHRsZTEYMBYGA1UEChQPQW1hem9uLmNv
bSBJbmMuMRowGAYDVQQDFBFpYW0uYW1hem9uYXdzLmNvbTCBnzANBgkqhkiG9w0B
AQEFAAOBjQAwgYkCgYEA3Xb0EGea2dB8QGEUwLcEpwvGawEkUdLZmGL1rQJZdeeN
3vaF+ZTm8Qw5Adk2Gr/RwYXtpx04xvQXmNm+9YmksHmCZdruCrW1eN/P9wBfqMMZ
X964CjVov3NrF5AuxU8jgtw0yu//C3hWnOuIVGdg76626ggOoJSaj48R2n0MnVcC
AwEAAaOCAdEwggHNMAkGA1UdEwQCMAAwCwYDVR0PBAQDAgWgMEUGA1UdHwQ+MDww
OqA4oDaGNGh0dHA6Ly9TVlJTZWN1cmUtRzItY3JsLnZlcmlzaWduLmNvbS9TVlJT
ZWN1cmVHMi5jcmwwRAYDVR0gBD0wOzA5BgtghkgBhvhFAQcXAzAqMCgGCCsGAQUF
BwIBFhxodHRwczovL3d3dy52ZXJpc2lnbi5jb20vcnBhMB0GA1UdJQQWMBQGCCsG
AQUFBwMBBggrBgEFBQcDAjAfBgNVHSMEGDAWgBSl7wsRzsBBA6NKZZBIshzgVy19
RzB2BggrBgEFBQcBAQRqMGgwJAYIKwYBBQUHMAGGGGh0dHA6Ly9vY3NwLnZlcmlz
aWduLmNvbTBABggrBgEFBQcwAoY0aHR0cDovL1NWUlNlY3VyZS1HMi1haWEudmVy
aXNpZ24uY29tL1NWUlNlY3VyZUcyLmNlcjBuBggrBgEFBQcBDARiMGChXqBcMFow
WDBWFglpbWFnZS9naWYwITAfMAcGBSsOAwIaBBRLa7kolgYMu9BSOJsprEsHiyEF
GDAmFiRodHRwOi8vbG9nby52ZXJpc2lnbi5jb20vdnNsb2dvMS5naWYwDQYJKoZI
hvcNAQEFBQADggEBALpFBXeG782QsTtGwEE9zBcVCuKjrsl3dWK1dFiq3OP4y/Bi
ZBYEywBt8zNuYFUE25Ub/zmvmpe7p0G76tmQ8bRp/4qkJoiSesHJvFgJ1mksr3IQ
3gaE1aN2BSUIHxGLn9N4F09hYwwbeEZaCxfgBiLdEIodNwzcvGJ+2LlDWGJOGrNI
NM856xjqhJCPxYzk9buuCl1B4Kzu0CTbexz/iEgYV+DiuTxcfA4uhwMDSe0nynbn
1qiwRk450mCOnqH4ly4P4lXo02t4A/DI1I8ZNct/Qfl69a2Lf6vc9rF7BELT0e5Y
R7CKx7fc5xRaeQdyGj/dJevm9BF/mSdnclS5vas=
-----END CERTIFICATE-----
```

The private key allows you to decrypt messages that are encrypted with your public key. The following figure is a sample private key:

Sample private key

```
-----BEGIN RSA PRIVATE KEY-----
MIICXgIBAAKBgQDCvrqjUowfPitkkryxATvssBmq4p/voCWkU2ihjSqe3jw6g9xu
VrJ/B2SAtaY2KEfKCiW9hpOvmnhPBsZKfucP581Pad4CQyU1wX+rumydcpebvKti
UMg4RhoPiOmR8yqnaUJiVA1Lo4WoyPGaTfdEZc6xRLZCCF3RKM+gZXGiOwIDAQAB
AoGAfiZ41hVB6XcnOsYG7w4imEbWyah1/A6cc58INyYvxqulDi6emucUR08tnmaM
3aYoIsuB+Qx1HJqOdnqn9lfQKo8tJcU4VTwTI5acB/AWLUuSVbrp1fGQYPTqw/Ag
07Lfg9V4pWCG6CrQUDNToPQLBM6gPVo4FMDk/STXbujjVJkCQQDhE5EolI8B7GsG
J1sFzHiUscxb9EEKi5+BeFjxdfl78TKDILRX8doj3S4wtB+5/oUpQBDZmSH+mQU6
E1i3pVhfAkEA3YBXDtsSGxc0q/6QhDPPJ7KOvqBRj9CziMuzz/S3GEu3xJUqNLFf
v1ie8meapgnOrpmW9ErJNaC/c3mIgfNzpQJBAM2Q4XL+u94130mviCKzrS2hddRG
MWFARF4rXJCr/0CD+m5o4E2yRlmbGSTCXnexTk1uhfU3NyUg/PUd1llkWmECQQCg
HhlQvN4uxSynNGMlngoeyT3U4TF0g8p0lcRLDLyajImwSp/y7VGokZh85JXvduF4
Z8Cuoa0n3ibng7BBOEqdAkEAvuq6aCH+yYiCYhzWe6IDYCUnDa8PP/ExURzUAyYA
fGLgzft0+yAIkqfgvkqLBvVGtvYItJuzU4LiUiGriX6NNA==
-----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.

Sample certificate chain

```
-----BEGIN CERTIFICATE-----
CA public key certificate
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
Intermediate certificate 2
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
Intermediate certificate 1
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
Optional: Root certificate
-----END CERTIFICATE-----
```

AWS Services that Support IAM

This section links to topics that describe how IAM integrates with the different AWS products, and how to write policies to control access to a particular AWS product and its resources.

The following table summarizes whether you can grant IAM permissions that control access to a service's actions, resources, or both. For example, you can use IAM to control which Amazon EC2 actions users have access to, but you can't use IAM to control users' access to AMIs, volumes, instances, etc.

AWS Product	Actions	Resources	For more information, see...
Auto Scaling	✓		Auto Scaling and AWS Identity and Access Management in the *Auto Scaling Developer Guide*
Amazon Web Services Account Billing Information	✓*		Controlling User Access to Your AWS Account Billing Information *You can use IAM policies to control user access to your account's Account Activity page and Usage Reports page.
AWS CloudFormation	✓		Controlling User Access With AWS Identity and Access Management in the *AWS CloudFormation User Guide*
Amazon CloudFront	✓		Controlling User Access to Your AWS Account in the *AWS CloudFormation User Guide*
Amazon CloudWatch	✓		Controlling User Access to Your AWS Account in the *Amazon CloudWatch Developer Guide*
AWS Data Pipeline	✓		To get started with permissions, see Getting Started with IAM. To see what actions you can set permissions on, see *AWS Data Pipeline API reference*.

AWS Product	Actions	Resources	For more information, see...
Amazon DynamoDB	✓	✓	Controlling Access to Amazon DynamoDB Resources in the *Amazon DynamoDB Developer Guide*
AWS Elastic Beanstalk	✓	✓	Using AWS Elastic Beanstalk with AWS Identity and Access Management (IAM) in the *AWS Elastic Beanstalk Developer Guide*
Amazon Elastic Compute Cloud (Amazon EC2)	✓		AWS Identity and Access Management in the *Amazon Elastic Compute Cloud User Guide*
Elastic Load Balancing	✓		Controlling User Access to Your AWS Account in the *Elastic Load Balancing Developer Guide*
Amazon Elastic MapReduce (Amazon EMR)	✓		Configuring User Permissions in the *Amazon Elastic MapReduce Developer Guide*
Amazon Elastic Transcoder	✓		Security Considerations for Elastic Transcoder in the *Amazon Elastic Transcoder Developer Guide*
Amazon ElastiCache	✓		Controlling User Access to Your AWS Account in the *Amazon ElastiCache User Guide*
Amazon Glacier	✓	✓	Access Control Using AWS Identity and Access Management (IAM) in the *Amazon Glacier Developer Guide*
AWS Identity and Access Management (IAM)	✓	✓	Example IAM Policies (p. 116)
AWS Marketplace	✓		Controlling Access to AWS Marketplace Subscriptions
AWS OpsWorks	✓		Granting Users Permissions to Work with AWS OpsWorks in the *AWS OpsWorks User Guide*
Amazon Redshift	✓		Controlling Access to Amazon Redshift Resources in the *Amazon Redshift Management Guide*
Amazon Relational Database Service (Amazon RDS)	✓		Controlling User Access to Your AWS Account in the *Amazon Relational Database Service Developer Guide*

AWS Product	Actions	Resources	For more information, see...
Amazon Route 53	✓	✓	Controlling User Access with IAM in the *Amazon Route 53 Developer Guide*
Amazon Simple Email Service (Amazon SES)	✓		Controlling User Access with IAM in the *Amazon Simple Email Service Developer Guide*
Amazon Simple Notification Service (Amazon SNS)	✓	✓	Controlling User Access to Your AWS Account in the *Amazon Simple Notification Service Getting Started Guide*
Amazon Simple Queue Service (Amazon SQS)	✓	✓	Controlling User Access to Your AWS Account in the *Amazon Simple Queue Service Developer Guide*
Amazon Simple Storage Service (Amazon S3)	✓	✓	Using IAM Policies in the *Amazon Simple Storage Service Developer Guide*
Amazon Simple Workflow Service (Amazon SWF)	✓		Using IAM to Manage Access to Amazon SWF Resources in the *Amazon Simple Workflow Service Developer Guide*.
Amazon SimpleDB	✓	✓	Managing Users of Amazon SimpleDB in the *Amazon SimpleDB Developer Guide*
AWS Storage Gateway	✓	✓	Access Control Using AWS Identity and Access Management (IAM) in the *AWS Storage Gateway User Guide*
AWS Support	✓*		IAM for AWS Support on the AWS Support site *For AWS Support, IAM can only grant access to all Amazon Support functionality within an account.
Amazon Virtual Private Cloud (Amazon VPC)	✓		Controlling VPC Management in the *Amazon Virtual Private Cloud User Guide*

Troubleshooting

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 IAM Policies (p. 112).

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. Temporary security credentials are derived from an IAM user or role, so the permissions are limited to the IAM user or role. For information about viewing or modifying policies for an IAM user or role, see Managing IAM Policies (p. 112).
- 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 Policy Variables (p. 139).

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 Granting Applications that Run on Amazon EC2 Instances Access to AWS Resources (p. 152). For information about adding permissions to a user, see Managing IAM Policies (p. 112).

  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. 155).

  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::1234567890:instance-profile/MyInstanceProfile).

For more information about the permissions necessary to work with roles, see "How Do I Get Started?" in Granting Applications that Run on Amazon EC2 Instances Access to AWS Resources (p. 152). For information about adding permissions to a user, see Managing IAM Policies (p. 112).

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::1234567890:role/MyRole)
- 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 a Role (p. 170).

For more information about the permissions necessary to work with roles, see "How Do I Get Started?" in Granting Applications that Run on Amazon EC2 Instances Access to AWS Resources (p. 152). For information about adding permissions to a user, see Managing IAM Policies (p. 112).

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. 207) 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 assume role policy attached to the role. 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 details about the API actions and errors for the IAM API or the AWS Security Token Service API, go to the AWS Identity and Access Management API Reference or the AWS Security Token Service API Reference.

IAM and AWS Security Token Service support Query requests for calling service actions. Query requests are simple HTTPS requests, using the GET or POST method. Query requests must contain an `Action` parameter to indicate the action to be performed.

The response is an XML document that conforms to a schema. The schema is included in the WSDL. The IAM WSDL is located at https://iam.amazonaws.com/doc/2010-05-08/AWSIdentityManagement.wsdl. The AWS Security Token Service WSDL is located at: https://sts.amazonaws.com/doc/2011-06-15/AWSSecurityTokenService.wsdl.

Endpoints

IAM and AWS Security Token Service each have a single global endpoint. The IAM endpoint is https://iam.amazonaws.com. The AWS Security Token Service endpoint is https://sts.amazonaws.com.

For more information about AWS product endpoints and regions go to Regions and Endpoints in the *Amazon Web Services General Reference*.

HTTPS Required

Because the Query API returns sensitive information such as security credentials, you must use HTTPS with all API requests.

Signing AWS API Requests

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 also available in the AWS General Reference.

Document History

This Document History is associated with the 2010-05-08 release of AWS Identity and Access Management. This guide was last updated on 3 April 2013.

The following table describes important changes since the last release of *Using AWS Identity and Access Management*.

Change	Description	Release Date
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 Policy Variables (p. 139). 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 IAM Policies (p. 116).	This release
Best Practices	This release includes a topic on IAM best practices. For details, see IAM Best Practices (p. 34).	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 Enabling Cross-Account API Access (p. 155).	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. 89).	July 8, 2012
Business Use Cases	This section has been rewritten and updated. For more information, see Business Use Cases (p. 38)	June 22, 2012
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 Roles (p. 151).	June 07, 2012

Change	Description	Release Date
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. 21).	May 02, 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 Managing Passwords (p. 63).	March 08, 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 Managing an IAM Password Policy (p. 65).	March 08, 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 User Access to Your AWS Account Billing Information.	March 08, 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
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

Change	Description	Release Date
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. 38).	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. 78).	November 02, 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 language, see Overview of Policies (p. 109) in *Using AWS Identity and Access Management*. 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 06, 2011
Amazon ElastiCache	This release introduces Amazon ElastiCache integration with IAM. For more information about using IAM with 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. 38).	August 22, 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 03, 2011
Cross-Account Access for IAM Users	This release of IAM introduces cross-account access for IAM users. For more information, see Roles (p. 151).	June 06, 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. 10).	May 03, 2011
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

Change	Description	Release Date
AWS CloudFormation	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*.	24 February 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. 38) in *Using AWS Identity and Access Management*.	22 February 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. 185). For information about the AWS Management Console, see AWS Management Console.	14 February 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.	24 January 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. 200). For more information about Amazon Route 53, go to Amazon Route 53 on the AWS website.	05 December 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.	29 November 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. 191).	14 October 2010
Initial Release	This is the first release of *Using AWS Identity and Access Management*.	02 September 2010