
AWS Identity and Access Management

User Guide



AWS Identity and Access Management: User Guide

Copyright © 2021 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

What is IAM?	1
Video introduction to IAM	1
IAM features	1
Accessing IAM	2
How IAM works	3
Terms	4
Principal	5
Request	5
Authentication	5
Authorization	5
Actions or operations	6
Resources	6
Users in AWS	6
First-time access only: Your root user credentials	7
IAM users	7
Federating existing users	9
Permissions and policies in IAM	10
Policies and accounts	10
Policies and users	10
Policies and groups	11
Federated users and roles	12
Identity-based and resource-based policies	12
What is ABAC?	13
Comparing ABAC to the traditional RBAC model	13
Security features outside IAM	14
Quick links to common tasks	15
Getting set up	17
Using IAM to give users access to your AWS resources	17
Do I need to sign up for IAM?	18
Additional resources	18
Getting started	19
Creating an IAM admin user and group	20
Creating an administrator IAM user and group (console)	20
Creating an IAM user and group (AWS CLI)	21
Related resources	23
Creating a delegated user	23
Creating a delegated IAM user and group (console)	20
Reducing the group permissions	25
How IAM users sign in	26
Permissions required for console activities	27
Logging sign-in details in CloudTrail	27
IAM console search	27
Using IAM console search	27
Icons in the IAM console search results	28
Sample search phrases	28
Tutorials	30
Delegate access to the billing console	30
Prerequisites	31
Step 1: Activate access to billing data on your AWS test account	31
Step 2: Create IAM policies that grant permissions to billing data	31
Step 3: Attach billing policies to your groups	32
Step 4: Test access to the billing console	32
Related resources	33
Summary	33

Delegate access across AWS accounts using roles	34
Prerequisites	35
Step 1: Create a role	35
Step 2: Grant access to the role	37
Step 3: Test access by switching roles	39
Related resources	42
Summary	42
Create a customer managed policy	43
Prerequisites	43
Step 1: Create the policy	43
Step 2: Attach the policy	44
Step 3: Test user access	44
Related resources	45
Summary	45
Use attribute-based access control (ABAC)	45
Tutorial overview	45
Prerequisites	46
Step 1: Create test users	47
Step 2: Create the ABAC policy	48
Step 3: Create roles	50
Step 4: Test creating secrets	51
Step 5: Test viewing secrets	53
Step 6: Test scalability	54
Step 7: Test updating and deleting secrets	55
Summary	56
Related resources	57
Use SAML session tags for ABAC	57
Enable users to manage their credentials and MFA settings	60
Prerequisites	60
Step 1: Create a policy to enforce MFA sign-in	61
Step 2: Attach policies to your test group	61
Step 3: Test your user's access	62
Related resources	63
Signing in to AWS	64
Sign in as the root user	64
Sign in as an IAM user	65
Your AWS account ID and its alias	67
Finding your AWS account ID	67
About account aliases	68
Creating, deleting, and listing an AWS account alias	68
AWS sign-in issues	69
I need my AWS account ID or AWS account alias	70
I forgot my IAM user name or password	70
I forgot the root user password for my AWS account	70
I don't have access to the email for my AWS account	70
I need to change the credit card for my AWS account	70
I need to report fraudulent AWS account activity	70
I need to close my AWS account	71
Identities	72
AWS account root user	72
IAM users	72
IAM groups	72
IAM roles	73
Temporary credentials in IAM	73
When to create an IAM user (instead of a role)	73
When to create an IAM role (instead of a user)	73
Users	74

How AWS identifies an IAM user	74
Users and credentials	75
Users and permissions	75
Users and accounts	76
Users as service accounts	76
Adding a user	76
Controlling user access to the console	80
How IAM users sign in to AWS	81
Managing users	84
Changing permissions for a user	87
Managing passwords	92
Access keys	102
Retrieving lost passwords or access keys	110
Multi-factor authentication (MFA)	111
Finding unused credentials	146
Getting credential reports	148
Using IAM with CodeCommit	153
Using IAM with Amazon Keyspaces	154
Managing server certificates	155
Groups	160
Creating groups	161
Managing groups	162
Roles	167
Terms and concepts	168
Common scenarios	170
Identity providers and federation	176
Service-linked roles	213
Creating roles	221
Using roles	246
Managing roles	272
Roles vs. resource-based policies	286
Tagging users and roles	289
Choose an AWS tag naming convention	289
Rules for tagging in IAM and AWS STS	289
Permissions required for tagging IAM entities	290
Managing tags on IAM entities (console)	292
Managing tags on IAM entities (AWS CLI or AWS API)	292
Session tags	293
Temporary security credentials	301
AWS STS and AWS regions	302
Common scenarios for temporary credentials	302
Requesting temporary security credentials	303
Using temporary credentials with AWS resources	313
Controlling permissions for temporary security credentials	316
Managing AWS STS in an AWS Region	327
Using AWS STS interface VPC endpoints	330
Using bearer tokens	332
Sample applications that use temporary credentials	332
Additional resources for temporary credentials	333
AWS account root user	333
Create or delete an AWS account	334
Enable MFA on the AWS account root user	334
Creating access keys for the root user	335
Deleting access keys for the root user	335
Changing the password for the root user	336
Securing the credentials for the root user	336
Log events with CloudTrail	336

IAM and AWS STS information in CloudTrail	337
Logging IAM and AWS STS API requests	337
Logging API requests to other AWS services	337
Logging regional sign-in events	338
Logging user sign-in events	340
Logging sign-in events for temporary credentials	340
Example IAM API events in CloudTrail log	341
Example AWS STS API events in CloudTrail log	342
Example sign-in events in CloudTrail log	348
Access management	350
Access management resources	351
Policies and permissions	351
Policy types	351
Policies and the root user	356
Overview of JSON policies	356
Managed policies and inline policies	359
Permissions boundaries	365
Identity vs resource	374
Controlling access using policies	376
Controlling access using IAM tags	384
Controlling access using resource tags	387
Example policies	389
Managing IAM policies	438
Creating IAM policies	439
Validating policies	444
Testing IAM policies	445
Add or remove identity permissions	454
Versioning IAM policies	462
Editing IAM policies	465
Deleting IAM policies	469
Refining permissions using access information	472
Understanding policies	490
Policy summary (list of services)	491
Service summary (list of actions)	501
Action summary (list of resources)	506
Example policy summaries	509
Permissions required	516
Permissions for administering IAM identities	517
Permissions for working in the AWS Management Console	518
Granting permissions across AWS accounts	518
Permissions for one service to access another	519
Required actions	519
Example policies for IAM	520
Security	523
Data protection	523
Data encryption in IAM and AWS STS	524
Key management in IAM and AWS STS	524
Internetwork traffic privacy in IAM and AWS STS	524
Logging and monitoring	524
Compliance validation	525
Resilience	526
Infrastructure security	526
Configuration and vulnerability analysis	526
Security best practices and use cases	527
Security best practices	527
Business use cases	534
Access Analyzer	538

Supported resource types	539
S3 buckets	539
IAM roles	539
KMS keys	539
Lambda functions	540
SQS queues	541
How Access Analyzer works	541
Getting started	541
Permissions required to use Access Analyzer	541
Enabling Access Analyzer	543
Access Analyzer status	544
Access Analyzer quotas	545
Using service-linked roles	545
Settings	548
Delegated administrator for Access Analyzer	548
Access Analyzer findings	549
Working with findings	549
Reviewing findings	550
Filtering findings	551
Archiving findings	553
Resolving findings	554
Archive rules	554
Access Analyzer reference	555
Access Analyzer filter keys	556
Monitoring with EventBridge	558
Findings events	558
Event notification frequency	558
Example event	558
Creating an event rule using the console	559
Security Hub integration	561
How Access Analyzer sends findings to Security Hub	561
Viewing Access Analyzer findings in Security Hub	562
Typical finding from Access Analyzer	563
Enabling and configuring the integration	563
How to stop sending findings	564
Logging with CloudTrail	564
Access Analyzer information in CloudTrail	564
Understanding Access Analyzer log file entries	565
Troubleshooting IAM	566
General issues	566
I can't sign in to my AWS account	566
I lost my access keys	566
I get "access denied" when I make a request to an AWS service	567
I get "access denied" when I make a request with temporary security credentials	567
Policy variables aren't working	568
Changes that I make are not always immediately visible	568
I am not authorized to perform: iam>DeleteVirtualMFADevice	569
How do I securely create IAM users?	570
IAM policies	570
Troubleshoot using the visual editor	571
Troubleshoot using policy summaries	574
Troubleshoot policy management	580
Troubleshoot JSON policy documents	580
U2F security keys	584
I can't enable my U2F security key	584
I can't sign in using my U2F security key	585
I lost or broke my U2F key	585

Other issues	585
IAM roles	585
I can't assume a role	586
A new role appeared in my AWS account	587
I can't edit or delete a role in my AWS account	587
I'm not authorized to perform: iam:PassRole	587
Why can't I assume a role with a 12-hour session? (AWS CLI, AWS API)	588
I receive an error when I try to switch roles in the IAM console	588
My role has a policy that allows me to perform an action, but I get "access denied"	588
The service did not create the role's default policy version	589
There is no use case for a service role in the console	590
IAM and Amazon EC2	590
When attempting to launch an instance, I don't see the role I expected to see in the Amazon EC2 console IAM Role list	591
The credentials on my instance are for the wrong role	591
When I attempt to call the <code>AddRoleToInstanceProfile</code> , I get an <code>AccessDenied</code> error	591
Amazon EC2: When I attempt to launch an instance with a role, I get an <code>AccessDenied</code> error ...	592
I can't access the temporary security credentials on my EC2 instance	592
What do the errors from the <code>info</code> document in the IAM subtree mean?	593
IAM and Amazon S3	593
How do I grant anonymous access to an Amazon S3 bucket?	594
I'm signed in as an AWS account root user; why can't I access an Amazon S3 bucket under my account?	594
SAML 2.0 federation	594
Invalid SAML response	595
RoleSessionName is required	595
Not authorized for <code>AssumeRoleWithSAML</code>	595
Invalid RoleSessionName characters	596
Invalid response signature	596
Failed to assume role	596
Could not parse metadata	596
Could not parse metadata	597
DurationSeconds exceeds MaxSessionDuration	597
Viewing a SAML response in your browser	597
Reference	600
IAM identifiers	600
Friendly names and paths	600
IAM ARNs	601
Unique identifiers	604
Quotas	606
IAM name requirements	606
IAM object quotas	607
IAM and STS character quotas	608
Services that work with IAM	611
Compute	612
Containers	613
Storage	613
Database	614
Developer tools	614
Security, identity, & compliance	615
Cryptography & PKI	616
Machine learning	617
Management tools	618
Migration & transfer	620
Mobile	620
Networking & content delivery	620
Media	621

Analytics	622
Application integration	623
Business applications	624
Satellite	624
Internet of Things	624
Robotics	625
Quantum Computing	625
Blockchain	625
Game development	625
AR & VR	626
Customer enablement	626
Customer engagement	626
End user computing	627
Additional resources	627
Policy reference	627
JSON element reference	628
Policy evaluation logic	666
Policy grammar	679
AWS managed policies for job functions	684
Global condition keys	692
IAM condition keys	708
Actions, resources, and condition keys	719
Resources	720
Users and groups	720
Credentials (passwords, access keys, and MFA devices)	720
Permissions and policies	720
Federation and delegation	721
IAM and other AWS products	721
Using IAM with Amazon EC2	721
Using IAM with Amazon S3	721
Using IAM with Amazon RDS	722
Using IAM with Amazon DynamoDB	722
General security practices	722
General resources	722
Making HTTP query requests	724
Endpoints	724
HTTPS required	725
Signing IAM API requests	725
Document history	726

What is IAM?

 Follow us on Twitter

AWS Identity and Access Management (IAM) is a web service that helps you securely control access to AWS resources. You use IAM to control who is authenticated (signed in) and authorized (has permissions) to use resources.

When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the [best practice of using the root user only to create your first IAM user](#). Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.

Contents

- [Video introduction to IAM \(p. 1\)](#)
- [IAM features \(p. 1\)](#)
- [Accessing IAM \(p. 2\)](#)
- [Understanding how IAM works \(p. 3\)](#)
- [Overview of AWS identity management: Users \(p. 6\)](#)
- [Overview of access management: Permissions and policies \(p. 10\)](#)
- [What is ABAC for AWS? \(p. 13\)](#)
- [Security features outside IAM \(p. 14\)](#)
- [Quick links to common tasks \(p. 15\)](#)

Video introduction to IAM

AWS Training and Certification provides a 10-minute video introduction to IAM:

[Introduction to AWS Identity and Access Management](#)

IAM features

IAM gives you the following features:

Shared access to your AWS account

You can grant other people permission to administer and use resources in your AWS account without having to share your password or access key.

Granular permissions

You can grant different permissions to different people for different resources. For example, you might allow some users complete access to Amazon Elastic Compute Cloud (Amazon EC2), Amazon Simple Storage Service (Amazon S3), Amazon DynamoDB, Amazon Redshift, and other AWS services. For other users, you can allow read-only access to just some S3 buckets, or permission to administer just some EC2 instances, or to access your billing information but nothing else.

Secure access to AWS resources for applications that run on Amazon EC2

You can use IAM features to securely provide credentials for applications that run on EC2 instances. These credentials provide permissions for your application to access other AWS resources. Examples include S3 buckets and DynamoDB tables.

Multi-factor authentication (MFA)

You can add two-factor authentication to your account and to individual users for extra security. With MFA you or your users must provide not only a password or access key to work with your account, but also a code from a specially configured device.

Identity federation

You can allow users who already have passwords elsewhere—for example, in your corporate network or with an internet identity provider—to get temporary access to your AWS account.

Identity information for assurance

If you use [AWS CloudTrail](#), you receive log records that include information about those who made requests for resources in your account. That information is based on IAM identities.

PCI DSS Compliance

IAM supports the processing, storage, and transmission of credit card data by a merchant or service provider, and has been validated as being compliant with Payment Card Industry (PCI) Data Security Standard (DSS). For more information about PCI DSS, including how to request a copy of the AWS PCI Compliance Package, see [PCI DSS Level 1](#).

Integrated with many AWS services

For a list of AWS services that work with IAM, see [AWS services that work with IAM \(p. 611\)](#).

Eventually Consistent

IAM, like many other AWS services, is [eventually consistent](#). IAM achieves high availability by replicating data across multiple servers within Amazon's data centers around the world. If a request to change some data is successful, the change is committed and safely stored. However, the change must be replicated across IAM, which can take some time. Such changes include creating or updating users, groups, roles, or policies. We recommend that you do not include such IAM changes in the critical, high-availability code paths of your application. Instead, make IAM changes in a separate initialization or setup routine that you run less frequently. Also, be sure to verify that the changes have been propagated before production workflows depend on them. For more information, see [Changes that I make are not always immediately visible \(p. 568\)](#).

Free to use

AWS Identity and Access Management (IAM) and AWS Security Token Service (AWS STS) are features of your AWS account offered at no additional charge. You are charged only when you access other AWS services using your IAM users or AWS STS temporary security credentials. For information about the pricing of other AWS products, see the [Amazon Web Services pricing page](#).

Accessing IAM

You can work with AWS Identity and Access Management in any of the following ways.

AWS Management Console

The console is a browser-based interface to manage IAM and AWS resources. For more information about accessing IAM through the console, see [Signing in to the AWS Management Console as an IAM user or root user \(p. 64\)](#). For a tutorial that guides you through using the console, see [Creating your first IAM admin user and group \(p. 20\)](#).

AWS Command Line Tools

You can use the AWS command line tools to issue commands at your system's command line to perform IAM and AWS tasks. Using the command line can be faster and more convenient than the console. The command line tools are also useful if you want to build scripts that perform AWS tasks.

AWS provides two sets of command line tools: the [AWS Command Line Interface](#) (AWS CLI) and the [AWS Tools for Windows PowerShell](#). For information about installing and using the AWS CLI, see the [AWS Command Line Interface User Guide](#). For information about installing and using the Tools for Windows PowerShell, see the [AWS Tools for Windows PowerShell User Guide](#).

AWS SDKs

AWS provides SDKs (software development kits) that consist of libraries and sample code for various programming languages and platforms (Java, Python, Ruby, .NET, iOS, Android, etc.). The SDKs provide a convenient way to create programmatic access to IAM and AWS. For example, the SDKs take care of tasks such as cryptographically signing requests, managing errors, and retrying requests automatically. For information about the AWS SDKs, including how to download and install them, see the [Tools for Amazon Web Services](#) page.

IAM HTTPS API

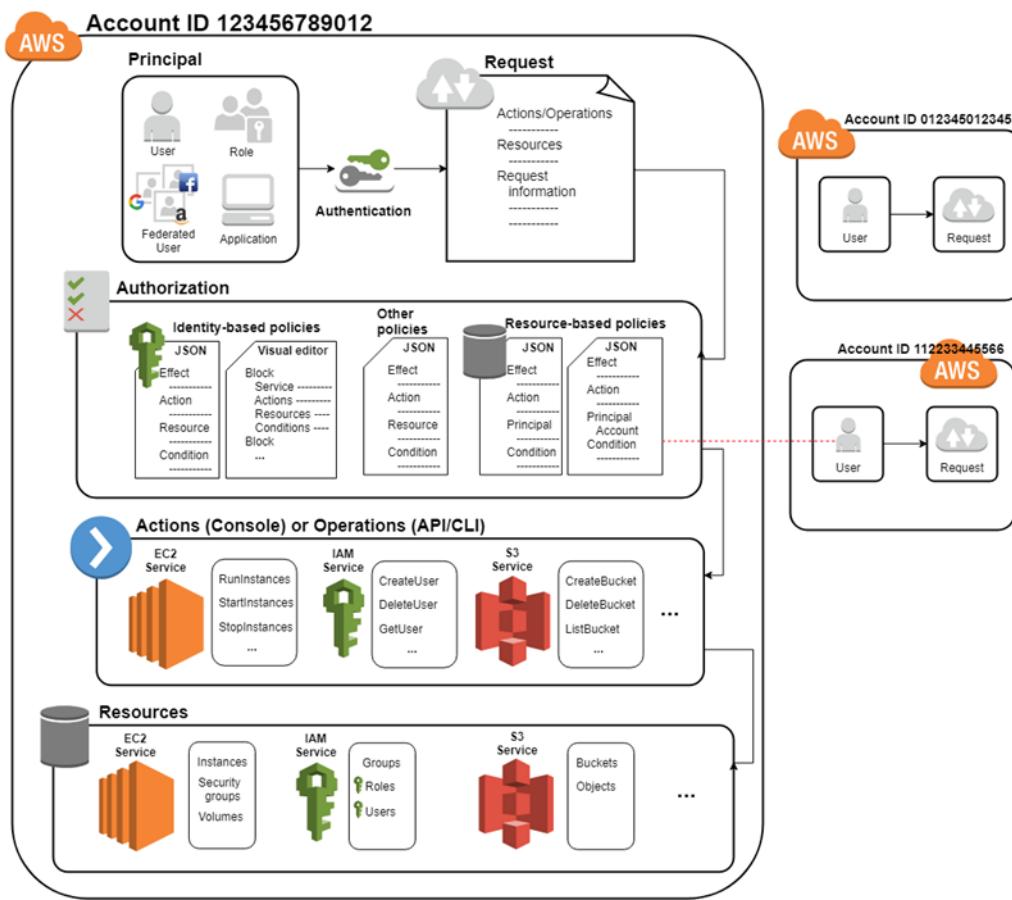
You can access IAM and AWS programmatically by using the IAM HTTPS API, which lets you issue HTTPS requests directly to the service. When you use the HTTPS API, you must include code to digitally sign requests using your credentials. For more information, see [Calling the IAM API using HTTP query requests \(p. 724\)](#) and the [IAM API Reference](#).

Understanding how IAM works

Before you create users, you should understand how IAM works. IAM provides the infrastructure necessary to control authentication and authorization for your account. The IAM infrastructure includes the following elements:

Contents

- [Terms \(p. 4\)](#)
- [Principal \(p. 5\)](#)
- [Request \(p. 5\)](#)
- [Authentication \(p. 5\)](#)
- [Authorization \(p. 5\)](#)
- [Actions or operations \(p. 6\)](#)
- [Resources \(p. 6\)](#)



Terms

Learn more about IAM terms.

Resources

The user, group, role, policy, and identity provider objects that are stored in IAM. As with other AWS services, you can add, edit, and remove resources from IAM.

Identities

The IAM resource objects that are used to identify and group. You can attach a policy to an IAM identity. These include users, groups, and roles.

Entities

The IAM resource objects that AWS uses for authentication. These include IAM users, federated users, and assumed IAM roles.

Principals

A person or application that uses the AWS account root user, an IAM user, or an IAM role to sign in and make requests to AWS.

Principal

A *principal* is a person or application that can make a request for an action or operation on an AWS resource. The principal is authenticated as the AWS account root user or an IAM entity to make requests to AWS. As a best practice, do not use your root user credentials for your daily work. Instead, create IAM entities (users and roles). You can also support federated users or programmatic access to allow an application to access your AWS account.

Request

When a principal tries to use the AWS Management Console, the AWS API, or the AWS CLI, that principal sends a *request* to AWS. The request includes the following information:

- **Actions or operations** – The actions or operations that the principal wants to perform. This can be an action in the AWS Management Console, or an operation in the AWS CLI or AWS API.
- **Resources** – The AWS resource object upon which the actions or operations are performed.
- **Principal** – The person or application that used an entity (user or role) to send the request. Information about the principal includes the policies that are associated with the entity that the principal used to sign in.
- **Environment data** – Information about the IP address, user agent, SSL enabled status, or the time of day.
- **Resource data** – Data related to the resource that is being requested. This can include information such as a DynamoDB table name or a tag on an Amazon EC2 instance.

AWS gathers the request information into a *request context*, which is used to evaluate and authorize the request.

Authentication

A principal must be authenticated (signed in to AWS) using their credentials to send a request to AWS. Some services, such as Amazon S3 and AWS STS, allow a few requests from anonymous users. However, they are the exception to the rule.

To authenticate from the console as a root user, you must sign in with your email address and password. As an IAM user, provide your account ID or alias, and then your user name and password. To authenticate from the API or AWS CLI, you must provide your access key and secret key. You might also be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more about the IAM entities that AWS can authenticate, see [IAM users \(p. 74\)](#) and [IAM roles \(p. 167\)](#).

Authorization

You must also be authorized (allowed) to complete your request. During authorization, AWS uses values from the request context to check for policies that apply to the request. It then uses the policies to determine whether to allow or deny the request. Most policies are stored in AWS as [JSON documents \(p. 356\)](#) and specify the permissions for principal entities. There are [several types of policies \(p. 351\)](#) that can affect whether a request is authorized. To provide your users with permissions to access the AWS resources in their own account, you need only identity-based policies. Resource-based policies are popular for granting [cross-account access \(p. 518\)](#). The other policy types are advanced features and should be used carefully.

AWS checks each policy that applies to the context of your request. If a single permissions policy includes a denied action, AWS denies the entire request and stops evaluating. This is called an *explicit deny*.

Because requests are *denied by default*, AWS authorizes your request only if every part of your request is allowed by the applicable permissions policies. The evaluation logic for a request within a single account follows these general rules:

- By default, all requests are denied. (In general, requests made using the AWS account root user credentials for resources in the account are always allowed.)
- An explicit allow in any permissions policy (identity-based or resource-based) overrides this default.
- The existence of an Organizations SCP, IAM permissions boundary, or a session policy overrides the allow. If one or more of these policy types exists, they must all allow the request. Otherwise, it is implicitly denied.
- An explicit deny in any policy overrides any allows.

To learn more about how all types of policies are evaluated, see [Policy evaluation logic \(p. 666\)](#). If you need to make a request in a different account, a policy in the other account must allow you to access the resource *and* the IAM entity that you use to make the request must have an identity-based policy that allows the request.

Actions or operations

After your request has been authenticated and authorized, AWS approves the actions or operations in your request. Operations are defined by a service, and include things that you can do to a resource, such as viewing, creating, editing, and deleting that resource. For example, IAM supports approximately 40 actions for a user resource, including the following actions:

- `CreateUser`
- `DeleteUser`
- `GetUser`
- `UpdateUser`

To allow a principal to perform an operation, you must include the necessary actions in a policy that applies to the principal or the affected resource. To see a list of actions, resource types, and condition keys supported by each service, see [Actions, Resources, and Condition Keys for AWS Services](#).

Resources

After AWS approves the operations in your request, they can be performed on the related resources within your account. A resource is an object that exists within a service. Examples include an Amazon EC2 instance, an IAM user, and an Amazon S3 bucket. The service defines a set of actions that can be performed on each resource. If you create a request to perform an unrelated action on a resource, that request is denied. For example, if you request to delete an IAM role but provide an IAM group resource, the request fails. To see AWS service tables that identify which resources are affected by an action, see [Actions, Resources, and Condition Keys for AWS Services](#).

Overview of AWS identity management: Users

For greater security and organization, you can give access to your AWS account to specific users—identities that you create with custom permissions. You can further simplify access for those users by federating existing identities into AWS.

Topics

- [First-time access only: Your root user credentials \(p. 7\)](#)
- [IAM users \(p. 7\)](#)
- [Federating existing users \(p. 9\)](#)

First-time access only: Your root user credentials

When you create an AWS account, you create an AWS account root user identity, which you use to sign in to AWS. You can sign in to the AWS Management Console using this root user identity—that is, the email address and password that you provided when creating the account. This combination of your email address and password is also called your *root user credentials*.

When you use your root user credentials, you have complete, unrestricted access to all resources in your AWS account, including access to your billing information and the ability to change your password. This level of access is necessary when you first set up your account. However, we recommend that you **don't** use root user credentials for everyday access. We especially recommend that you do not share your root user credentials with anyone, because doing so gives them unrestricted access to your account. Only service control policies (SCPs) in organizations can restrict the permissions that are granted to the root user.

The following sections explain how you can use IAM to create and manage user identity and permissions to provide secure, limited access to your AWS resources, both for yourself and for others who need to work with your AWS resources.

IAM users

The "identity" aspect of AWS Identity and Access Management (IAM) helps you with the question "Who is that user?", often referred to as *authentication*. Instead of sharing your root user credentials with others, you can create individual IAM users within your account that correspond to users in your organization. IAM users are not separate accounts; they are users within your account. Each user can have its own password for access to the AWS Management Console. You can also create an individual access key for each user so that the user can make programmatic requests to work with resources in your account. In the following figure, the users Li, Mateo, DevApp1, DevApp2, TestApp1, and TestApp2 have been added to a single AWS account. Each user has its own credentials.



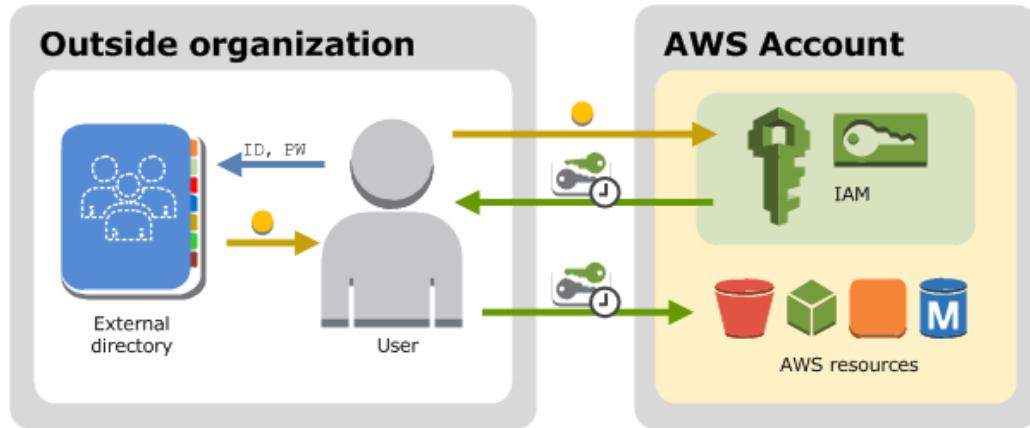
Notice that some of the users are actually applications (for example, DevApp1). An IAM user doesn't have to represent an actual person; you can create an IAM user in order to generate an access key for an application that runs in your corporate network and needs AWS access.

We recommend that you create an IAM user for yourself and then assign yourself administrative permissions for your account. You can then sign in as that user to add more users as needed.

Federating existing users

If the users in your organization already have a way to be authenticated, such as by signing in to your corporate network, you don't have to create separate IAM users for them. Instead, you can *federate* those user identities into AWS.

The following diagram shows how a user can use IAM to get temporary AWS security credentials to access resources in your AWS account.



Federation is particularly useful in these cases:

- **Your users already have identities in a corporate directory.**

If your corporate directory is compatible with Security Assertion Markup Language 2.0 (SAML 2.0), you can configure your corporate directory to provide single-sign on (SSO) access to the AWS Management Console for your users. For more information, see [Common scenarios for temporary credentials \(p. 302\)](#).

If your corporate directory is not compatible with SAML 2.0, you can create an identity broker application to provide single-sign on (SSO) access to the AWS Management Console for your users. For more information, see [Enabling custom identity broker access to the AWS console \(p. 206\)](#).

If your corporate directory is Microsoft Active Directory, you can use [AWS Directory Service](#) to establish trust between your corporate directory and your AWS account.

- **Your users already have Internet identities.**

If you are creating a mobile app or web-based app that can let users identify themselves through an Internet identity provider like Login with Amazon, Facebook, Google, or any OpenID Connect (OIDC) compatible identity provider, the app can use federation to access AWS. For more information, see [About web identity federation \(p. 177\)](#).

Tip

To use identity federation with Internet identity providers, we recommend you use [Amazon Cognito](#).

Overview of access management: Permissions and policies

The access management portion of AWS Identity and Access Management (IAM) helps you define what a principal entity is allowed to do in an account. A principal entity is a person or application that is authenticated using an IAM entity (user or role). Access management is often referred to as *authorization*. You manage access in AWS by creating policies and attaching them to IAM identities (users, groups of users, or roles) or AWS resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal uses an IAM entity (user or role) to make a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about policy types and uses, see [Policies and permissions in IAM \(p. 351\)](#).

Policies and accounts

If you manage a single account in AWS, then you define the permissions within that account using policies. If you manage permissions across multiple accounts, it is more difficult to manage permissions for your users. You can use IAM roles, resource-based policies, or access control lists (ACLs) for cross-account permissions. However, if you own multiple accounts, we instead recommend using the AWS Organizations service to help you manage those permissions. For more information, see [What is AWS Organizations?](#) in the *Organizations User Guide*.

Policies and users

IAM users are identities in the service. When you create an IAM user, they can't access anything in your account until you give them permission. You give permissions to a user by creating an identity-based policy, which is a policy that is attached to the user or a group to which the user belongs. The following example shows a JSON policy that allows the user to perform all Amazon DynamoDB actions (`dynamodb : *`) on the `Books` table in the `123456789012` account within the `us-east-2` Region.

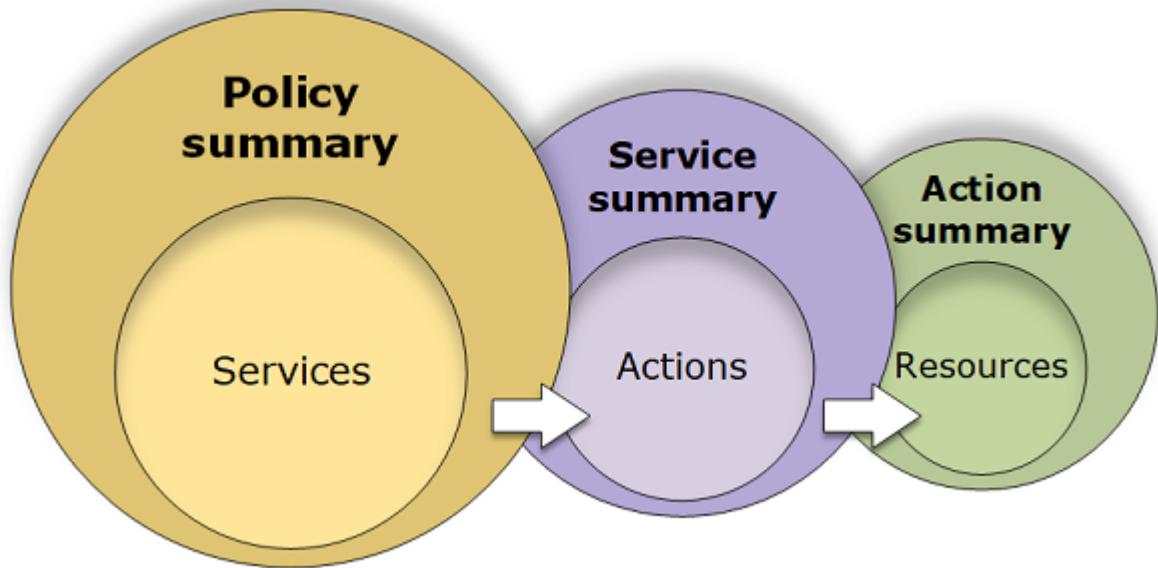
```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "dynamodb:*",  
            "Resource": "arn:aws:dynamodb:us-east-2:123456789012:table/Books"  
        }  
    ]  
}
```

After you attach this policy to your IAM user, the user only has those DynamoDB permissions. Most users have multiple policies that together represent the permissions for that user.

Actions or resources that are not explicitly allowed are denied by default. For example, if the preceding policy is the only policy that is attached to a user, then that user is allowed to only perform DynamoDB actions on the `Books` table. Actions on all other tables are prohibited. Similarly, the user is not allowed to perform any actions in Amazon EC2, Amazon S3, or in any other AWS service. The reason is that permissions to work with those services are not included in the policy.

The IAM console includes *policy summary* tables that describe the access level, resources, and conditions that are allowed or denied for each service in a policy. Policies are summarized in three tables: the [policy summary \(p. 491\)](#), the [service summary \(p. 501\)](#), and the [action summary \(p. 506\)](#). The *policy summary* table includes a list of services. Choose a service there to see the *service summary*. This summary table includes a list of the actions and associated permissions for the chosen service. You can

choose an action from that table to view the *action summary*. This table includes a list of resources and conditions for the chosen action.



You can view policy summaries on the **Users** page for all policies (managed and inline) that are attached to that user. View summaries on the **Policies** page for all managed policies.

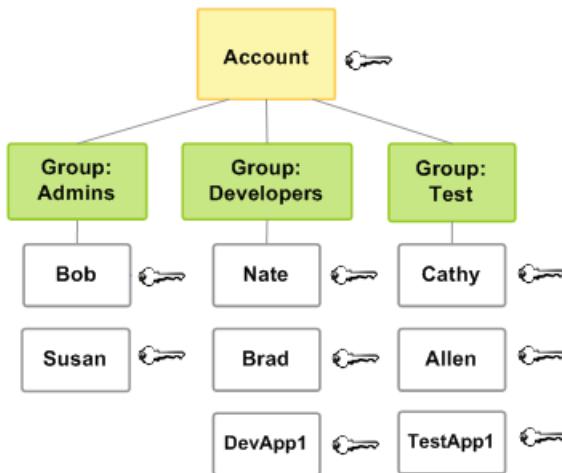
For example, the previous policy is summarized in the AWS Management Console as follows:

Service	Access level	Resource	Request condition
Allow (1 of 102 services) Show remaining 101			
DynamoDB	Full access	TableName = Books	None

You can also view the JSON document for the policy. For information about viewing the summary or JSON document, see [Understanding permissions granted by a policy \(p. 490\)](#).

Policies and groups

You can organize IAM users into *IAM groups* and attach a policy to a group. In that case, individual users still have their own credentials, but all the users in a group have the permissions that are attached to the group. Use groups for easier permissions management, and to follow our [Security best practices in IAM \(p. 527\)](#).



Users or groups can have multiple policies attached to them that grant different permissions. In that case, the users' permissions are calculated based on the combination of policies. But the basic principle still applies: If the user has not been granted an explicit permission for an action and a resource, the user does not have those permissions.

Federated users and roles

Federated users don't have permanent identities in your AWS account the way that IAM users do. To assign permissions to federated users, you can create an entity referred to as a *role* and define permissions for the role. When a federated user signs in to AWS, the user is associated with the role and is granted the permissions that are defined in the role. For more information, see [Creating a role for a third-party Identity Provider \(federation\) \(p. 234\)](#).

Identity-based and resource-based policies

Identity-based policies are permissions policies that you attach to an IAM identity, such as an IAM user, group, or role. Resource-based policies are permissions policies that you attach to a resource such as an Amazon S3 bucket or an IAM role trust policy.

Identity-based policies control what actions the identity can perform, on which resources, and under what conditions. Identity-based policies can be further categorized:

- **Managed policies** – Standalone identity-based policies that you can attach to multiple users, groups, and roles in your AWS account. You can use two types of managed policies:
 - **AWS managed policies** – Managed policies that are created and managed by AWS. If you are new to using policies, we recommend that you start by using AWS managed policies.
 - **Customer managed policies** – Managed policies that you create and manage in your AWS account. Customer managed policies provide more precise control over your policies than AWS managed policies. You can create and edit an IAM policy in the visual editor or by creating the JSON policy document directly. For more information, see [Creating IAM policies \(p. 439\)](#) and [Editing IAM policies \(p. 465\)](#).
- **Inline policies** – Policies that you create and manage and that are embedded directly into a single user, group, or role. In most cases, we don't recommend using inline policies.

Resource-based policies control what actions a specified principal can perform on that resource and under what conditions. Resource-based policies are inline policies, and there are no managed resource-

based policies. To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy.

The IAM service supports only one type of resource-based policy called a role *trust policy*, which is attached to an IAM role. Because an IAM role is both an identity and a resource that supports resource-based policies, you must attach both a trust policy and an identity-based policy to an IAM role. Trust policies define which principal entities (accounts, users, roles, and federated users) can assume the role. To learn how IAM roles are different from other resource-based policies, see [How IAM roles differ from resource-based policies \(p. 286\)](#).

To see which services support resource-based policies, see [AWS services that work with IAM \(p. 611\)](#). To learn more about resource-based policies, see [Identity-based policies and resource-based policies \(p. 374\)](#).

What is ABAC for AWS?

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called *tags*. Tags can be attached to IAM principals (users or roles) and to AWS resources. You can create a single ABAC policy or small set of policies for your IAM principals. These ABAC policies can be designed to allow operations when the principal's tag matches the resource tag. ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

For example, you can create three roles with the `access-project` tag key. Set the tag value of the first role to `Heart`, the second to `Sun`, and the third to `Lightning`. You can then use a single policy that allows access when the role and the resource are tagged with the same value for `access-project`. For a detailed tutorial that demonstrates how to use ABAC in AWS, see [IAM Tutorial: Define permissions to access AWS resources based on tags \(p. 45\)](#).

Comparing ABAC to the traditional RBAC model

The traditional authorization model used in IAM is called role-based access control (RBAC). RBAC defines permissions based on a person's job function, known outside of AWS as a *role*. Within AWS a role usually refers to an IAM role, which is an identity in IAM that you can assume. IAM does include [managed policies for job functions \(p. 684\)](#) that align permissions to a job function in an RBAC model.

In IAM, you implement RBAC by creating different policies for different job functions. You then attach the policies to identities (IAM users, groups of users, or IAM roles). As a best practice, you grant the minimum permissions necessary for the job function. This is known as [granting least privilege \(p. 529\)](#). Do this by listing the specific resources that the job function can access. The disadvantage to using the traditional RBAC model is that when employees add new resources, you must update policies to allow access to those resources.

For example, assume that you have three projects, named `Heart`, `Sun`, and `Lightning`, on which your employees work. You create an IAM role for each project. You then attach policies to each IAM role to define the resources that anyone allowed to assume the role can access. If an employee changes jobs within your company, you assign them to a different IAM role. People or programs can be assigned to more than one role. However, the `Sun` project might require additional resources, such as a new Amazon S3 bucket. In that case, you must update the policy attached to the `Sun` role to specify the new bucket resource. Otherwise, `Sun` project members are not allowed to access the new bucket.

ABAC provides the following advantages over the traditional RBAC model:

- **ABAC permissions scale with innovation.** It's no longer necessary for an administrator to update existing policies to allow access to new resources. For example, assume that you designed your ABAC

strategy with the `access-project` tag. A developer uses the role with the `access-project = Heart` tag. When people on the `Heart` project need additional Amazon EC2 resources, the developer can create new Amazon EC2 instances with the `access-project = Heart` tag. Then anyone on the `Heart` project can start and stop those instances because their tag values match.

- **ABAC requires fewer policies.** Because you don't have to create different policies for different job functions, you create fewer policies. Those policies are easier to manage.
- **Using ABAC, teams can change and grow quickly.** This is because permissions for new resources are automatically granted based on attributes. For example, if your company already supports the `Heart` and `Sun` projects using ABAC, it's easy to add a new `Lightning` project. An IAM administrator creates a new role with the `access-project = Lightning` tag. It's not necessary to change the policy to support a new project. Anyone that has permissions to assume the role can create and view instances tagged with `access-project = Lightning`. Additionally, a team member might move from the `Heart` project to the `Lightning` project. The IAM administrator assigns the user to a different IAM role. It's not necessary to change the permissions policies.
- **Granular permissions are possible using ABAC.** When you create policies, it's a best practice to [grant least privilege \(p. 529\)](#). Using traditional RBAC, you must write a policy that allows access to only specific resources. However, when you use ABAC, you can allow actions on all resources, but only if the resource tag matches the principal's tag.
- **Use employee attributes from your corporate directory with ABAC.** You can configure your SAML-based or web identity provider to pass session tags to AWS. When your employees federate into AWS, their attributes are applied to their resulting principal in AWS. You can then use ABAC to allow or deny permissions based on those attributes.

For a detailed tutorial that demonstrates how to use ABAC in AWS, see [IAM Tutorial: Define permissions to access AWS resources based on tags \(p. 45\)](#).

Security features outside IAM

You use IAM to control access to tasks that are performed using the AWS Management Console, the [AWS Command Line Tools](#), or service API operations using the [AWS SDKs](#). Some AWS products have other ways to secure their resources as well. The following list provides some examples, though it is not exhaustive.

Amazon EC2

In Amazon Elastic Compute Cloud you log into an instance with a key pair (for Linux instances) or using a user name and password (for Microsoft Windows instances).

For more information, see the following documentation:

- [Getting Started with Amazon EC2 Linux Instances](#) in the *Amazon EC2 User Guide for Linux Instances*
- [Getting Started with Amazon EC2 Windows Instances](#) in the *Amazon EC2 User Guide for Windows Instances*

Amazon RDS

In Amazon Relational Database Service you log into the database engine with a user name and password that are tied to that database.

For more information, see [Getting Started with Amazon RDS](#) in the *Amazon RDS User Guide*.

Amazon EC2 and Amazon RDS

In Amazon EC2 and Amazon RDS you use security groups to control traffic to an instance or database.

For more information, see the following documentation:

- [Amazon EC2 Security Groups for Linux Instances](#) in the *Amazon EC2 User Guide for Linux Instances*
- [Amazon EC2 Security Groups for Windows Instances](#) in the *Amazon EC2 User Guide for Windows Instances*
- [Amazon RDS Security Groups](#) in the *Amazon RDS User Guide*

Amazon WorkSpaces

In Amazon WorkSpaces, users sign in to a desktop with a user name and password.

For more information, see [Getting Started with Amazon WorkSpaces](#) in the *Amazon WorkSpaces Administration Guide*.

Amazon WorkDocs

In Amazon WorkDocs, users get access to shared documents by signing in with a user name and password.

For more information, see [Getting Started with Amazon WorkDocs](#) in the *Amazon WorkDocs Administration Guide*.

These access control methods are not part of IAM. IAM lets you control how these AWS products are administered—creating or terminating an Amazon EC2 instance, setting up new Amazon WorkSpaces desktops, and so on. That is, IAM helps you control the tasks that are performed by making requests to Amazon Web Services, and it helps you control access to the AWS Management Console. However, IAM does not help you manage security for tasks like signing in to an operating system (Amazon EC2), database (Amazon RDS), desktop (Amazon WorkSpaces), or collaboration site (Amazon WorkDocs).

When you work with a specific AWS product, be sure to read the documentation to learn the security options for all the resources that belong to that product.

Quick links to common tasks

Use the following links to get help with common tasks associated with IAM.

Sign in as an IAM user

See [How IAM users sign in to AWS \(p. 81\)](#).

Manage passwords for IAM users

You need a password in order to access the AWS Management Console, including access to billing information.

For your AWS account root user, see [Changing the AWS account root user password \(p. 92\)](#).

For an IAM user, see [Managing passwords for IAM users \(p. 96\)](#).

Manage permissions for IAM users

You use policies to grant permissions to the IAM users in your AWS account. IAM users have no permissions when they are created, so you must add permissions to allow them to use AWS resources.

For more information, see [Managing IAM policies \(p. 438\)](#).

List the users in your AWS account and get information about their credentials

See [Getting credential reports for your AWS account \(p. 148\)](#).

Add multi-factor authentication (MFA)

To add a virtual MFA device, see one of the following:

- [Enable a virtual MFA device for your AWS account root user \(console\) \(p. 115\)](#)
- [Enable a virtual MFA device for an IAM user \(console\) \(p. 114\)](#)

To add a U2F security key, see one of the following:

- [Enable a U2F security key for the AWS account root user \(console\) \(p. 120\)](#)
- [Enable a U2F security key for another IAM user \(console\) \(p. 119\)](#)

To add a hardware MFA device, see one of the following:

- [Enable a hardware MFA device for the AWS account root user \(console\) \(p. 125\)](#)
- [Enable a hardware MFA device for another IAM user \(console\) \(p. 124\)](#)

Get an access key

You need an access key if you want to make AWS requests using the AWS SDKs, the [AWS Command Line Tools](#), or the API operations.

Important

You can view and download your secret access key *only* when you create the access key. You cannot view or recover a secret access key later. However, if you lose your secret access key, you can create a new access key.

For your AWS account, see [Managing Access Keys for your AWS Account](#).

For an IAM user, see [Managing access keys for IAM users \(p. 102\)](#).

Tag a user or role

You can tag an IAM user or role using the IAM console, the AWS CLI, or the API through one of the AWS SDKs.

To learn about tags in IAM, see [Tagging IAM users and roles \(p. 289\)](#).

For details about how to manage tags in IAM, see [Managing tags on IAM entities \(console\) \(p. 292\)](#).

To learn about using IAM tags to control access to AWS, see [Controlling access to and for IAM users and roles using IAM resource tags \(p. 384\)](#).

View the actions, resources, and condition keys for all services

This set of reference documentation can help you write detailed IAM policies. Each AWS service defines the actions, resources, and condition context keys that you use in IAM policies. To learn more, see [Actions, Resources, and Condition Keys for AWS Services](#).

Get started with all of AWS

This set of documentation deals primarily with the IAM service. To learn about getting started with AWS and using multiple services to solve a problem such as building and launching your first project, see the [Getting Started Resource Center](#).

Getting set up with IAM

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

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

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

Topics

- [Using IAM to give users access to your AWS resources \(p. 17\)](#)
- [Do I need to sign up for IAM? \(p. 18\)](#)
- [Additional resources \(p. 18\)](#)

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 in 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 in your AWS account, see Getting started with IAM (p. 19) . To learn about using the IAM API or AWS Command Line Interface to create users in your AWS account, see Creating your first IAM admin user and group (p. 20) . For more information about working with IAM users, see IAM Identities (users, groups, and roles) (p. 72) .
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 Temporary security credentials in IAM (p. 301) . For information about the AWS Security Token Service API, go to the AWS Security Token Service API Reference .

Type of access	Why would I use it?	Where can I get more information?
Cross-account access between AWS accounts	You want to share access to certain AWS resources with users under other AWS accounts.	To learn how to use IAM to grant permissions to other AWS accounts, see Roles terms and concepts (p. 168) .

Do I need to sign up for IAM?

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

Note

IAM works only with AWS products that are integrated with IAM. For a list of services that support IAM, see [AWS services that work with IAM \(p. 611\)](#).

To sign up for AWS

1. Open <https://portal.aws.amazon.com/billing/signup>.
2. Follow the online instructions.

Part of the sign-up procedure involves receiving a phone call and entering a verification code on the phone keypad.

Additional resources

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

- Manage your AWS account credentials: [AWS Security Credentials](#) in the [AWS General Reference](#)
- Get started with and learn more about [What is IAM? \(p. 1\)](#)
- Set up a command line interface (CLI) to use with IAM. For the cross-platform AWS CLI, see the [AWS Command Line Interface Documentation](#) and [IAM CLI reference](#). You can also manage IAM with Windows PowerShell; see the [AWS Tools for Windows PowerShell Documentation](#) and [IAM Windows PowerShell reference](#).
- Download an AWS SDK for convenient programmatic access to IAM: [Tools for Amazon Web Services](#)
- Get the FAQ: [AWS Identity and Access Management FAQ](#)
- Get technical support: [AWS Support Center](#)
- Get premium technical support: [AWS Premium Support Center](#)
- Find definitions of AWS terms: [Amazon Web Services Glossary](#)
- Get community support: [IAM Discussion Forums](#)
- Contact AWS: [Contact Us](#)

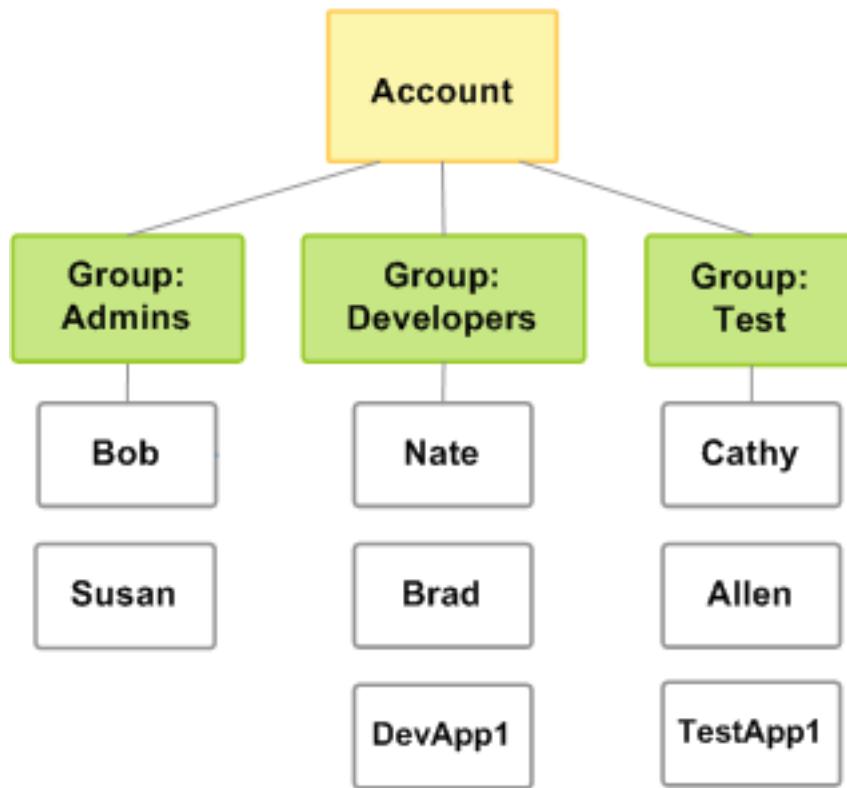
Getting started with IAM

This topic shows you how to give access to your AWS resources by creating AWS Identity and Access Management (IAM) users in your AWS account. First, you'll learn about IAM 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.

Note

This set of documentation deals primarily with the IAM service. To learn about getting started with AWS and using multiple services to solve a problem such as building and launching your first project, see the [Getting Started Resource Center](#).

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



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

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

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

Topics

- [Creating your first IAM admin user and group \(p. 20\)](#)
- [Creating your first IAM delegated user and group \(p. 23\)](#)
- [How IAM users sign in to your AWS account \(p. 26\)](#)
- [IAM console search \(p. 27\)](#)

Creating your first IAM admin user and group

Important

If you found this page because you are looking for information about the Product Advertising API to sell Amazon products on your website, see the [Product Advertising API 5.0 Documentation](#).

As a [best practice \(p. 528\)](#), do not use the AWS account root user for any task where it's not required. Instead, create a new IAM user for each person that requires administrator access. Then make those users administrators by placing the users into an "Administrators" group to which you attach the AdministratorAccess managed policy.

Thereafter, the users in the administrators group should set up the groups, users, and so on, for the AWS account. All future interaction should be through the AWS account's users and their own keys instead of the root user. However, to perform some account and service management tasks, you must log in using the root user credentials. To view the tasks that require you to sign in as the root user, see [AWS Tasks that Require Account Root User](#).

Creating an administrator IAM user and group (console)

This procedure describes how to use the AWS Management Console to create an IAM user for yourself and add that user to a group that has administrative permissions from an attached managed policy.

To create an administrator user for yourself and add the user to an administrators group (console)

1. Sign in to the [IAM console](#) as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

Note

We strongly recommend that you adhere to the best practice of using the **Administrator** IAM user below and securely lock away the root user credentials. Sign in as the root user only to perform a few [account and service management tasks](#).

2. Enable access to billing data for the IAM admin user that you will create as follows:
 - a. On the navigation bar, choose your account name, and then choose **My Account**.
 - b. Next to **IAM User and Role Access to Billing Information**, choose **Edit**. You must be signed in as the root user for this section to be displayed on the account page.
 - c. Select the check box to **Activate IAM Access** and choose **Update**.
 - d. On the navigation bar, choose **Services** and then **IAM** to return to the IAM dashboard.

3. In the navigation pane, choose **Users** and then choose **Add user**.
4. On the **Details** page, do the following:
 - a. For **User name**, type **Administrator**.
 - b. Select the check box for **AWS Management Console access**, select **Custom password**, and then type your new password in the text box.
 - c. By default, AWS forces the new user to create a new password when first signing in. You can optionally clear the check box next to **User must create a new password at next sign-in** to allow the new user to reset their password after they sign in.
 - d. Choose **Next: Permissions**.
5. On the **Permissions** page, do the following:
 - a. Choose **Add user to group**.
 - b. Choose **Create group**.
 - c. In the **Create group** dialog box, for **Group name** type **Administrators**.
 - d. Select the check box for the **AdministratorAccess** policy.
 - e. Choose **Create group**.
 - f. Back on the page with the list of groups, select the check box for your new group. Choose **Refresh** if you don't see the new group in the list.
 - g. Choose **Next: Tags**.
6. (Optional) On the **Tags** page, add metadata to the user by attaching tags as key-value pairs. For more information, see [Tagging IAM users and roles \(p. 289\)](#).
7. Choose **Next: Review**. Verify the group memberships to be added to the new user. When you are ready to proceed, choose **Create user**.
8. (Optional) On the **Complete** page, you can download a .csv file with login information for the user, or send email with login instructions to the user.

You can use this same process to create more groups and users and to give your users access to your AWS account resources. To learn about using policies that restrict user permissions to specific AWS resources, see [Access management for AWS resources \(p. 350\)](#) and [Example IAM identity-based policies \(p. 389\)](#). To add additional users to the group after it's created, see [Adding and removing users in an IAM group \(p. 163\)](#).

Creating an IAM user and group (AWS CLI)

If you followed the steps in the previous section, you used the AWS Management Console to set up an administrators group while creating the IAM user in your AWS account. This procedure shows an alternative way to create a group.

Overview: Setting up an administrators group

1. Create a group and give it a name (for example, Admins). For more information, see [Creating a group \(AWS CLI\) \(p. 21\)](#).
2. Attach a policy that gives the group administrative permissions—access to all AWS actions and resources. For more information, see [Attaching a policy to the group \(AWS CLI\) \(p. 22\)](#).
3. Add at least one user to the group. For more information, see [Creating an IAM user in your AWS account \(p. 76\)](#).

Creating a group (AWS CLI)

This section shows how to create a group in the IAM system.

Requirements

Install the AWS Command Line Interface (AWS CLI). For more information, see [Installing the AWS CLI](#) in the *AWS Command Line Interface User Guide*.

To create an administrators group (AWS CLI)

1. Type the `aws iam create-group` command with the name you've chosen for the group. Optionally, you can include a path as part of the group name. For more information about paths, see [Friendly names and paths \(p. 600\)](#). The name can consist of letters, digits, and the following characters: plus (+), equal (=), comma (,), period (.), at (@), underscore (_), and hyphen (-). The name is not case sensitive and can be a maximum of 128 characters in length.

In this example, you create a group named Admins.

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

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

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

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

Attaching a policy to the group (AWS CLI)

This section shows how to attach a policy that lets any user in the group perform any action on any resource in the AWS account. You do this by attaching the [AWS managed policy \(p. 359\)](#) called AdministratorAccess to the Admins group. For more information about policies, see [Access management for AWS resources \(p. 350\)](#).

To add a policy giving full administrator permissions (AWS CLI)

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

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

If the command is successful, there is no response.

2. Type the `aws iam list-attached-group-policies` command to confirm the policy is attached to the Admins group.

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

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

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

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

Important

After you have the administrators group set up, you must add at least one user to it. For more information about adding users to a group, see [Creating an IAM user in your AWS account \(p. 76\)](#).

Related resources

For related information found in the *Amazon Web Services General Reference*, see the following resources:

- [AWS Tasks that Require Account Root User](#)

For related information in the *IAM User Guide*, see the following resources:

- [Refining permissions in AWS using last accessed information \(p. 472\)](#)
- [IAM Tutorial: Delegate access to the billing console \(p. 30\)](#)

Creating your first IAM delegated user and group

To support multiple users in your AWS account, you must delegate permission to allow other people to perform only the actions you want to allow. To do this, create an IAM group with the permissions those people need and then add IAM users to the necessary groups as you create them. You can use this process to set up the groups, users, and permissions for your entire AWS account.

This solution is best used by small and medium organizations where an AWS administrator can manually manage the users and groups. For large organizations, you can use [custom IAM roles \(p. 206\)](#), [federation \(p. 176\)](#), or [single sign-on](#).

Creating a delegated IAM user and group (console)

You can use the AWS Management Console to create an IAM group with delegated permissions, and then create an IAM user for someone else and add it to the group.

To create a delegated group and user for someone else (console)

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 on the left, choose **Policies**.

If this is your first time choosing **Policies**, the **Welcome to Managed Policies** page appears. Choose **Get Started**.

3. Choose **Create policy**.
4. Choose the **JSON** tab and on the right side of the window choose **Import managed policy**.
5. In the **Import managed policies** window, type **power** to reduce the list of policies. Then select the button next to the **PowerUserAccess** AWS managed policy.
6. Choose **Import**.

The imported policy is added to your JSON policy.

7. Choose **Review policy**.
8. On the **Review** page, for **Name**, type **PowerUserExampleCorp**. For **Description**, type **Allows full access to all services except those for user management**. Then choose **Create policy** to save your work.
9. In the navigation pane, choose **Groups** and then choose **Create New Group**.
10. In the **Group Name** box, type **PowerUsers**.
11. In the list of policies, select the check box next to **PowerUserExampleCorp**. Then choose **Next Step**.
12. Choose **Create Group**.
13. In the navigation pane, choose **Users** and then choose **Add user**.
14. For **User name**, type **mary.major@examplecorp.com**.
15. Choose **Add another user** and type **diego.ramirez@examplecorp.com** for the second user.
16. Select the check box next to **AWS Management Console access** and select **Autogenerated password**. By default, AWS forces the new user to create a new password when first signing in. Select the check box next to **User must create a new password at next sign-in**.
17. Choose **Next: Permissions**.
18. On the **Set permissions** page, do not add permissions to the users. You will add a policy after the user confirms that they have changed their password and signed in.
19. Choose **Next: Tags**.
20. (Optional) Add metadata to the user by attaching tags as key-value pairs. For more information about using tags in IAM, see [Tagging IAM users and roles \(p. 289\)](#).
21. Choose **Next: Review** to see the list of group memberships to be added to the new user. When you are ready to proceed, choose **Create users**.
22. Download or copy the passwords for your new users and deliver them to the users securely. Separately, provide your users with a link to your IAM user console page and their user names.
23. After your user confirms that they can successfully sign in, go to the **Permissions** tab. Choose **Add permissions**. Choose **Add user to group**, and then select the check box next to **PowerUsers**.

Reducing the group permissions

Members of the **PowerUser** group have full access to all services except a few that provide user management actions (like IAM and Organizations). After a predefined period of inactivity (such as 90 days) has passed, you can review the services that your group members have accessed. Then you can reduce the permissions of the **PowerUserExampleCorp** policy to include only the services that your team needs.

For more information about the last accessed information, see [Refining permissions in AWS using last accessed information \(p. 472\)](#).

Reviewing last accessed information

Wait for a predefined period of inactivity (such as 90 days) to pass. Then you can review the last accessed information for your users or groups to learn when your users last attempted to access the services that your **PowerUserExampleCorp** policy allows.

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, choose **Groups** and then choose the **PowerUser** group name.
3. On the group summary page, choose the **Access Advisor** tab.

The table of last accessed information shows when the group members last attempted to access each service, in chronological order from the most recent attempt. The table includes only the services that the policy allows. In this case, the **PowerUserExampleCorp** policy allows access to all AWS services.

4. Review the table and make a list of the services that your group members have recently accessed.

For example, assume that within the last month, your team has accessed only the Amazon EC2 and Amazon S3 services. But six months ago, they accessed Amazon EC2 Auto Scaling and IAM. You know that they were investigating EC2 Auto Scaling, but decided that it wasn't necessary. You also know that they used IAM to create a role to allow Amazon EC2 to access data in an S3 bucket. So you decide to scale back the user's permissions to allow access to only the Amazon EC2 and Amazon S3 services.

Editing a policy to reduce permissions

After you review your last accessed information, you can edit your policy to allow access to only the services that your users need.

To use data to allow access to only necessary services

1. In the navigation pane, choose **Policies** and then choose the **PowerUserExampleCorp** policy name.
2. Choose **Edit policy**, and then choose the **JSON** tab.
3. Edit the JSON policy document to allow only the services you want.

For example, edit the first statement that includes the **Allow** effect and the **NotAction** element to allow only Amazon EC2 and Amazon S3 actions. To do this, replace it with the statement with the **FullAccessToSomeServices** ID. Your new policy will look like the following example policy.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "FullAccessToSomeServices",  
            "Effect": "Allow",  
            "Action": "AmazonEC2:*",  
            "Resource": "*"  
        },  
        {  
            "Sid": "FullAccessToSomeServices",  
            "Effect": "Allow",  
            "Action": "AmazonS3:*",  
            "Resource": "*"  
        }  
    ]  
}
```

```
        "Effect": "Allow",
        "Action": [
            "ec2:*",
            "s3:)"
        ],
        "Resource": "*"
    },
    {
        "Effect": "Allow",
        "Action": [
            "iam:CreateServiceLinkedRole",
            "iam>DeleteServiceLinkedRole",
            "iam>ListRoles",
            "organizations:DescribeOrganization"
        ],
        "Resource": "*"
    }
]
```

4. To further reduce your policies' permissions to specific actions and resources, view your events in CloudTrail **Event history**. There you can view detailed information about the specific actions and resources that your user has accessed. For more information, see [Viewing CloudTrail Events in the CloudTrail Console](#) in the *AWS CloudTrail User Guide*.

How IAM users sign in to your AWS account

After you create IAM users (with passwords), those users can sign in to the AWS Management Console. To sign in, they need your account ID or alias. They can also sign in from a custom URL that includes your account ID.

Note

If your company has an existing identity system, you might want to create a single sign-on (SSO) option. SSO gives users access to the AWS Management Console without requiring them to have an IAM user identity. SSO also eliminates the need for users to sign in to your organization's site and to AWS separately. For more information, see [Enabling custom identity broker access to the AWS console \(p. 206\)](#).

Before you create a sign-in URL for your account, you create an account alias so that the URL includes your account name instead of an account ID. For more information, see [Your AWS account ID and its alias \(p. 67\)](#).

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

IAM users sign-in link:

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

[Customize](#) | [Copy Link](#)

To create a sign-in URL for your IAM users, use the following pattern:

`https://account-ID-or-alias.signin.aws.amazon.com/console`

IAM users can also sign in at the following endpoint and enter the account ID or alias manually, instead of using your custom URL:

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

Permissions required for console activities

IAM users in your account have access only to the AWS resources that you specify in a policy. That policy must be attached to the user or to an IAM group that the user belongs to. To work in the console, users must have permissions to perform the actions that the console performs, such as listing and creating AWS resources. For more information, see [Access management for AWS resources \(p. 350\)](#) and [Example IAM identity-based policies \(p. 389\)](#).

If users in your account need programmatic access, you can create an access key pair (an access key ID and a secret access key) for each user. For more information, see [Managing access keys \(console\) \(p. 104\)](#).

Logging sign-in details in CloudTrail

If you enable CloudTrail to log sign-in events, you must understand how CloudTrail logs the events. CloudTrail includes global and Regions log entries. Where a sign-in event is logged in CloudTrail depends on how your users sign in. For details, see [Logging IAM Events with CloudTrail](#).

IAM console search

As you navigate through the IAM Management Console to manage various IAM resources, you often need to locate access keys. Or you might need to browse to the deeply nested IAM resources to find what you need. A faster option is to use the IAM console search page. You can locate access keys related to your account, IAM entities (such as users, groups, roles, identity providers), policies by name, and more.

The IAM console search feature can locate any of the following:

- IAM entity names that match your search keywords (for users, groups, roles, identity providers, and policies)
- AWS documentation topic names that match your search keywords
- Tasks that match your search keywords

The IAM console search feature does not return information about IAM Access Analyzer.

Every line in the search result is an active link. For example, you can choose the user name in the search result, which takes you to that user's detail page. Or you can choose an action link, for example **Create user**, to go to the [Create User](#) page.

Note

Access key search requires you to type the full access key ID in the search box. The search result shows the user associated with that key. From there you can navigate directly to that user's page, where you can manage their access key.

Using IAM console search

Use the **Search** page in the IAM console to find items related to that account.

To search for items in the IAM console

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, choose **Search**.
3. In the **Search** box, type your search keywords.

4. Choose a link in the search results list to navigate to the corresponding part of the console or documentation.

Icons in the IAM console search results

The following icons identify the types of items that are found by a search:

Icon	Description
	IAM users
	IAM groups
	IAM roles
	IAM policies
	Tasks such as "create user" or "attach policy"
	Results from the keyword delete
	IAM documentation

Sample search phrases

You can use the following phrases in the IAM search. Replace terms in italics with the names of actual IAM users, groups, roles, access keys, policies, or identity providers respectively that you want to locate.

- `user_name` or `group_name` or `role_name` or `policy_name` or `identity_provider_name`
- `access_key`
- `add user user_name to groups` or `add users to group group_name`
- `remove user user_name from groups`
- `delete user_name` or `delete group_name` or `delete role_name`, or `delete policy_name`, or `delete identity_provider_name`
- `manage access keys user_name`
- `manage signing certificates user_name`
- `users`
- `manage MFA for user_name`
- `manage password for user_name`
- `create role`
- `password policy`

- edit trust policy for role *role_name*
- show policy document for role *role_name*
- attach policy to *role_name*
- create managed policy
- create user
- create group
- attach policy to *group_name*
- attach entities to *policy_name*
- detach entities to *policy_name*
- what is IAM
- how do I create an IAM user
- how do I use IAM console
- what is a user or what is a group, or what is a policy, or what is a role, or what is an identity provider

IAM tutorials

The following tutorials present complete end-to-end procedures for common tasks for AWS Identity and Access Management (IAM). They are intended for a lab-type environment, with fictitious company names, user names, and so on. Their purpose is to provide general guidance. They are not intended for direct use in a production environment without careful review and adaptation to meet the unique needs of your organization's environment.

Tutorials

- [IAM Tutorial: Delegate access to the billing console \(p. 30\)](#)
- [IAM Tutorial: Delegate access across AWS accounts using IAM roles \(p. 34\)](#)
- [IAM Tutorial: Create and attach your first customer managed policy \(p. 43\)](#)
- [IAM Tutorial: Define permissions to access AWS resources based on tags \(p. 45\)](#)
- [IAM Tutorial: Enable users to manage their credentials and MFA settings \(p. 60\)](#)

IAM Tutorial: Delegate access to the billing console

AWS account owners can delegate access to specific IAM users who need to view or manage the AWS Billing and Cost Management data for an AWS account. The instructions that follow will help you set up a pretested scenario. This scenario helps you gain hands-on experience configuring billing permissions without concern for affecting your main AWS production account. If you attach a managed policy to your IAM users instead of following this tutorial, you must first activate access to the AWS Billing and Cost Management console in [Step 1 \(p. 31\)](#).

This workflow has four basic steps.

Step 1: Activate access to billing data on your AWS test account (p. 31)

If you create a single AWS account, only the AWS account owner ([AWS account root user \(p. 333\)](#)) has access to view and manage billing information. IAM users cannot access billing data until the account owner activates IAM access and also attaches policies that provide billing actions to the user or role. To view additional tasks that require you to sign in as the root user, see [AWS Tasks that Require Account Root User](#).

If you [create a member account](#) using AWS Organizations, this feature is enabled by default.

Step 2: Create IAM policies that grant permissions to billing data (p. 31)

After enabling billing access on your account, you must still explicitly grant access to billing data to specific IAM users or groups. You grant this access with a customer managed policy.

Step 3: Attach billing policies to your groups (p. 32)

When you attach a policy to a group, all members of that group receive the complete set of access permissions that are associated with that policy. In this scenario, you attach the new billing policies to groups containing only those users who require the billing access.

Step 4: Test access to the billing console (p. 32)

After you've completed the core tasks, you're ready to test the policy. Testing ensures that the policy works the way you want it to.

Prerequisites

Create a test AWS account to use with this tutorial. In this account create two test users and two test groups as summarized in the following table. Be sure to assign a password to each user so that you can sign in later in Step 4.

<i>Create user accounts</i>	<i>Create and configure group accounts</i>	
FinanceManager	BillingFullAccessGroup	FinanceManager
FinanceUser	BillingViewAccessGroup	FinanceUser

Step 1: Activate access to billing data on your AWS test account

First, activate billing access for your test users. To do this, see [Activating Access to the Billing and Cost Management Console in the AWS Billing and Cost Management User Guide](#).

Note

If you [create a member account](#) using AWS Organizations, this feature is enabled by default.

Step 2: Create IAM policies that grant permissions to billing data

Next, create custom policies that grant both view and full access permissions to the pages within the Billing and Cost Management console. For general information about IAM permissions policies, see [Managed Policies and Inline Policies \(p. 359\)](#).

To create IAM policies that grant permissions to billing data

1. Sign in to the AWS Management Console as a user with administrator credentials. To adhere to IAM best practices, don't sign in with your root user credentials. For more information, see [Create individual IAM users \(p. 528\)](#).
2. Open the IAM console at <https://console.aws.amazon.com/iam/>.
3. In the navigation pane, choose **Policies**, and then choose **Create policy**.
4. On the **Visual editor** tab, choose **Choose a service** to get started. Then choose **Billing**.
5. Follow these steps to create two policies:

Full access

- a. Choose **Select actions** and then select the check box next to **All Actions (*)**. You do not need to select a resource or condition for this policy.
- b. Choose **Review policy**.
- c. On the **Review** page, next to **Name**, type **BillingFullAccess**, and then choose **Create policy** to save it.

Read-only access

- a. Repeat steps [3 and 4 \(p. 31\)](#).
- b. Choose **Select actions** and then select the check box next to **Read**. You do not need to select a resource or condition for this policy.

- c. Choose **Review policy**.
- d. On the **Review** page, for **Name**, type **BillingViewAccess**. Then choose **Create policy** to save it.

To review descriptions for each of the permissions available in IAM policies that grant users access to the Billing and Cost Management console, see [Billing Permissions Descriptions](#).

Step 3: Attach billing policies to your groups

Now that you have custom billing policies available, you can attach them to their corresponding groups that you created earlier. Although you can attach a policy directly to a user or role, we recommend (in accordance with IAM best practices) that you use groups instead. For more information, see [Use groups to assign permissions to IAM users \(p. 528\)](#).

To attach billing policies to your groups

1. In the navigation pane, choose **Policies** to display the full list of policies available to your AWS account. To attach each policy to its appropriate group, follow these steps:

Full access

- a. In the policy search box, type **BillingFullAccess**, and then select the check box next to the policy name.
- b. Choose **Policy actions**, and then choose **Attach**.
- c. In the identity (user, group, and role) search box, type **BillingFullAccessGroup**, select the check box next to the name of the group, and then choose **Attach policy**.

Read-only access

- a. In the policy search box, type **BillingViewAccess**, and then select the check box next to the policy name.
 - b. Choose **Policy actions**, and then choose **Attach**.
 - c. In the identity (user, group, and role) search box, type **BillingViewAccessGroup**, select the check box next to the name of the group, and then choose **Attach policy**.
2. Sign out of the console, and then proceed to [Step 4: Test access to the billing console \(p. 32\)](#).

Step 4: Test access to the billing console

We recommend that you test access by signing in as each of the test users to learn what your users might experience. Use the following steps to sign in using both test accounts to see the difference between access rights.

To test billing access by signing in with both test user accounts

1. Use your AWS account ID or account alias, your IAM user name, and your password to sign in to the [IAM console](#).

Note

For your convenience, the AWS sign-in page uses a browser cookie to remember your IAM user name and account information. If you previously signed in as a different user, choose **Sign in to a different account** near the bottom of the page to return to the main sign-in page. From there, you can type your AWS account ID or account alias to be redirected to the IAM user sign-in page for your account.

2. Sign in with each account using the steps provided below so you can compare the different user experiences.

Full access

- a. Sign in to your AWS account as the user FinanceManager.
- b. On the navigation bar, choose **FinanceManager@<account alias or ID number>**, and then choose **My Billing Dashboard**.
- c. Browse through the pages and choose the various buttons to ensure that you have full modify permissions.

Read-only access

- a. Sign in to your AWS account as the user FinanceUser.
- b. On the navigation bar, choose **FinanceUser@<account alias or ID number>**, and then choose **My Billing Dashboard**.
- c. Browse through the pages. Notice that you can display costs, reports, and billing data with no problems. However, if you choose an option to modify a value, you receive an **Access Denied** message. For example, on the **Preferences** page, choose any of the check boxes on the page, and then choose **Save preferences**. The console message informs you that you need **ModifyBilling** permissions to make changes to that page.

Related resources

For related information found in the *AWS Billing and Cost Management User Guide*, see the following resources:

- [Activating Access to the Billing and Cost Management Console](#)
- [Example 4: Allow full access to AWS services but deny IAM users access to the Billing and Cost Management console.](#)
- [Billing Permissions Descriptions](#)

For related information in the *IAM User Guide*, see the following resources:

- [Managed policies and inline policies \(p. 359\)](#)
- [Controlling user access to the AWS Management Console \(p. 80\)](#)
- [Attaching a policy to an IAM group \(p. 164\)](#)

Summary

You've now successfully completed all of the steps necessary to delegate user access to the Billing and Cost Management console. As a result, you've seen firsthand what your users' billing console experience will be like. You can now proceed to implement this logic in your production environment at your convenience.

IAM Tutorial: Delegate access across AWS accounts using IAM roles

This tutorial teaches you how to use a role to delegate access to resources that are in different AWS accounts that you own (Production and Development). You share resources in one account with users in a different account. By setting up cross-account access in this way, you don't need to create individual IAM users in each account. In addition, users don't have to sign out of one account and sign into another in order to access resources in different AWS accounts. After configuring the role, you see how to use the role from the AWS Management Console, the AWS CLI, and the API.

Note

IAM roles and resource-based policies delegate access across accounts only within a single partition. For example, assume that you have an account in US West (N. California) in the standard aws partition. You also have an account in China (Beijing) in the aws-cn partition. You can't use an Amazon S3 resource-based policy in your account in China (Beijing) to allow access for users in your standard aws account.

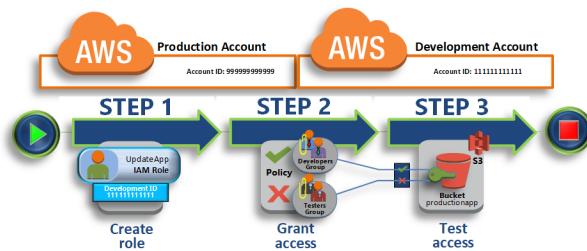
In this tutorial, imagine that the Production account is where live applications are managed. The Development account is a sandbox where developers and testers can freely test applications. In each account, application information is stored in Amazon S3 buckets. You manage IAM users in the Development account, where you have two IAM groups: Developers and Testers. Users in both groups have permissions to work in the Development account and access resources there. From time to time, a developer must update the live applications in the Production account. These applications are stored in an Amazon S3 bucket called `productionapp`.

At the end of this tutorial, you have the following:

- Users in the Development account (the trusted account) that are allowed to assume a specific role in the Production account.
- A role in the Production account (the trusting account) that is allowed to access a specific Amazon S3 bucket.
- The `productionapp` bucket in the Production account.

Developers can use the role in the AWS Management Console to access the `productionapp` bucket in the Production account. They can also access the bucket by using API calls that are authenticated by temporary credentials provided by the role. Similar attempts by a Tester to use the role fail.

This workflow has three basic steps.



Step 1: Create a role (p. 35)

First, you use the AWS Management Console to establish trust between the Production account (ID number 999999999999) and the Development account (ID number 111111111111). You start by creating an IAM role named *UpdateApp*. When you create the role, you define the Development account as a trusted entity and specify a permissions policy that allows trusted users to update the `productionapp` bucket.

Step 2: Grant access to the role (p. 37)

In this step of the tutorial, you modify the IAM group policy so that Testers are denied access to the UpdateApp role. Because Testers have PowerUser access in this scenario, we must explicitly deny the ability to use the role.

Step 3: Test access by switching roles (p. 39)

Finally, as a Developer, you use the UpdateApp role to update the productionapp bucket in the Production account. You see how to access the role through the AWS console, the AWS CLI, and the API.

Prerequisites

This tutorial assumes that you have the following already in place:

- Two separate AWS accounts that you can use, one to represent the Development account, and one to represent the Production account.
- Users and groups in the Development account created and configured as follows:

User	Group	Permissions
David	Developers	Both users are able to sign in and use the AWS Management Console in the Development account.
Theresa		

- You do not need to have any users or groups created in the Production account.
- An Amazon S3 bucket created in the Production account. We call it `ProductionApp` in this tutorial, but because S3 bucket names must be globally unique, you must use a bucket with a different name.

Step 1: Create a role

You can allow users from one AWS account to access resources in another AWS account. To do this, create a role that defines who can access it and what permissions it grants to users that switch to it.

In this step of the tutorial, you create the role in the Production account and specify the Development account as a trusted entity. You also limit the role's permissions to only read and write access to the `productionapp` bucket. Anyone who is granted permission to use the role can read and write to the `productionapp` bucket.

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

To obtain the development AWS account ID

1. Sign in to the AWS Management Console as an administrator of the Development account, and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation bar, choose **Support**, and then **Support Center**. Your currently signed-in 12-digit account number (ID) appears in the **Support Center** navigation pane. For this scenario, we pretend the Development account ID is 111111111111. However, you should use a valid account ID if you are reconstructing the scenario in your test environment.

To create a role in the production account that can be used by the development account

1. Sign in to the AWS Management Console as an administrator of the Production account, and open the IAM console.

2. Before creating the role, prepare the managed policy that defines the permissions that the role requires. You attach this policy to the role in a later step.

You want to set read and write access to the `productionapp` bucket. Although AWS provides some Amazon S3 managed policies, there isn't one that provides read and write access to a single Amazon S3 bucket. You can create your own policy instead.

In the navigation pane on the left, choose **Policies** and then choose **Create policy**.

3. Choose the **JSON** tab and copy the text from the following JSON policy document. Paste this text into the **JSON** text box, replacing the resource ARN (`arn:aws:s3:::productionapp`) with the real one appropriate to your S3 bucket.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "s3>ListAllMyBuckets",  
            "Resource": "*"  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3>ListBucket",  
                "s3:GetBucketLocation"  
            ],  
            "Resource": "arn:aws:s3:::productionapp"  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3.GetObject",  
                "s3:PutObject",  
                "s3>DeleteObject"  
            ],  
            "Resource": "arn:aws:s3:::productionapp/*"  
        }  
    ]  
}
```

The `ListBucket` permission allows users to view objects in the `productionapp` bucket. The `GetObject`, `PutObject`, `DeleteObject` permissions allows users to view, update, and delete contents in the `productionapp` bucket.

4. When you are finished, choose **Review policy**. The [Policy Validator \(p. 444\)](#) reports any syntax errors.

Note

You can switch between the **Visual editor** and **JSON** tabs anytime. However, if you make changes or choose **Review policy** in the **Visual editor** tab, IAM might restructure your policy to optimize it for the visual editor. For more information, see [Policy restructuring \(p. 571\)](#).

5. On the **Review** page, type `read-write-app-bucket` for the policy name. Review the policy **Summary** to see the permissions granted by your policy, and then choose **Create policy** to save your work.

The new policy appears in the list of managed policies.

6. In the navigation pane on the left, choose **Roles** and then choose **Create role**.
7. Choose the **Another AWS account** role type.
8. For **Account ID**, type the Development account ID.

This tutorial uses the example account ID **111111111111** for the Development account. You should use a valid account ID. If you use an invalid account ID, such as **111111111111**, IAM does not let you create the new role.

For now you do not need to require an external ID, or require users to have multi-factor authentication (MFA) in order to assume the role. So leave these options unselected. For more information, see [Using multi-factor authentication \(MFA\) in AWS \(p. 111\)](#)

9. Choose **Next: Permissions** to set the permissions that will be associated with the role.
10. Select the box next to the policy that you created previously.

Tip

For **Filter**, choose **Customer managed** to filter the list to include only the policies that you have created. This hides the AWS created policies and makes it much easier to find the one you're looking for.

Then choose **Next: Tags**.

11. (Optional) Add metadata to the user by attaching tags as key-value pairs. For more information about using tags in IAM, see [Tagging IAM users and roles \(p. 289\)](#).
12. Choose **Next: Review** and type **UpdateApp** for the role name.
13. (Optional) For **Role description**, type a description for the new role.
14. After reviewing the role, choose **Create role**.

The **UpdateApp** role appears in the list of roles.

Now you must obtain the role's Amazon Resource Name (ARN), which is a unique identifier for the role. When you modify the Developers and Testers group's policy, you will specify the role's ARN to grant or deny permissions.

To obtain the ARN for UpdateApp

1. In the navigation pane of the IAM console, choose **Roles**.
2. In the list of roles, choose the **UpdateApp** role.
3. In the **Summary** section of the details pane, copy the **Role ARN** value.

The Production account has an account ID of 999999999999, so the role ARN is `arn:aws:iam::999999999999:role/UpdateApp`. Ensure that you supply the real AWS account ID for your 'production' account.

At this point, you have established trust between the Production and Development accounts. You did this by creating a role in the Production account that identifies the Development account as a trusted principal. You also defined what users who switch to the **UpdateApp** role can do.

Next, modify the permissions for the groups.

Step 2: Grant access to the role

At this point, both Testers and Developers group members have permissions that allow them to freely test applications in the Development account. Here are the steps required to add permissions to allow switching to the role.

To modify the developers group to allow them to switch to the UpdateApp role

1. Sign in as an administrator in the Development account, and open the IAM console.
2. Choose **Groups**, and then choose **Developers**.

3. Choose the **Permissions** tab, expand the **Inline Policies** section, and then choose **Create Group Policy**. If no inline policy exists yet, then the button does not appear. Instead, choose the link at the end of "To create one, click here."
4. Choose **Custom Policy** and then choose **Select** button.
5. Type a policy name like **allow-assume-S3-role-in-production**.
6. Add the following policy statement to allow the AssumeRole action on the UpdateApp role in the Production account. Be sure that you change **PRODUCTION-ACCOUNT-ID** in the Resource element to the actual AWS account ID of the Production account.

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

The Allow effect explicitly allows the Developers group access to the UpdateApp role in the Production account. Any developer who tries to access the role will succeed.

7. Choose **Apply Policy** to add the policy to the Developer group.

In most environments, the following procedure is likely not needed. If, however, you use Power User permissions, then some groups might already be able to switch roles. The following procedure shows how to add a "Deny" permission to the Testers group to ensure that they cannot assume the role. If this procedure is not needed in your environment, then we recommend that you do not add it. "Deny" permissions make the overall permissions picture more complicated to manage and understand. Use "Deny" permissions only when there is not a better option.

To modify the testers group to deny permission to assume the UpdateApp role

1. Choose **Groups**, and then choose **Testers**.
2. Choose the **Permissions** tab, expand the **Inline Policies** section, and then choose **Create Group Policy**.
3. Choose **Custom Policy** and then choose the **Select** button.
4. Type a policy name like **deny-assume-S3-role-in-production**.
5. Add the following policy statement to deny the AssumeRole action on the UpdateApp role. Be sure that you change **PRODUCTION-ACCOUNT-ID** in the Resource element to the actual AWS account ID of the Production account.

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

The Deny effect explicitly denies the Testers group access to the UpdateApp role in the Production account. Any tester who tries to access the role will get an access denied message.

6. Choose **Apply Policy** to add the policy to the Tester group.

The Developers group now has permissions to use the `UpdateApp` role in the Production account. The Testers group is prevented from using the `UpdateApp` role.

Next, you'll learn how David, a developer, can access the `productionapp` bucket in the Production account. David can access the bucket from the AWS Management Console, the AWS CLI, or the AWS API.

Step 3: Test access by switching roles

After completing the first two steps of this tutorial, you have a role that grants access to a resource in the Production account. You also have one group in the Development account whose users are allowed to use that role. The role is now ready to use. This step discusses how to test switching to that role from the AWS Management Console, the AWS CLI, and the AWS API.

Important

You can switch to a role only when you are signed in as an IAM user or a federated user.

Additionally, if you launch an Amazon EC2 instance to run an application, the application can assume a role through its instance profile. You cannot switch to a role when you are signed in as the AWS account root user.

Switch roles (console)

If David needs to work with in the Production environment in the AWS Management Console, he can do so by using **Switch Role**. He specifies the account ID or alias and the role name, and his permissions immediately switch to those permitted by the role. He can then use the console to work with the `productionapp` bucket, but cannot work with any other resources in Production. While David is using the role, he also cannot make use of his power-user privileges in the Development account. That's because only one set of permissions can be in effect at a time.

Important

Switching roles using the AWS Management Console works only with accounts that do not require an `ExternalId`. For example, assume that you grant access to your account to a third party and require an `ExternalId` in a `Condition` element in your permissions policy. In that case, the third party can access your account only by using the AWS API or a command line tool. The third party cannot use the console because it cannot supply a value for `ExternalId`. For more information about this scenario, see [How to use an external ID when granting access to your AWS resources to a third party \(p. 225\)](#), and [How to Enable Cross-Account Access to the AWS Management Console in the AWS Security Blog](#).

There are two ways that David can use to enter the **Switch Role** page:

- David receives a link from his administrator that points to a pre-defined Switch Role configuration. The link is provided to the administrator on the final page of the **Create role** wizard or on the **Role Summary** page for a cross-account role. Choosing this link takes David to the **Switch Role** page with the **Account ID** and **Role name** fields already filled in. All David needs to do is choose **Switch Role** and he's done.
- The administrator does not send the link in email, but instead sends the **Account ID** number and **Role Name** values. David must manually type them to switch roles. This is illustrated in the following procedure.

To assume a role

1. David signs into the AWS Management Console using his normal user that is in the Development group.
2. He chooses the link that his administrator sent to him in email. This takes him to the **Switch Role** page with the account ID or alias and the role name information already filled in.

—or—

He chooses his name (the Identity menu) on the navigation bar, and then chooses **Switch Role**.

If this is the first time that David tries to access the Switch Role page this way, he will first land on a first-run **Switch Role** page. This page provides additional information on how switching roles can enable users to manage resources across AWS accounts. David must choose the **Switch Role** button on this page to complete the rest of this procedure.

3. Next, in order to access the role, David must manually type the Production account ID number (999999999999) and the role name (UpdateApp).

Also, David wants to monitor which roles (and associated permissions) are currently active. To keep track of this information, he types PRODUCTION in the **Display Name** text box, selects the red color option, and then chooses **Switch Role**.

4. David can now use the Amazon S3 console to work with the Amazon S3 bucket, or any other resource to which the UpdateApp role has permissions.
5. When he is done with the work he needs to do, David can return to his original permissions. To do that, he chooses the **PRODUCTION** role display name on the navigation bar and then chooses **Back to David @ 111111111111**.
6. The next time that David wants to switch roles and chooses the Identity menu in the navigation bar, he sees the PRODUCTION entry still there from last time. He can simply choose that entry to switch roles immediately without having to reenter the account ID and role name.

Switch roles (AWS CLI)

If David needs to work in the Production environment at the command line, he can do so by using the [AWS CLI](#). He runs the `aws sts assume-role` command and passes the role ARN to get temporary security credentials for that role. He then configures those credentials in environment variables so subsequent AWS CLI commands work using the role's permissions. While David is using the role, he cannot use his power-user privileges in the Development account. The reason is that only one set of permissions can be in effect at a time.

Note that all access keys and tokens are examples only and cannot be used as shown. Replace with the appropriate values from your live environment.

To assume a role

1. David opens a command prompt window, and confirms that the AWS CLI client is working by running the command:

```
aws help
```

Note

David's default environment uses the David user credentials from his default profile that he created with the `aws configure` command. For more information, see [Configuring the AWS Command Line Interface](#) in the [AWS Command Line Interface User Guide](#).

2. He begins the switch role process by running the following command to switch to the UpdateApp role in the Production account. He got the role ARN from the administrator that created the role. The command requires that you provide a session name as well, you can choose any text you like for that.

```
aws sts assume-role --role-arn "arn:aws:iam::999999999999:role/UpdateApp" --role-session-name "David-ProdUpdate"
```

David then sees the following in the output:

```
{
    "Credentials": {
        "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
        "SessionToken": "AQoDYXdzEGcaEXAMPLE2gsYULo
+Im5ZEXAMLeEYjs1M2FUIgIJx9tQqNMBEXAMPLE
CvSRyh0FW7jEXAMPLEW+vE/7s1HRpXviG7b+qYf4nD00EXAMPLEmj4wxS04L/
uZEXAMPLECihzFB5lTYLto9dyBgSDy
EXAMPLE9/
g7QRUhZp4bqbEXAMPLENwGPY0j59pFA41NKC1kVgkREXAMPLEjlzxQ7y52gekeVEXAMPLEDiB9ST3Uuysg
sKdEXAMPLE1TVastU1AOSKFEXAMPLEiywCC/Cs8EXAMPLEpZgOs+6hz4AP4KEXAMPLERbASP
+4eZScEXAMPLEsnf87e
NhyDHq6ikBQ==",
        "Expiration": "2014-12-11T23:08:07Z",
        "AccessKeyId": "AKIAIOSFODNN7EXAMPLE"
    }
}
```

- David sees the three pieces that he needs in the Credentials section of the output.

- AccessKeyId
- SecretAccessKey
- SessionToken

David needs to configure the AWS CLI environment to use these parameters in subsequent calls. For information about the various ways to configure your credentials, see [Configuring the AWS Command Line Interface](#). You cannot use the `aws configure` command because it does not support capturing the session token. However, you can manually type the information into a configuration file. Because these are temporary credentials with a relatively short expiration time, it is easiest to add them to the environment of your current command line session.

- To add the three values to the environment, David cuts and pastes the output of the previous step into the following commands. You might want to cut and paste into a simple text editor to address line wrap issues in the output of the session token. It must be added as a single long string, even though it is shown line wrapped here for clarity.

Note

The following example shows commands given in the Windows environment, where "set" is the command to create an environment variable. On a Linux or macOS computer, you would use the command "export" instead. All other parts of the example are valid in all three environments.

```
set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
set AWS_SESSION_TOKEN=AQoDYXdzEGcaEXAMPLE2gsYULo
+Im5ZEXAMLeEYjs1M2FUIgIJx9tQqNMBEXAMPLECvS
Ryh0FW7jEXAMPLEW+vE/7s1HRpXviG7b+qYf4nD00EXAMPLEmj4wxS04L/
uZEXAMPLECihzFB5lTYLto9dyBgSDyEXA
MLEKEY9/
g7QRUhZp4bqbEXAMPLENwGPY0j59pFA41NKC1kVgkREXAMPLEjlzxQ7y52gekeVEXAMPLEDiB9ST3UusKd
EXAMPLE1TVastU1AOSKFEXAMPLEiywCC/Cs8EXAMPLEpZgOs+6hz4AP4KEXAMPLERbASP
+4eZScEXAMPLENhykxiHen
DHq6ikBQ==
```

At this point, any following commands run under the permissions of the role identified by those credentials. In David's case, the `UpdateApp` role.

- Run the command to access the resources in the Production account. In this example, David simply lists the contents of his S3 bucket with the following command.

```
aws s3 ls s3://productionapp
```

Because Amazon S3 bucket names are universally unique, there is no need to specify the account ID that owns the bucket. To access resources for other AWS services, refer to the AWS CLI documentation for that service for the commands and syntax that are required to reference its resources.

Using AssumeRole (AWS API)

When David needs to make an update to the Production account from code, he makes an `AssumeRole` call to assume the `UpdateApp` role. The call returns temporary credentials that he can use to access the `productionapp` bucket in the Production account. With those credentials, David can make API calls to update the `productionapp` bucket. However, he cannot make API calls to access any other resources in the Production account, even though he has power-user permissions in the Development account.

To assume a role

1. David calls `AssumeRole` as part of an application. He must specify the `UpdateApp` ARN:
`arn:aws:iam::999999999999:role/UpdateApp`.

The response from the `AssumeRole` call includes the temporary credentials with an `AccessKeyId` and a `SecretAccessKey`. It also includes an `Expiration` time that indicates when the credentials expire and you must request new ones.

2. With the temporary credentials, David makes an `s3:PutObject` call to update the `productionapp` bucket. He would pass the credentials to the API call as the `AuthParams` parameter. Because the temporary role credentials have only read and write access to the `productionapp` bucket, any other actions in the Production account are denied.

For a code example (using Python), see [Switching to an IAM role \(AWS API\) \(p. 262\)](#).

Related resources

- For more information about IAM users and groups, see [IAM Identities \(users, groups, and roles\) \(p. 72\)](#).
- For more information about Amazon S3 buckets, see [Create a Bucket](#) in the *Amazon Simple Storage Service Getting Started Guide*.
- To learn whether principals in accounts outside of your zone of trust (trusted organization or account) have access to assume your roles, see [What is IAM Access Analyzer?](#).

Summary

You have completed the cross-account API access tutorial. You created a role to establish trust with another account and defined what actions trusted entities can take. Then, you modified a group policy to control which IAM users can access the role. As a result, developers from the Development account can make updates to the `productionapp` bucket in the Production account by using temporary credentials.

IAM Tutorial: Create and attach your first customer managed policy

In this tutorial, you use the AWS Management Console to create a [customer managed policy \(p. 361\)](#) and then attach that policy to an IAM user in your AWS account. The policy you create allows an IAM test user to sign in directly to the AWS Management Console with read-only permissions.

This workflow has three basic steps:

Step 1: Create the policy (p. 43)

By default, IAM users do not have permissions to do anything. They cannot access the AWS Management Console or manage the data within unless you allow it. In this step, you create a customer managed policy that allows any attached user to sign in to the console.

Step 2: Attach the policy (p. 44)

When you attach a policy to a user, the user inherits all of the access permissions that are associated with that policy. In this step, you attach the new policy to a test user account.

Step 3: Test user access (p. 44)

Once the policy is attached, you can sign in as the user and test the policy.

Prerequisites

To perform the steps in this tutorial, you need to already have the following:

- An AWS account that you can sign in to as an IAM user with administrative permissions.
- A test IAM user that has no permissions assigned or group memberships as follows:

User name	Group	Permissions
PolicyUser	<none>	<none>

Step 1: Create the policy

In this step, you create a customer managed policy that allows any attached user to sign in to the AWS Management Console with read-only access to IAM data.

To create the policy for your test user

1. Sign in to the IAM console at <https://console.aws.amazon.com/iam/> with your user that has administrator permissions.
2. In the navigation pane, choose **Policies**.
3. In the content pane, choose **Create policy**.
4. Choose the **JSON** tab and copy the text from the following JSON policy document. Paste this text into the **JSON** text box.

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

```
"Statement": [ {  
    "Effect": "Allow",  
    "Action": [  
        "iam:GenerateCredentialReport",  
        "iam:Get*",  
        "iam>List*"  
    ],  
    "Resource": "*"  
} ]  
}
```

5. When you are finished, choose **Review policy**. The [Policy Validator \(p. 444\)](#) reports any syntax errors.

Note

You can switch between the **Visual editor** and **JSON** tabs anytime. However, if you make changes or choose **Review policy** in the **Visual editor** tab, IAM might restructure your policy to optimize it for the visual editor. For more information, see [Policy restructuring \(p. 571\)](#).

6. On the **Review** page, type **UsersReadOnlyAccessToIAMConsole** for the policy name. Review the policy **Summary** to see the permissions granted by your policy, and then choose **Create policy** to save your work.

The new policy appears in the list of managed policies and is ready to attach.

Step 2: Attach the policy

Next you attach the policy you just created to your test IAM user.

To attach the policy to your test user

1. In the IAM console, in the navigation pane, choose **Policies**.
2. At the top of the policy list, in the search box, start typing **UsersReadOnlyAccessToIAMConsole** until you can see your policy. Then check the box next to **UsersReadOnlyAccessToIAMConsole** in the list.
3. Choose the **Policy actions** button, and then chose **Attach**.
4. For **Filter**, choose **Users**.
5. In the search box, start typing **PolicyUser** until that user is visible on the list. Then check the box next to that user in the list.
6. Choose **Attach Policy**.

You have attached the policy to your IAM test user, which means that user now has read-only access to the IAM console.

Step 3: Test user access

For this tutorial, we recommend that you test access by signing in as the test user so you can see what your users might experience.

To test access by signing in with your test user account

1. Sign in to the IAM console at <https://console.aws.amazon.com/iam/> with your **PolicyUser** test user.
2. Browse through the pages of the console and try to create a new user or group. Notice that **PolicyUser** can display data but cannot create or modify existing IAM data.

Related resources

For related information in the *IAM User Guide*, see the following resources:

- [Managed policies and inline policies \(p. 359\)](#)
- [Controlling user access to the AWS Management Console \(p. 80\)](#)
- [Create individual IAM users \(p. 528\)](#)

Summary

You've now successfully completed all of the steps necessary to create and attach a customer managed policy. As a result, you are able to sign in to the IAM console with your test account to see what the experience is like for your users.

IAM Tutorial: Define permissions to access AWS resources based on tags

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called *tags*. You can attach tags to IAM principals (users or roles) and to AWS resources. You can then define policies that use tag condition keys to grant permissions to your principals based on their tags. When you use tags to control access to your AWS resources, you allow your teams and resources to grow with fewer changes to AWS policies. ABAC policies are more flexible than traditional AWS policies, which require you to list each individual resource. For more information about ABAC and its advantage over traditional policies, see [What is ABAC for AWS? \(p. 13\)](#).

Topics

- [Tutorial overview \(p. 45\)](#)
- [Prerequisites \(p. 46\)](#)
- [Step 1: Create test users \(p. 47\)](#)
- [Step 2: Create the ABAC policy \(p. 48\)](#)
- [Step 3: Create roles \(p. 50\)](#)
- [Step 4: Test creating secrets \(p. 51\)](#)
- [Step 5: Test viewing secrets \(p. 53\)](#)
- [Step 6: Test scalability \(p. 54\)](#)
- [Step 7: Test updating and deleting secrets \(p. 55\)](#)
- [Summary \(p. 56\)](#)
- [Related resources \(p. 57\)](#)
- [IAM Tutorial: Use SAML session tags for ABAC \(p. 57\)](#)

Tutorial overview

This tutorial shows how to create and test a policy that allows IAM roles with principal tags to access resources with matching tags. When a principal makes a request to AWS, their permissions are granted based on whether the principal and resource tags match. This strategy allows individuals to view or edit only the AWS resources required for their jobs.

Scenario

Assume that you're a lead developer at a large company named Example Corporation, and you're an experienced IAM administrator. You're familiar with creating and managing IAM users, roles, and policies. You want to ensure that your development engineers and quality assurance team members can access the resources they need. You also need a strategy that scales as your company grows.

You choose to use AWS resource tags and IAM role principal tags to implement an ABAC strategy for services that support it, beginning with AWS Secrets Manager. To learn which services support authorization based on tags, see [AWS services that work with IAM \(p. 611\)](#). To learn which tagging condition keys you can use in a policy with each service's actions and resources, see [Actions, Resources, and Condition Keys for AWS Services](#). You can configure your SAML-based or web identity provider to pass [session tags \(p. 293\)](#) to AWS. When your employees federate into AWS, their attributes are applied to their resulting principal in AWS. You can then use ABAC to allow or deny permissions based on those attributes. To learn how using session tags with a SAML federated identity differs from this tutorial, see [IAM Tutorial: Use SAML session tags for ABAC \(p. 57\)](#).

Your Engineering and Quality Assurance team members are on either the **Pegasus** or **Unicorn** project. You choose the following 3-character project and team tag values:

- `access-project = peg` for the **Pegasus** project
- `access-project = uni` for the **Unicorn** project
- `access-team = eng` for the Engineering team
- `access-team = qas` for the Quality Assurance team

Additionally, you choose to require the `cost-center` cost allocation tag to enable custom AWS billing reports. For more information, see [Using Cost Allocation Tags](#) in the *AWS Billing and Cost Management User Guide*.

Summary of key decisions

- Employees sign in with IAM user credentials and then assume the IAM role for their team and project. If your company has its own identity system, you can set up federation to allow employees to assume a role without IAM users. For more information, see [IAM Tutorial: Use SAML session tags for ABAC \(p. 57\)](#).
- The same policy is attached to all of the roles. Actions are allowed or denied based on tags.
- Employees can create new resources, but only if they attach the same tags to the resource that are applied to their role. This ensures that employees can view the resource after they create it. Administrators are no longer required to update policies with the ARN of new resources.
- Employees can read resources owned by their team, regardless of the project.
- Employees can update and delete resources owned by their own team and project.
- IAM administrators can add a new role for new projects. They can create and tag a new IAM user to allow access to the appropriate role. Administrators are not required to edit a policy to support a new project or team member.

In this tutorial, you will tag each resource, tag your project roles, and add policies to the roles to allow the behavior previously described. The resulting policy allows the roles `Create`, `Read`, `Update`, and `Delete` access to resources that are tagged with the same project and team tags. The policy also allows cross-project `Read` access for resources that are tagged with the same team.

Prerequisites

To perform the steps in this tutorial, you must already have the following:

- An AWS account that you can sign in to as an IAM user with administrative permissions. If you have a new account and sign in as the AWS account root user, then [create an IAM admin user \(p. 20\)](#).

- Your 12-digit account ID, which you use to create the roles in step 3.

To find your AWS account ID number using the AWS Management Console, choose **Support** on the navigation bar on the upper right, and then choose **Support Center**. The account number (ID) appears in the navigation pane on the left.



Support Center X

Account number: 123412341234

- Experience creating and editing IAM users, roles, and policies in the AWS Management Console. However, if you need help remembering an IAM management process, this tutorial provides links where you can view step-by-step instructions.

Step 1: Create test users

For testing, create four IAM users with permissions to assume roles with the same tags. This makes it easier to add more users to your teams. When you tag the users, they automatically get access to assume the correct role. You don't have to add the users to the trust policy of the role if they work on only one project and team.

1. Create the following customer managed policy named `access-assume-role`. For more information about creating a JSON policy, see [Creating IAM policies \(console\) \(p. 439\)](#).

ABAC policy: Assume any ABAC role, but only when the user and role tags match

The following policy allows a user to assume any role in your account with the `access-` name prefix. The role must also be tagged with the same project, team, and cost center tags as the user.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "TutorialAssumeRole",
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Resource": "arn:aws:iam::123456789012:role/access-*",
            "Condition": {
                "StringEquals": {
                    "iam:ResourceTag/access-project": "${aws:PrincipalTag/access-project}",
                    "iam:ResourceTag/access-team": "${aws:PrincipalTag/access-team}",
                    "iam:ResourceTag/cost-center": "${aws:PrincipalTag/cost-center}"
                }
            }
        }
    ]
}
```

To scale this tutorial to a large number of users, you can attach the policy to a group and add each user to the group. For more information, see [Creating IAM groups \(p. 161\)](#) and [Adding and removing users in an IAM group \(p. 163\)](#).

2. Create the following IAM users, attach the `access-assume-role` permissions policy, and add the following tags. For more information about creating and tagging a new user, see [Creating IAM users \(console\) \(p. 77\)](#).

ABAC users

User name	User tags
access-Arnav-peg-eng	<code>access-project = peg</code> <code>access-team = eng</code> <code>cost-center = 987654</code>
access-Mary-peg-qas	<code>access-project = peg</code> <code>access-team = qas</code> <code>cost-center = 987654</code>
access-Saanvi-uni-eng	<code>access-project = uni</code> <code>access-team = eng</code> <code>cost-center = 123456</code>
access-Carlos-uni-qas	<code>access-project = uni</code> <code>access-team = qas</code> <code>cost-center = 123456</code>

Step 2: Create the ABAC policy

Create the following policy named `access-same-project-team`. You will add this policy to the roles in a later step. For more information about creating a JSON policy, see [Creating IAM policies \(console\) \(p. 439\)](#).

For additional policies that you can adapt for this tutorial, see the following pages:

- [Controlling access for IAM principals \(p. 386\)](#)
- [Amazon EC2: Allows starting or stopping EC2 instances a user has tagged, programmatically and in the console \(p. 410\)](#)
- [EC2: Start or stop instances based on matching principal and resource tags \(p. 411\)](#)
- [EC2: Start or stop instances based on tags \(p. 411\)](#)
- [IAM: Assume roles that have a specific tag \(p. 416\)](#)

ABAC Policy: Access Secrets Manager Resources Only When the Principal and Resource Tags Match

The following policy allows principals to create, read, edit, and delete resources, but only when those resources are tagged with the same key-value pairs as the principal. When a principal creates a resource, they must add `access-project`, `access-team`, and `cost-center` tags with values that match the principal's tags. The policy also allows adding optional `Name` or `OwnedBy` tags.

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

```
{
    "Sid": "AllActionsSecretsManagerSameProjectSameTeam",
    "Effect": "Allow",
    "Action": "secretsmanager:*",
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "aws:ResourceTag/access-project": "${aws:PrincipalTag/access-project}",
            "aws:ResourceTag/access-team": "${aws:PrincipalTag/access-team}",
            "aws:ResourceTag/cost-center": "${aws:PrincipalTag/cost-center}"
        },
        "ForAllValues:StringEquals": {
            "aws:TagKeys": [
                "access-project",
                "access-team",
                "cost-center",
                "Name",
                "OwnedBy"
            ]
        },
        "StringEqualsIfExists": {
            "aws:RequestTag/access-project": "${aws:PrincipalTag/access-project}",
            "aws:RequestTag/access-team": "${aws:PrincipalTag/access-team}",
            "aws:RequestTag/cost-center": "${aws:PrincipalTag/cost-center}"
        }
    }
},
{
    "Sid": "AllResourcesSecretsManagerNoTags",
    "Effect": "Allow",
    "Action": [
        "secretsmanager:GetRandomPassword",
        "secretsmanager>ListSecrets"
    ],
    "Resource": "*"
},
{
    "Sid": "ReadSecretsManagerSameTeam",
    "Effect": "Allow",
    "Action": [
        "secretsmanager:Describe*",
        "secretsmanager:Get*",
        "secretsmanager>List*"
    ],
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "aws:ResourceTag/access-team": "${aws:PrincipalTag/access-team}"
        }
    }
},
{
    "Sid": "DenyUntagSecretsManagerReservedTags",
    "Effect": "Deny",
    "Action": "secretsmanager:UntagResource",
    "Resource": "*",
    "Condition": {
        "StringLike": {
            "aws:TagKeys": "access-*"
        }
    }
},
{
    "Sid": "DenyPermissionsManagement",
    "Effect": "Deny",
    "Action": "secretsmanager:*Policy",
}
```

```
        "Resource": "*"
    ]
}
```

What does this policy do?

- The `AllActionsSecretsManagerSameProjectSameTeam` statement allows all of this service's actions on all related resources, but only if the resource tags match the principal tags. By adding `"Action": "secretsmanager:/*"` to the policy, the policy grows as Secrets Manager grows. If Secrets Manager adds a new API operation, you are not required to add that action to the statement. The statement implements ABAC using three condition blocks. The request is allowed only if all three blocks return true.
 - The first condition block of this statement returns true if the specified tag keys are present on the resource, and their values match the principal's tags. This block returns false for mismatched tags, or for actions that don't support resource tagging. To learn which actions are not allowed by this block, see [Actions, Resources, and Condition Keys for AWS Secrets Manager](#). That page shows that actions performed on the `Secret` resource type support the `secretsmanager:ResourceTag/tag-key` condition key. Some [Secrets Manager actions](#) don't support that resource type, including `GetRandomPassword` and `ListSecrets`. You must create additional statements to allow those actions.
 - The second condition block returns true if every tag key passed in the request is included in the specified list. This is done using `ForAllValues` with the `StringEquals` condition operator. If no keys or a subset of the set of keys are passed, then the condition returns true. This allows `Get*` operations that do not allow passing tags in the request. If the requester includes a tag key that is not in the list, the condition returns false. Every tag key that is passed in the request must match a member of this list. For more information, see [Using multiple keys and values \(p. 653\)](#).
 - The third condition block returns true if the request supports passing tags, if all three of the tags are present, and if they match the principal tag values. This block also returns true if the request does not support passing tags. This is thanks to [...IfExists \(p. 650\)](#) in the condition operator. The block returns false if there is no tag passed during an action that supports it, or if the tag keys and values don't match.
- The `AllResourcesSecretsManagerNoTags` statement allows the `GetRandomPassword` and `ListSecrets` actions that are not allowed by the first statement.
- The `ReadSecretsManagerSameTeam` statement allows read-only operations if the principal is tagged with the same access-team tag as the resource. This is allowed regardless of the project or cost-center tag.
- The `DenyUntagSecretsManagerReservedTags` statement denies requests to remove tags with keys that begin with "access-" from Secrets Manager. These tags are used to control access to resources, therefore removing tags might remove permissions.
- The `DenyPermissionsManagement` statement denies access to create, edit, or delete Secrets Manager resource-based policies. These policies could be used to change the permissions of the secret.

Important

This policy uses a strategy to allow all actions for a service, but explicitly deny permissions-altering actions. Denying an action overrides any other policy that allows the principal to perform that action. This can have unintended results. As a best practice, use explicit denies only when there is no circumstance that should allow that action. Otherwise, allow a list of individual actions, and the unwanted actions are denied by default.

Step 3: Create roles

Create the following IAM roles and attach the `access-same-project-team` policy that you created in the previous step. For more information about creating IAM roles, see [Creating a role to delegate](#)

permissions to an IAM user (p. 221). If you choose to use federation instead of IAM users and roles, see [IAM Tutorial: Use SAML session tags for ABAC \(p. 57\)](#).

ABAC roles

Job function	Role tags	Role name	Role description
Project Pegasus Engineering	<code>access-project = peg</code> <code>access-team = eng</code> <code>cost-center = 987654</code>	<code>access-peg-engineering</code>	Allows engineers to read all engineering resources and create and manage Pegasus engineering resources.
Project Pegasus Quality Assurance	<code>access-project = peg</code> <code>access-team = qas</code> <code>cost-center = 987654</code>	<code>access-peg-quality-assurance</code>	Allows the QA team to read all QA resources and create and manage all Pegasus QA resources.
Project Unicorn Engineering	<code>access-project = uni</code> <code>access-team = eng</code> <code>cost-center = 123456</code>	<code>access-uni-engineering</code>	Allows engineers to read all engineering resources and create and manage Unicorn engineering resources.
Project Unicorn Quality Assurance	<code>access-project = uni</code> <code>access-team = qas</code> <code>cost-center = 123456</code>	<code>access-uni-quality-assurance</code>	Allows the QA team to read all QA resources and create and manage all Unicorn QA resources.

Step 4: Test creating secrets

The permissions policy attached to the roles allows the employees to create secrets. This is allowed only if the secret is tagged with their project, team, and cost center. Confirm that your permissions are working as expected by signing in as your users, assuming the correct role, and testing activity in Secrets Manager.

To test creating a secret with and without the required tags

1. In your main browser window, remain signed in as the administrator user so that you can review users, roles, and policies in IAM. Use a browser incognito window or separate browser for your testing. There, sign in as the `access-Arnav-peg-eng` IAM user and open the Secrets Manager console at <https://console.aws.amazon.com/secretsmanager/>.
2. Attempt to switch to the `access-uni-engineering` role. For more information about switching roles in the AWS Management Console, see [Switching to a role \(console\) \(p. 253\)](#).

This operation fails because the `access-project` and `cost-center` tag values do not match for the `access-Arnav-peg-eng` user and `access-uni-engineering` role.

3. Switch to the `access-peg-engineering` role. For more information about switching roles in the AWS Management Console, see [Switching to a role \(console\) \(p. 253\)](#).
4. Store a new secret using the following information. To learn how to store a secret, see [Creating a Basic Secret](#) in the *AWS Secrets Manager User Guide*.
 1. In the **Select secret type** section, choose **Other type of secrets**. In the two text boxes, enter `test-access-key` and `test-access-secret`.

You must have additional permissions to save credentials for specific AWS services. For example, to create credentials for an Amazon RDS database, you must have permission to describe RDS instances, RDS clusters, and Amazon Redshift clusters.

2. Enter `test-access-peg-eng` for the **Secret name** field.
3. Add different tag combinations from the following table and view the expected behavior.
4. Choose **Store** to attempt to create the secret. When the storage fails, return to the previous Secrets Manager console pages and use the next tag set from the following table. The last tag set is allowed and will successfully create the secret.

ABAC tag combinations for `test-access-peg-eng` role

access-project Tag value	access-team Tag value	cost-center Tag value	Additional tags	Expected behavior
(none)	(none)	(none)	(none)	Denied because the access-project tag value does not match the role's value of <code>peg</code> .
uni	eng	987654	(none)	Denied because the access-project tag value does not match the role's value of <code>peg</code> .
peg	qas	987654	(none)	Denied because the access-team tag value does not match the role's value of <code>eng</code> .
peg	eng	123456	(none)	Denied because the cost-center tag value does not match the role's value of <code>987654</code> .
peg	eng	987654	owner = Jane	Denied because the additional tag <code>owner</code> is not allowed by the policy, even though all three required tags are present and their values match the role's values.
peg	eng	987654	Name = Jane	Allowed because all three required tags are present and their values match the role's values. You are also allowed to include the optional <code>Name</code> tag.

5. Sign out and repeat the first three steps of this procedure for each of the following roles and tag values. In the fourth step in this procedure, test any set of missing tags, optional tags, disallowed tags, and invalid tag values that you choose. Then use the required tags to create a secret with the following tags and name.

ABAC roles and tags

User name	Role name	Secret name	Secret tags
access-Mary-peg-qas	access-peg-quality-assurance	test-access-peg-qas	access-project = peg access-team = qas

User name	Role name	Secret name	Secret tags
			cost-center = 987654
access-Saanvi-uni-eng	access-uni-engineering	test-access-uni-eng	access-project = uni access-team = eng cost-center = 123456
access-Carlos-uni-qas	access-uni-quality-assurance	test-access-uni-qas	access-project = uni access-team = qas cost-center = 123456

Step 5: Test viewing secrets

The policy that you attached to each role allows the employees to view any secrets tagged with their team name, regardless of their project. Confirm that your permissions are working as expected by testing your roles in Secrets Manager.

To test viewing a secret with and without the required tags

1. Sign in as one of the following IAM users:

- access-Arnav-peg-eng
- access-Mary-peg-qas
- access-Saanvi-uni-eng
- access-Carlos-peg-qas

2. Switch to the matching role:

- access-peg-engineering
- access-peg-quality-assurance
- access-uni-engineering
- access-uni-quality-assurance

For more information about switching roles in the AWS Management Console, see [Switching to a role \(console\) \(p. 253\)](#).

3. In the navigation pane on the left, choose the menu icon to expand the menu and then choose **Secrets**.
4. You should see all four secrets in the table, regardless of your current role. This is expected because the policy named `access-same-project-team` allows the `secretsmanager>ListSecrets` action for all resources.
5. Choose the name of one of the secrets.
6. On the details page for the secret, your role's tags determine whether you can view the page content. Compare the name of your role to the name of your secret. If they share the same team name, then the `access-team` tags match. If they don't match, then access is denied.

ABAC secret viewing behavior for each role

Role name	Secret name	Expected behavior
access-peg-engineering	test-access-peg-eng	Allowed
	test-access-peg-qas	Denied
	test-access-uni-eng	Allowed
	test-access-uni-qas	Denied
access-peg-quality-assurance	test-access-peg-eng	Denied
	test-access-peg-qas	Allowed
	test-access-uni-eng	Denied
	test-access-uni-qas	Allowed
access-uni-engineering	test-access-peg-eng	Allowed
	test-access-peg-qas	Denied
	test-access-uni-eng	Allowed
	test-access-uni-qas	Denied
access-uni-quality-assurance	test-access-peg-eng	Denied
	test-access-peg-qas	Allowed
	test-access-uni-eng	Denied
	test-access-uni-qas	Allowed

- From the breadcrumbs at the top of the page, choose **Secrets** to return to the list of secrets. Repeat the steps in this procedure using different roles to test whether you can view each of the secrets.

Step 6: Test scalability

An important reason for using attribute-based access control (ABAC) over role-based access control (RBAC) is scalability. As your company adds new projects, teams, or people to AWS, you don't need to update your ABAC-driven policies. For example, assume that Example Company is funding a new project, code named **Centaurs**. An engineer named Saanvi Sarkar will be the lead engineer for **Centaurs** while continuing to work on the **Unicorn** project. There are also several newly hired engineers, including Nikhil Jayashankar, who will work on only the **Centaurs** project.

To add the new project to AWS

- Sign in as the IAM administrator user and open the IAM console at <https://console.aws.amazon.com/iam/>.
- In the navigation pane on the left, choose **Roles** and add an IAM role named **access-cen-engineering**. Attach the **access-same-project-team** permissions policy to the role and add the following tags:
 - access-project = cen**
 - access-team = eng**

- cost-center = 101010
3. In the navigation pane on the left, choose **Users**.
 4. Add a new user named access-Nikhil-cen-eng, and attach the policy named access-assume-role.
 5. Use the procedures in [Step 4: Test creating secrets \(p. 51\)](#) and [Step 5: Test viewing secrets \(p. 53\)](#). In another browser window, test that Nikhil can create only **Centaur** engineering secrets, and that he can view all engineering secrets.
 6. In the main browser window where you signed in as the administrator, choose access-Saanvi-uni-eng.
 7. On the **Permissions** tab, remove the **access-assume-role** permissions policy.
 8. Add the following inline policy named **access-assume-specific-roles**. For more information about adding an inline policy to a user, see [To embed an inline policy for a user or role \(console\) \(p. 456\)](#).

ABAC policy: Assume only specific roles

This policy allows Saanvi to assume the engineering roles for the Pegasus or **Centaur** projects. It is necessary to create this custom policy because IAM does not support multivalued tags. You can't tag Saanvi's user with `access-project = peg` and `access-project = cen`. Additionally, the AWS authorization model can't match both values. For more information, see [Rules for tagging in IAM and AWS STS \(p. 289\)](#). Instead, you must manually specify the two roles that she can assume.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "TutorialAssumeSpecificRoles",  
            "Effect": "Allow",  
            "Action": "sts:AssumeRole",  
            "Resource": [  
                "arn:aws:iam::123456789012:role/access-peg-engineering",  
                "arn:aws:iam::123456789012:role/access-cen-engineering"  
            ]  
        }  
    ]  
}
```

9. Use the procedures in [Step 4: Test creating secrets \(p. 51\)](#) and [Step 5: Test viewing secrets \(p. 53\)](#). In another browser window, confirm that Saanvi can assume both roles. Check that she can create secrets for only her project, team, and cost center, depending on the role's tags. Also confirm that she can view details about any secrets owned by the engineering team, including the ones that she just created.

Step 7: Test updating and deleting secrets

The `access-same-project-team` policy that is attached to the roles allows the employees to update and delete any secrets tagged with their project, team, and cost center. Confirm that your permissions are working as expected by testing your roles in Secrets Manager.

To test updating and deleting a secret with and without the required tags

1. Sign in as one of the following IAM users:
 - `access-Arnav-peg-eng`
 - `access-Mary-peg-qas`
 - `access-Saanvi-uni-eng`

- access-Carlos-peg-qas
 - access-Nikhil-cen-eng
2. Switch to the matching role:
- access-peg-engineering
 - access-peg-quality-assurance
 - access-uni-engineering
 - access-peg-quality-assurance
 - access-cen-engineering

For more information about switching roles in the AWS Management Console, see [Switching to a role \(console\) \(p. 253\)](#).

3. For each role, try to update the secret description and then try to delete the following secrets. For more information, see [Modifying a Secret](#) and [Deleting and Restoring a Secret](#) in the *AWS Secrets Manager User Guide*.

ABAC secret updating and deleting behavior for each role

Role name	Secret name	Expected behavior
access-peg-engineering	test-access-peg-eng	Allowed
	test-access-uni-eng	Denied
	test-access-uni-qas	Denied
access-peg-quality-assurance	test-access-peg-qas	Allowed
	test-access-uni-eng	Denied
access-uni-engineering	test-access-uni-eng	Allowed
	test-access-uni-qas	Denied
access-peg-quality-assurance	test-access-uni-qas	Allowed

Summary

You've now successfully completed all of the steps necessary to use tags for attribute-based access control (ABAC). You've learned how to define a tagging strategy. You applied that strategy to your principals and resources. You created and applied a policy that enforces the strategy for Secrets Manager. You also learned that ABAC scales easily when you add new projects and team members. As a result, you are able to sign in to the IAM console with your test roles and experience how to use tags for ABAC in AWS.

Note

You added policies that allow actions only under specific conditions. If you apply a different policy to your users or roles that has broader permissions, then the actions might not be limited to require tagging. For example, if you give a user full administrative permissions using the `AdministratorAccess` AWS managed policy, then these policies don't restrict that access. For more information about how permissions are determined when multiple policies are involved, see [Determining whether a request is allowed or denied within an account \(p. 669\)](#).

Related resources

For related information in the *IAM User Guide*, see the following resources:

- [What is ABAC for AWS? \(p. 13\)](#)
- [AWS global condition context keys \(p. 692\)](#)
- [Creating your first IAM admin user and group \(p. 20\)](#)
- [Creating IAM users \(console\) \(p. 77\)](#)
- [Creating a role to delegate permissions to an IAM user \(p. 221\)](#)
- [Tagging IAM users and roles \(p. 289\)](#)
- [Controlling access to AWS resources using resource tags \(p. 387\)](#)
- [Switching to a role \(console\) \(p. 253\)](#)
- [IAM Tutorial: Use SAML session tags for ABAC \(p. 57\)](#)

To learn how to monitor the tags in your account, see [Monitor tag changes on AWS resources with serverless workflows and Amazon CloudWatch Events](#).

IAM Tutorial: Use SAML session tags for ABAC

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called tags. You can [attach tags to IAM entities \(p. 289\)](#) (users or roles) and to AWS resources. When the entities are used to make requests to AWS, they become principals and those principals include tags.

You can also pass [session tags \(p. 293\)](#) when you assume a role or federate a user. You can then define policies that use tag condition keys to grant permissions to your principals based on their tags. When you use tags to control access to your AWS resources, you allow your teams and resources to grow with fewer changes to AWS policies. ABAC policies are more flexible than traditional AWS policies, which require you to list each individual resource. For more information about ABAC and its advantage over traditional policies, see [What is ABAC for AWS? \(p. 13\)](#).

If your company uses a SAML-based identity provider (IdP) to manage corporate user identities, you can use SAML attributes for fine-grained access control in AWS. Attributes can include cost center identifiers, user email addresses, department classifications, and project assignments. When you pass these attributes as session tags, you can then control access to AWS based on these session tags.

To complete the [ABAC tutorial \(p. 45\)](#) by passing SAML attributes to your session principal, complete the tasks in [IAM Tutorial: Define permissions to access AWS resources based on tags \(p. 45\)](#), with the changes that are included in this topic.

Prerequisites

To perform the steps to use SAML session tags for ABAC, you must already have the following:

- Access to a SAML-based IdP where you can create test users with specific attributes.
- An AWS account that you can sign in to as an IAM user with administrative permissions. If you have a new account and sign in as the AWS account root user, then [create an IAM admin user \(p. 20\)](#).
- Experience creating and editing IAM users, roles, and policies in the AWS Management Console. However, if you need help remembering an IAM management process, the ABAC tutorial provides links where you can view step-by-step instructions.
- Experience setting up a SAML-based IdP in IAM. To view more details and links to detailed IAM documentation, see [Passing session tags using AssumeRoleWithSAML \(p. 297\)](#).

Step 1: Create a test IAM user

Skip the instructions in [Step 1: Create test users \(p. 47\)](#). Because your identities are defined in your provider, it's not necessary for you to add IAM users for your employees.

Step 2: Create the ABAC policy

Follow the instructions in [Step 2: Create the ABAC policy \(p. 48\)](#) to create the specified managed policy in IAM.

Step 3: Create and configure the SAML role

When you use the ABAC tutorial for SAML, you must perform additional steps to create the role, configure the SAML IdP, and enable AWS Management Console access. For more information, see [Step 3: Create roles \(p. 50\)](#).

Step 3A: Create the SAML role

Create a single role that trusts your SAML identity provider and the `test-session-tags` user that you created in step 1. The ABAC tutorial uses separate roles with different role tags. Because you are passing session tags from your SAML IdP, you need only one role. To learn how to create a SAML-based role, see [Creating a role for SAML 2.0 federation \(console\) \(p. 241\)](#).

Name the role `access-session-tags`. Attach the `access-same-project-team` permissions policy to the role. Edit the role trust policy to use the following policy. For detailed instructions on how to edit the trust relationship of a role, see [Modifying a role \(console\) \(p. 273\)](#).

The following role trust policy allows your SAML identity provider and the `test-session-tags` user to assume the role. When they assume the role, they must pass the three specified session tags. The `sts:TagSession` action is required to allow passing session tags.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowSamlIdentityAssumeRole",  
            "Effect": "Allow",  
            "Action": [  
                "sts:AssumeRoleWithSAML",  
                "sts:TagSession"  
            ],  
            "Principal": {"Federated": "arn:aws:iam::123456789012:saml-provider/ExampleCorpProvider"},  
            "Condition": {  
                "StringLike": {  
                    "aws:RequestTag/cost-center": "*",  
                    "aws:RequestTag/access-project": "*",  
                    "aws:RequestTag/access-team": [  
                        "eng",  
                        "qas"  
                    ]  
                },  
                "StringEquals": {"SAML:aud": "https://signin.aws.amazon.com/saml"}  
            }  
        }  
    ]  
}
```

The `AllowSamlIdentityAssumeRole` statement allows members of the Engineering and Quality Assurance teams to assume this role when they federate into AWS from the Example Corporation IdP.

The ExampleCorpProvider SAML provider is defined in IAM. The administrator has already set up the SAML assertion to pass the three required session tags. The assertion can pass additional tags, but these three must be present. The identity's attributes can have any value for the `cost-center` and `access-project` tags. However, the `access-team` attribute value must match `eng` or `qas` to indicate that the identity is on the Engineering or Quality Assurance team.

Step 3B: Configure the SAML IdP

Configure your SAML IdP to pass the `cost-center`, `access-project`, and `access-team` attributes as session tags. For more information, see [Passing session tags using AssumeRoleWithSAML \(p. 297\)](#).

To pass these attributes as session tags, include the following elements in your SAML assertion.

```
<Attribute Name="https://aws.amazon.com/SAML/Attributes/PrincipalTag:cost-center">
  <AttributeValue>987654</AttributeValue>
</Attribute>
<Attribute Name="https://aws.amazon.com/SAML/Attributes/PrincipalTag:access-project">
  <AttributeValue>peg</AttributeValue>
</Attribute>
<Attribute Name="https://aws.amazon.com/SAML/Attributes/PrincipalTag:access-team">
  <AttributeValue>eng</AttributeValue>
</Attribute>
```

Step 3B: Enable console access

Enable console access for your federated SAML users. For more information, see [Enabling SAML 2.0 federated users to access the AWS Management Console \(p. 203\)](#).

Step 4: Test creating secrets

Federate into the AWS Management Console using the `access-session-tags` role. For more information, see [Enabling SAML 2.0 federated users to access the AWS Management Console \(p. 203\)](#). Then follow the instructions in [Step 4: Test creating secrets \(p. 51\)](#) to create secrets. Use different SAML identities with attributes to match the tags that are indicated in the ABAC tutorial. For more information, see [Step 4: Test creating secrets \(p. 51\)](#).

Step 5: Test viewing secrets

Follow the instructions in [Step 5: Test viewing secrets \(p. 53\)](#) to view the secrets that you created in the previous step. Use different SAML identities with attributes to match the tags that are indicated in the ABAC tutorial.

Step 6: Test scalability

Follow the instructions in [Step 6: Test scalability \(p. 54\)](#) to test scalability. Do this by adding a new identity in your SAML-based IdP with the following attributes:

- `cost-center` = 101010
- `access-project` = cen
- `access-team` = eng

Step 7: Test updating and deleting secrets

Follow the instructions in [Step 7: Test updating and deleting secrets \(p. 55\)](#) to update and delete secrets. Use different SAML identities with attributes to match the tags that are indicated in the ABAC tutorial.

Important

Delete all of the secrets that you created to avoid billing charges. For details about pricing in Secrets Manager, see [AWS Secrets Manager Pricing](#).

Summary

You've now successfully completed all of the steps necessary to use SAML session tags and resource tags for permissions management.

Note

You added policies that allow actions only under specific conditions. If you apply a different policy to your users or roles that has broader permissions, then the actions might not be limited to require tagging. For example, if you give a user full administrative permissions using the AdministratorAccess AWS managed policy, then these policies don't restrict that access. For more information about how permissions are determined when multiple policies are involved, see [Determining whether a request is allowed or denied within an account \(p. 669\)](#).

IAM Tutorial: Enable users to manage their credentials and MFA settings

You can enable your users to manage their own multi-factor authentication (MFA) devices and credentials on the **My Security Credentials** page. You can use the AWS Management Console to configure credentials (access keys, passwords, signing certificates, and SSH public keys) and MFA devices for your users. This is useful for a small number of users. But that task could quickly become time consuming as the number of users grows. Security best practices specify that users should regularly change their passwords and rotate their access keys. They should also delete or deactivate credentials that are not needed. We also highly recommend that they use MFA for sensitive operations. This tutorial shows you how to enable these best practices without burdening your administrators.

This tutorial shows how to allow users to access AWS services, but **only** when they sign in with MFA. If they are not signed in with an MFA device, then users cannot access other services.

This workflow has three basic steps.

[Step 1: Create a policy to enforce MFA sign-in \(p. 61\)](#)

Create a customer managed policy that prohibits all actions **except** the few IAM actions that allow a user to change their own credentials and manage their MFA devices on the **My Security Credentials** page. For more information about accessing that page, see [How IAM users change their own password \(console\) \(p. 101\)](#).

[Step 2: Attach policies to your test group \(p. 61\)](#)

Create a group whose members have full access to all Amazon EC2 actions if they sign in with MFA. To create such a group, you attach both the AWS managed policy called `AmazonEC2FullAccess` and the customer managed policy you created in the first step.

[Step 3: Test your user's access \(p. 62\)](#)

Sign in as the test user to verify that access to Amazon EC2 is blocked *until* the user creates an MFA device. The user can then sign in using that device.

Prerequisites

To perform the steps in this tutorial, you must already have the following:

- An AWS account that you can sign in to as an IAM user with administrative permissions.
- Your account ID number, which you type into the policy in Step 1.

To find your account ID number, on the navigation bar at the top of the page, choose **Support** and then choose **Support Center**. You can find your account ID under this page's **Support** menu.

- A [virtual \(software-based\) MFA device \(p. 114\)](#), [U2F security key \(p. 117\)](#), or [hardware-based MFA device \(p. 122\)](#).
- A test IAM user who is a member of a group as follows:

Create user account		Create and configure group account		
MFAUser	Choose only the option for AWS Management Console access , and assign a password.	EC2MFA	MFAUser	Do NOT attach any policies or otherwise grant permissions to this group.

Step 1: Create a policy to enforce MFA sign-in

You begin by creating an IAM customer managed policy that denies all permissions except those required for IAM users to manage their own credentials and MFA devices.

1. Sign in to the AWS Management Console as a user with administrator credentials. To adhere to IAM best practices, don't sign in with your AWS account root user credentials. For more information, see [Create individual IAM users](#).
2. Open the IAM console at <https://console.aws.amazon.com/iam/>.
3. In the navigation pane, choose **Policies**, and then choose **Create policy**.
4. Choose the **JSON** tab and copy the text from the following JSON policy document: [AWS: Allows MFA-authenticated IAM users to manage their own credentials on the My Security Credentials page \(p. 393\)](#).
5. Paste the policy text into the **JSON** text box, then choose **Review policy**. The [Policy Validator \(p. 444\)](#) reports any syntax errors.

Note

You can switch between the **Visual editor** and **JSON** tabs anytime. However, the policy above includes the `NotAction` element, which is not supported in the visual editor. For this policy, you will see a notification on the **Visual editor** tab. Return to the **JSON** tab to continue working with this policy.

6. On the **Review** page, type `Force_MFA` for the policy name. For the policy description, type `This policy allows users to manage their own passwords and MFA devices but nothing else unless they authenticate with MFA.` Review the policy **Summary** to see the permissions granted by your policy, and then choose **Create policy** to save your work.

The new policy appears in the list of managed policies and is ready to attach.

Step 2: Attach policies to your test group

Next you attach two policies to the test IAM group, which will be used to grant the MFA-protected permissions.

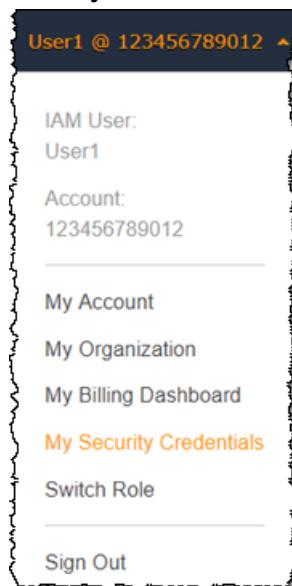
1. In the navigation pane, choose **Groups**.
2. In the search box, type `EC2MFA`, and then choose the group name (not the check box) in the list.

3. On the **Permissions** tab, and click **Attach Policy**.
4. On the **Attach Policy** page, in the search box, type **EC2Full** and then select the check box next to **AmazonEC2FullAccess** in the list. Don't save your changes yet.
5. In the search box, type **Force**, and then select the check box next to **Force_MFA** in the list.
6. Choose **Attach Policy**.

Step 3: Test your user's access

In this part of the tutorial, you sign in as the test user and verify that the policy works as intended.

1. Sign in to your AWS account as **MFAUser** with the password you assigned in the previous section. Use the URL: <https://<alias or account ID number>.signin.aws.amazon.com/console>
2. Choose **EC2** to open the Amazon EC2 console and verify that the user has no permissions to do anything.
3. In the navigation bar on the upper right, choose the **MFAUser** user name, and then choose **My Security Credentials**.



4. Now add an MFA device. In the **Multi-factor Authentication (MFA)** section, choose **Assign MFA device**.

Note

You might receive an error that you are not authorized to perform `iam>DeleteVirtualMFADevice`. This could happen if someone previously began assigning a virtual MFA device to this user and cancelled the process. To continue, you or another administrator must delete the user's existing MFA device. For more information, see [I am not authorized to perform: iam>DeleteVirtualMFADevice \(p. 569\)](#).

5. For this tutorial, we use a virtual (software-based) MFA device, such as the Google Authenticator app on a mobile phone. Choose **Virtual MFA device**, and then click **Continue**.

IAM generates and displays configuration information for the virtual MFA device, including a QR code graphic. The graphic is a representation of the secret configuration key that is available for manual entry on devices that do not support QR codes.

6. Open your virtual MFA app. (For a list of apps that you can use for hosting virtual MFA devices, see [Virtual MFA Applications](#).) If the virtual MFA app supports multiple accounts (multiple virtual MFA devices), choose the option to create a new account (a new virtual MFA device).

7. Determine whether the MFA app supports QR codes, and then do one of the following:
 - From the wizard, choose **Show QR code**. Then use the app to scan the QR code. For example, you might choose the camera icon or choose an option similar to **Scan code**, and then use the device's camera to scan the code.
 - In the **Manage MFA Device** wizard, choose **Show secret key**, and then type the secret key into your MFA app.

When you are finished, the virtual MFA device starts generating one-time passwords.

8. In the **Manage MFA Device** wizard, in the **MFA Code 1** box, type the one-time password that currently appears in the virtual MFA device. Wait up to 30 seconds for the device to generate a new one-time password. Then type the second one-time password into the **MFA Code 2** box. Choose **Assign MFA**.

Important

Submit your request immediately after generating the codes. If you generate the codes and then wait too long to submit the request, the MFA device is successfully associated with the user. However, the MFA device is out of sync. This happens because time-based one-time passwords (TOTP) expire after a short period of time. If this happens, you can [resync the device \(p. 130\)](#).

The virtual MFA device is now ready to use with AWS.

9. Sign out of the console and then sign in as **MFAUser** again. This time AWS prompts you for an MFA code from your phone. When you get it, type the code in the box and then choose **Submit**.
10. Choose **EC2** to open the Amazon EC2 console again. Note that this time you can see all the information and perform any actions you want. If you go to any other console as this user, you see access denied messages. The reason is that the policies in this tutorial grant access only to Amazon EC2.

Related resources

For related information found in the *IAM User Guide*, see the following resources:

- [Using multi-factor authentication \(MFA\) in AWS \(p. 111\)](#)
- [Enabling MFA devices for users in AWS \(p. 112\)](#)
- [Using MFA devices with your IAM sign-in page \(p. 83\)](#)

Signing in to the AWS Management Console as an IAM user or root user

The AWS Management Console provides a web-based user interface that you can use to create and manage your AWS resources. For example, you can start and stop Amazon EC2 instances, create Amazon DynamoDB tables, create Amazon S3 buckets, and so on.

Before you can use the AWS Management Console, you must sign in to your AWS account. The process that you will use to sign in to your AWS account depends on what type of AWS user you are. There are two different types of users in AWS. You are either the account owner (root user) or you are an IAM user. The root user is created when the AWS account is created using the email address and password that were used to create the account. IAM users are created by the root user or an IAM administrator within the AWS account.

If you do not remember your credentials or have trouble signing in using your credentials, see [AWS sign-in issues \(p. 69\)](#).

Contents

- [Sign in as the root user \(p. 64\)](#)
- [Sign in as an IAM user \(p. 65\)](#)
- [Your AWS account ID and its alias \(p. 67\)](#)
- [Troubleshooting AWS sign-in or account issues \(p. 69\)](#)

Sign in as the root user

Before you sign in to an AWS account as the root user, be sure that you have the following required information.

Requirements

- The email address used to create the AWS account.
- The password for the root user.

To sign in to an AWS account as the root user

1. Open <https://console.aws.amazon.com/>.
2. If you have not signed in previously using this browser, the main sign-in page appears as follows. Choose **Root user**, enter the email address associated with your account, and choose **Next**.

Sign in

The screenshot shows the first step of the AWS sign-in process. It features two options: 'Root user' (selected) and 'IAM user'. Below each option is a brief description and a 'Learn more' link. A 'Root user email address' input field contains 'username@example.com', and a blue 'Next' button is at the bottom.

Root user
Account owner that performs tasks requiring unrestricted access. [Learn more](#)

IAM user
User within an account that performs daily tasks. [Learn more](#)

Root user email address

username@example.com

Next

If you have signed in as a root user previously using this browser, your browser might remember the email address for the AWS account. If so, you'll see the screen shown in the next step instead.

If you have signed in previously as an IAM user using this browser, your browser might display the IAM user sign in page instead. To return to the main sign-page, choose **Sign in using root user email**.

3. Enter your password and choose **Sign in**.

The screenshot shows the second step of the AWS sign-in process for a root user. It includes fields for 'Email' (user@example.com), 'Password' (input field), and 'Forgot password?'. A large blue 'Sign in' button is prominent. Below the form are links for 'Sign in to a different account' and 'Create a new AWS account'.

Root user sign in

Email: user@example.com

Password [Forgot password?](#)

Sign in

[Sign in to a different account](#)

[Create a new AWS account](#)

Sign in as an IAM user

Before you sign into an AWS account as an IAM user, be sure that you have the following required information. If you do not have this information, contact the administrator for the AWS account.

Requirements

- One of the following:
 - The account alias.
 - The 12-digit AWS account ID.
 - The user name for your IAM user.
 - The password for your IAM user.

If you are a root user or IAM administrator and need to provide the AWS account ID or AWS account alias to an IAM user, see [Your AWS account ID and its alias \(p. 67\)](#).

If you are an IAM user, you can log in using either a sign-in URL or the main sign-in page.

To sign in to an AWS account as an IAM user using an IAM users sign-in URL

1. Open a browser and enter the following sign-in URL, replacing `account_alias_or_id` with the account alias or account ID provided by your administrator.

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

2. Enter your IAM user name and password and choose **Sign in**.

Sign in as IAM user

Account ID (12 digits) or account alias

IAM user name

Password

Sign in

[Sign in using root user email](#)

[Forgot password?](#)

To sign in to an AWS account as an IAM user using the main sign-in page

1. Open <https://console.aws.amazon.com/>.
2. If you have not signed in previously using this browser, the main sign-in page appears. Choose **IAM user**, enter the account alias or account ID, and choose **Next**.

Sign in

Root user

Account owner that performs tasks requiring unrestricted access. [Learn more](#)

IAM user

User within an account that performs daily tasks. [Learn more](#)

Account ID (12 digits) or account alias

Next

If you have signed in as an IAM user previously using this browser, your browser might remember the account alias or account ID for the AWS account. If so, you'll see the screen shown in the next step instead.

3. Enter your IAM user name and password and choose **Sign in**.

Sign in as IAM user

Account ID (12 digits) or account alias

IAM user name

Password

Sign in

[Sign in using root user email](#)

[Forgot password?](#)

If you have signed in as an IAM user for a different AWS account previously using this browser, or you need to sign in as a root user instead, choose **Sign in using root user email** to return to the main sign-in page.

Your AWS account ID and its alias

To sign in to an AWS account as an IAM user, you must have an account alias or an account ID for the AWS account. If you are signed in to the AWS Management Console or have configured the AWS CLI or an AWS SDK with your account credentials, you can find the account alias or account ID for the AWS account. If you cannot sign in, ask your administrator for the information that you need to sign in.

Topics

- [Finding your AWS account ID \(p. 67\)](#)
- [About account aliases \(p. 68\)](#)
- [Creating, deleting, and listing an AWS account alias \(p. 68\)](#)

Finding your AWS account ID

You can find the account ID for your AWS account using the following methods.

Finding Your Account ID using the console

In the navigation bar, choose **Support**, and then **Support Center**. Your currently signed-in 12-digit account number (ID) appears in the **Support Center** navigation pane.

Finding Your Account ID using the AWS CLI

Use the following command to view your user ID, account ID, and your user ARN:

- [aws sts get-caller-identity](#)

Finding Your Account ID using the API

Use the following API to view your user ID, account ID, and your user ARN:

- [GetCallerIdentity](#)

About account aliases

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

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

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

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

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

The original URL containing your AWS account ID remains active and can be used after you create your AWS account alias.

Tip

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

Creating, deleting, and listing an AWS account alias

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

Considerations

- Your AWS account can have only one alias. If you create a new alias for your AWS account, the new alias overwrites the previous alias, and the URL containing the previous alias stops working.
- The account alias must be unique across all Amazon Web Services products. It must contain only digits, lowercase letters, and hyphens. For more information on limitations on AWS account entities, see [IAM and STS quotas \(p. 606\)](#).

Creating, editing, and deleting aliases (console)

You can create, edit, and delete an account alias from the AWS Management Console.

To create, edit, or remove an account alias (console)

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, choose **Dashboard**.

3. Find the **Sign-in URL for IAM users in this account**, and choose **Customize** to the right of the link. If an alias already exists, then choose **Edit** to the right of the link.
4. Type the name you want to use for your alias, then choose **Create alias**. If an alias already exists, then choose **Save**.
5. To remove the alias, choose **Delete alias**, and then choose **Delete**. The sign-in URL reverts to using your AWS account ID.

Creating, deleting, and listing aliases (AWS CLI)

To create an alias for your AWS Management Console sign-in page URL, run the following command:

- `aws iam create-account-alias`

To delete an AWS account ID alias, run the following command:

- `aws iam delete-account-alias`

To display your AWS account ID alias, run the following command:

- `aws iam list-account-aliases`

Creating, deleting, and listing aliases (AWS API)

To create an alias for your AWS Management Console sign-in page URL, call the following operation:

- `CreateAccountAlias`

To delete an AWS account ID alias, call the following operation:

- `DeleteAccountAlias`

To display your AWS account ID alias, call the following operation:

- `ListAccountAliases`

Troubleshooting AWS sign-in or account issues

Use the information here to help you troubleshoot sign-in and other AWS account issues. For step-by-step directions to sign in to an AWS account, see [Signing in to the AWS Management Console as an IAM user or root user \(p. 64\)](#).

If you are having trouble signing in to Amazon.com, see [Amazon Customer Service instead](#).

Issues

- [I need my AWS account ID or AWS account alias \(p. 70\)](#)
- [I forgot my IAM user name or password \(p. 70\)](#)
- [I forgot the root user password for my AWS account \(p. 70\)](#)
- [I don't have access to the email for my AWS account \(p. 70\)](#)
- [I need to change the credit card for my AWS account \(p. 70\)](#)
- [I need to report fraudulent AWS account activity \(p. 70\)](#)

- [I need to close my AWS account \(p. 71\)](#)

I need my AWS account ID or AWS account alias

If you are an IAM user and you are not signed in, you must ask your administrator for the AWS account ID or AWS account alias. You need this information, plus your IAM user name and password, to sign in to an AWS account.

I forgot my IAM user name or password

If you are an IAM user, your administrator provides your credentials. If you forget your password, you must ask your administrator to reset it.

For security purposes, AWS doesn't have access to view, provide, or change your credentials.

I forgot the root user password for my AWS account

If you are a root user and you have lost or forgot the password for your AWS account, you can reset your password. You must know the email address used to create the AWS account and you must have access to the email account. For more information, see [Resetting lost or forgotten passwords or access keys for AWS \(p. 110\)](#).

I don't have access to the email for my AWS account

When you create an AWS account, you provide an email address and password. These are the credentials for the AWS account root user. If you are not sure of the email address associated with your AWS account, check for saved correspondence from no-reply@amazon.com to any email address for your organization that might have been used to open the AWS account.

If you know the email address but no longer have access to the email, first try to recover access to the email using one of the following options:

- If you own the domain for email address, you can restore a deleted email address. Alternatively, you can set up a catch-all for your email account, which "catches all" messages sent to email addresses that no longer exist in the mail server and redirects them to another email address.
- If the email address on the account is part of your corporate email system, we recommend that you contact your IT system administrators. They might be able to help you regain access to the email.

If you're still not able to sign in to your AWS account, you can find alternate support options at [Contact us](#). Expand **I cannot login to my AWS account** and choose **Request Support for AWS Account Credentials**. Provide the information in the form and choose **Submit**.

I need to change the credit card for my AWS account

To change the credit card for your AWS account, you must be able to sign in. AWS has protections in place that require you to prove that you're the account owner. For directions, see [Managing your credit card payment methods](#) in the *AWS Billing and Cost Management User Guide*.

I need to report fraudulent AWS account activity

If you suspect fraudulent activity using your AWS account and would like to make a report, see [How do I report abuse of AWS resources](#).

If you are having trouble with a purchase made on Amazon.com, see [Amazon Customer Service](#).

I need to close my AWS account

If you have an AWS account, you can use the following directions to close it: [Closing an Account](#) in the *Billing and Cost Management User Guide*.

IAM Identities (users, groups, and roles)

Having trouble signing in to AWS? Make sure that you're on the correct [AWS sign-in page \(p. 64\)](#) for your type of user. If you are the AWS account root user (account owner), you can sign in to AWS using the credentials that you set up when you created the AWS account. If you are an IAM user, your account administrator can give you the credentials that you can use to sign in to AWS. If you need to request support, do not use the feedback link on this page, as the form is received by the AWS Documentation team, not AWS Support. Instead, on the [Contact Us](#) page, expand **I cannot login to my AWS Account** and choose **Request Support for AWS Account Credentials**.

The AWS account root user or an IAM administrator for the account can create *IAM identities*. An IAM identity provides access to an AWS account. A *group* is a collection of IAM users managed as a unit. An IAM identity represents a user, and can be authenticated and then authorized to perform actions in AWS. Each IAM identity can be associated with one or more *policies*. Policies determine what actions a user, role, or member of a group can perform, on which AWS resources, and under what conditions.

AWS account root user (p. 333)

When you first create an Amazon Web Services (AWS) account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account.

Important

We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the [best practice of using the root user only to create your first IAM user](#). Then securely lock away the root user credentials and use them to perform only a few account and service management tasks. To view the tasks that require you to sign in as the root user, see [AWS Tasks That Require Root User](#).

IAM users (p. 74)

An IAM [user \(p. 74\)](#) is an entity that you create in AWS. The IAM user represents the person or service who uses the IAM user to interact with AWS. A primary use for IAM users is to give people the ability to sign in to the AWS Management Console for interactive tasks and to make programmatic requests to AWS services using the API or CLI. A user in AWS consists of a name, a password to sign into the AWS Management Console, and up to two access keys that can be used with the API or CLI. When you create an IAM user, you grant it permissions by making it a member of a group that has appropriate permission policies attached (recommended), or by directly attaching policies to the user. You can also clone the permissions of an existing IAM user, which automatically makes the new user a member of the same groups and attaches all the same policies.

IAM groups (p. 160)

An IAM [group \(p. 160\)](#) is a collection of IAM users. You can use groups to specify permissions for a collection of users, which can make those permissions easier to manage for those users. For example, you could have a group called *Admins* and give that group the types of permissions that administrators

typically need. Any user in that group automatically has the permissions that are assigned to the group. If a new user joins your organization and should have administrator privileges, you can assign the appropriate permissions by adding the user to that group. Similarly, if a person changes jobs in your organization, instead of editing that user's permissions, you can remove him or her from the old groups and add him or her to the appropriate new groups. A group cannot be identified as a Principal in a resource-based policy. A group is a way to attach policies to multiple users at one time. When you attach an identity-based policy to a group, all of the users in the group receive the permissions from the group. For more information about these policy types, see [Identity-based policies and resource-based policies \(p. 374\)](#).

IAM roles (p. 167)

An IAM [role \(p. 167\)](#) is very similar to a user, in that it is an identity with permission policies that determine what the identity can and cannot do in AWS. However, a role does not have any credentials (password or access keys) associated with it. Instead of being uniquely associated with one person, a role is intended to be assumable by anyone who needs it. An IAM user can assume a role to temporarily take on different permissions for a specific task. A role can be assigned to a [federated user \(p. 176\)](#) who signs in by using an external identity provider instead of IAM. AWS uses details passed by the identity provider to determine which role is mapped to the federated user.

Temporary credentials in IAM (p. 301)

Temporary credentials are primarily used with IAM roles, but there are also other uses. You can request temporary credentials that have a more restricted set of permissions than your standard IAM user. This prevents you from accidentally performing tasks that are not permitted by the more restricted credentials. A benefit of temporary credentials is that they expire automatically after a set period of time. You have control over the duration that the credentials are valid.

When to create an IAM user (instead of a role)

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

- **You created an AWS account and you're the only person who works in your account.**
It's possible to work with AWS using the root user credentials for your AWS account, but we don't recommend it. Instead, we strongly recommend that you create an IAM user for yourself and use the credentials for that user when you work with AWS. For more information, see [Security best practices in IAM \(p. 527\)](#).
- **Other people in your group need to work in your AWS account, and your group is using no other identity mechanism.**

Create IAM users for the individuals who need access to your AWS resources, assign appropriate permissions to each user, and give each user his or her own credentials. We strongly recommend that you never share credentials among multiple users.

When to create an IAM role (instead of a user)

Create an IAM role in the following situations:

You're creating an application that runs on an Amazon Elastic Compute Cloud (Amazon EC2) instance and that application makes requests to AWS.

Don't create an IAM user and pass the user's credentials to the application or embed the credentials in the application. Instead, create an IAM role that you attach to the EC2 instance to give temporary security credentials to applications running on the instance. When an application uses these credentials in AWS, it can perform all of the operations that are allowed by the policies attached to the role. For details, see [Using an IAM role to grant permissions to applications running on Amazon EC2 instances \(p. 263\)](#).

You're creating an app that runs on a mobile phone and that makes requests to AWS.

Don't create an IAM user and distribute the user's access key with the app. Instead, use an identity provider like Login with Amazon, Amazon Cognito, Facebook, or Google to authenticate users and map the users to an IAM role. The app can use the role to get temporary security credentials that have the permissions specified by the policies attached to the role. For more information, see the following:

- [Amazon Cognito Overview](#) in the *AWS Mobile SDK for Android Developer Guide*
- [Amazon Cognito Overview](#) in the *AWS Mobile SDK for iOS Developer Guide*
- [About web identity federation \(p. 177\)](#)

Users in your company are authenticated in your corporate network and want to be able to use AWS without having to sign in again—that is, you want to allow users to federate into AWS.

Don't create IAM users. Configure a federation relationship between your enterprise identity system and AWS. You can do this in two ways:

- If your company's identity system is compatible with SAML 2.0, you can establish trust between your company's identity system and AWS. For more information, see [About SAML 2.0-based federation \(p. 182\)](#).
- Create and use a custom proxy server that translates user identities from the enterprise into IAM roles that provide temporary AWS security credentials. For more information, see [Enabling custom identity broker access to the AWS console \(p. 206\)](#).

IAM users

An AWS Identity and Access Management (IAM) *user* is an entity that you create in AWS to represent the person or application that uses it to interact with AWS. A user in AWS consists of a name and credentials.

An IAM user with administrator permissions is not the same thing as the AWS account root user. For more information about the root user, see [AWS account root user \(p. 333\)](#).

Important

If you found this page because you are looking for information about the Product Advertising API to sell Amazon products on your website, see the [Product Advertising API 5.0 Documentation](#).

How AWS identifies an IAM user

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

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

```
arn:aws:iam::account-ID-without-hyphens:user/Richard
```

- A unique identifier for the user. This ID is returned only when you use the API, Tools for Windows PowerShell, or AWS CLI to create the user; you do not see this ID in the console.

For more information about these identifiers, see [IAM identifiers \(p. 600\)](#).

Users and credentials

You can access AWS in different ways depending on the user credentials:

- [Console password \(p. 92\)](#): A password that the user can type to sign in to interactive sessions such as the AWS Management Console.
- [Access keys \(p. 102\)](#): A combination of an access key ID and a secret access key. You can assign two to a user at a time. These can be used to make programmatic calls to AWS. For example, you might use access keys when using the API for code or at a command prompt when using the AWS CLI or the AWS PowerShell tools.
- [SSH keys for use with CodeCommit \(p. 153\)](#): An SSH public key in the OpenSSH format that can be used to authenticate with CodeCommit.
- [Server certificates \(p. 155\)](#): SSL/TLS certificates that you can use to authenticate with some AWS services. We recommend that you use AWS Certificate Manager (ACM) to provision, manage, and deploy your server certificates. Use IAM only when you must support HTTPS connections in a region that is not supported by ACM. To learn which regions support ACM, see [AWS Certificate Manager endpoints and quotas](#) in the [AWS General Reference](#).

You can choose the credentials that are right for your IAM user. When you use the AWS Management Console to create a user, you must choose to at least include a console password or access keys. By default, a brand new IAM user created using the AWS CLI or AWS API has no credentials of any kind. You must create the type of credentials for an IAM user based on the needs of your user.

Take advantage of the following options to administer passwords, access keys, and MFA devices:

- [Manage passwords for your IAM users \(p. 92\)](#). Create and change the passwords that permit access to the AWS Management Console. Set a password policy to enforce a minimum password complexity. Allow users to change their own passwords.
- [Manage access keys for your IAM users \(p. 102\)](#). Create and update access keys for programmatic access to the resources in your account.
- You can enhance the security of the user's credentials by enabling [multi-factor authentication \(MFA\) \(p. 111\)](#) for the user. With MFA, users have to provide two forms of identification: First, they provide the credentials that are part of their user identity (a password or access key). In addition, they provide a temporary numeric code that's generated on a hardware device or by an application on a smartphone or tablet, or sent by AWS to an SMS-compatible mobile device.
- [Find unused passwords and access keys \(p. 146\)](#). Anyone who has a password or access keys for your account or an IAM user in your account has access to your AWS resources. The security [best practice](#) is to remove passwords and access keys when users no longer need them.
- [Download a credential report for your account \(p. 148\)](#). You can generate and download a credential report that lists all IAM users in your account and the status of their various credentials, including passwords, access keys, and MFA devices. For passwords and access keys, the credential report shows how recently the password or access key has been used.

Users and permissions

By default, a brand new IAM user has no [permissions \(p. 350\)](#) to do anything. The user is not authorized to perform any AWS operations or to access any AWS resources. An advantage of having individual

IAM users is that you can assign permissions individually to each user. You might assign administrative permissions to a few users, who then can administer your AWS resources and can even create and manage other IAM users. In most cases, however, you want to limit a user's permissions to just the tasks (AWS actions or operations) and resources that are needed for the job.

Imagine a user named Diego. When you create the IAM user Diego, you can create a password for that user. You also attach permissions to the IAM user that let him launch a specific Amazon EC2 instance and read (GET) information from a table in an Amazon RDS database. For procedures on how to create users and grant them initial credentials and permissions, see [Creating an IAM user in your AWS account \(p. 76\)](#). For procedures on how to change the permissions for existing users, see [Changing permissions for an IAM user \(p. 87\)](#). For procedures on how to change the user's password or access keys, see [Managing user passwords in AWS \(p. 92\)](#) and [Managing access keys for IAM users \(p. 102\)](#).

You can also add a permissions boundary to your users. A permissions boundary is an advanced feature that allows you to use AWS managed policies to limit the maximum permissions that an identity-based policy can grant to a user or role. For more information about policy types and uses, see [Policies and permissions in IAM \(p. 351\)](#).

Users and accounts

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

The number and size of IAM resources in an AWS account are limited. For more information, see [IAM and STS quotas \(p. 606\)](#).

Users as service accounts

An IAM user is a resource in IAM that has associated credentials and permissions. An IAM user can represent a person or an application that uses its credentials to make AWS requests. This is typically referred to as a *service account*. If you choose to use the long-term credentials of an IAM user in your application, **do not embed access keys directly into your application code**. The AWS SDKs and the AWS Command Line Interface allow you to put access keys in known locations so that you do not have to keep them in code. For more information, see [Manage IAM User Access Keys Properly](#) in the *AWS General Reference*. Alternatively, and as a best practice, you can [use temporary security credentials \(IAM roles\) instead of long-term access keys](#).

Creating an IAM user in your AWS account



Follow us on Twitter

You can create one or more IAM users in your AWS account. You might create an IAM user when someone joins your team, or when you create a new application that needs to make API calls to AWS.

Important

If you found this page because you are looking for information about the Product Advertising API to sell Amazon products on your website, see the [Product Advertising API 5.0 Documentation](#).

If you arrived at this page from the IAM console, it is possible that your account does not include IAM users, even though you are signed in. You could be signed in as the AWS account root user, using a role, or signed in with temporary credentials. To learn more about these IAM identities, see [IAM Identities \(users, groups, and roles\) \(p. 72\)](#).

Topics

- [Creating IAM users \(console\) \(p. 77\)](#)
- [Creating IAM users \(AWS CLI\) \(p. 79\)](#)

- [Creating IAM users \(AWS API\) \(p. 80\)](#)

The process of creating a user and enabling that user to perform work tasks consists of the following steps:

1. Create the user in the AWS Management Console, the AWS CLI, Tools for Windows PowerShell, or using an AWS API operation. If you create the user in the AWS Management Console, then steps 1–4 are handled automatically, based on your choices. If you create the users programmatically, then you must perform each of those steps individually.
2. Create credentials for the user, depending on the type of access the user requires:
 - **Programmatic access:** The IAM user might need to make API calls, use the AWS CLI, or use the Tools for Windows PowerShell. In that case, create an access key (access key ID and a secret access key) for that user.
 - **AWS Management Console access:** If the user needs to access the AWS Management Console, [create a password for the user \(p. 96\)](#).

As a best practice, create only the credentials that the user needs. For example, for a user who requires access only through the AWS Management Console, do not create access keys.

3. Give the user permissions to perform the required tasks by adding the user to one or more groups. You can also grant permissions by attaching permissions policies directly to the user. However, we recommend instead that you put your users in groups and manage permissions through policies that are attached to those groups. You can also use a [permissions boundary \(p. 365\)](#) to limit the permissions that a user can have, though this is not common.
4. (Optional) Add metadata to the user by attaching tags. For more information about using tags in IAM, see [Tagging IAM users and roles \(p. 289\)](#).
5. Provide the user with the necessary sign-in information. This includes the password and the console URL for the account sign-in page where the user provides those credentials. For more information, see [How IAM users sign in to AWS \(p. 81\)](#).
6. (Optional) Configure [multi-factor authentication \(MFA\) \(p. 111\)](#) for the user. MFA requires the user to provide a one-time-use code each time he or she signs into the AWS Management Console.
7. (Optional) Give users permissions to manage their own security credentials. (By default, users do not have permissions to manage their own credentials.) For more information, see [Permitting IAM users to change their own passwords \(p. 99\)](#).

For information about the permissions that you need in order to create a user, see [Permissions required to access IAM resources \(p. 516\)](#).

Creating IAM users (console)

You can use the AWS Management Console to create IAM users.

To create one or more IAM users (console)

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, choose **Users** and then choose **Add user**.
3. Type the user name for the new user. This is the sign-in name for AWS. If you want to add multiple users, choose **Add another user** for each additional user and type their user names. You can add up to 10 users at one time.

Note

The number and size of IAM resources in an AWS account are limited. For more information, see [IAM and STS quotas \(p. 606\)](#). User names can be a combination of up to 64 letters, digits, and these characters: plus (+), equal (=), comma (,), period (.), at sign (@), underscore

(.), and hyphen (-). Names must be unique within an account. They are not distinguished by case. For example, you cannot create two users named *TESTUSER* and *testuser*.

4. Select the type of access this set of users will have. You can select programmatic access, access to the AWS Management Console, or both.
 - Select **Programmatic access** if the users require access to the API, AWS CLI, or Tools for Windows PowerShell. This creates an access key for each new user. You can view or download the access keys when you get to the **Final** page.
 - Select **AWS Management Console access** if the users require access to the AWS Management Console. This creates a password for each new user.
- a. For **Console password**, choose one of the following:
 - **Autogenerated password**. Each user gets a randomly generated password that meets the [account password policy \(p. 93\)](#). You can view or download the passwords when you get to the **Final** page.
 - **Custom password**. Each user is assigned the password that you type in the box.
- b. (Optional) We recommend that you select **Require password reset** to ensure that users are forced to change their password the first time they sign in.

Note

If an administrator has enabled the [Allow users to change their own password account password policy setting](#), then this check box does nothing. Otherwise, it automatically attaches an AWS managed policy named [IAMUserChangePassword](#) to the new users. The policy grants them permission to change their own passwords.

5. Choose **Next: Permissions**.
6. On the **Set permissions** page, specify how you want to assign permissions to this set of new users. Choose one of the following three options:
 - **Add user to group**. Choose this option if you want to assign the users to one or more groups that already have permissions policies. IAM displays a list of the groups in your account, along with their attached policies. You can select one or more existing groups, or choose **Create group** to create a new group. For more information, see [Changing permissions for an IAM user \(p. 87\)](#).
 - **Copy permissions from existing user**. Choose this option to copy all of the group memberships, attached managed policies, embedded inline policies, and any existing [permissions boundaries \(p. 365\)](#) from an existing user to the new users. IAM displays a list of the users in your account. Select the one whose permissions most closely match the needs of your new users.
 - **Attach existing policies to user directly**. Choose this option to see a list of the AWS managed and customer managed policies in your account. Select the policies that you want to attach to the new users or choose **Create policy** to open a new browser tab and create a new policy from scratch. For more information, see step 4 in the procedure [Creating IAM policies \(console\) \(p. 439\)](#). After you create the policy, close that tab and return to your original tab to add the policy to the new user. As a [best practice \(p. 528\)](#), we recommend that you instead attach your policies to a group and then make users members of the appropriate groups.
7. (Optional) Set a [permissions boundary \(p. 365\)](#). This is an advanced feature.

Open the **Set permissions boundary** section and choose **Use a permissions boundary to control the maximum user permissions**. IAM displays a list of the AWS managed and customer managed policies in your account. Select the policy to use for the permissions boundary or choose **Create policy** to open a new browser tab and create a new policy from scratch. For more information, see step 4 in the procedure [Creating IAM policies \(console\) \(p. 439\)](#). After you create the policy, close that tab and return to your original tab to select the policy to use for the permissions boundary.

8. Choose **Next: Tags**.
9. (Optional) Add metadata to the user by attaching tags as key-value pairs. For more information about using tags in IAM, see [Tagging IAM users and roles \(p. 289\)](#).

10. Choose **Next: Review** to see all of the choices you made up to this point. When you are ready to proceed, choose **Create user**.
11. To view the users' access keys (access key IDs and secret access keys), choose **Show** next to each password and access key that you want to see. To save the access keys, choose **Download .csv** and then save the file to a safe location.

Important

This is your only opportunity to view or download the secret access keys, and you must provide this information to your users before they can use the AWS API. Save the user's new access key ID and secret access key in a safe and secure place. **You will not have access to the secret keys again after this step.**

12. Provide each user with his or her credentials. On the final page you can choose **Send email** next to each user. Your local mail client opens with a draft that you can customize and send. The email template includes the following details to each user:

- User name
- URL to the account sign-in page. Use the following example, substituting the correct account ID number or account alias:

```
https://AWS-account-ID or alias.signin.aws.amazon.com/console
```

For more information, see [How IAM users sign in to AWS \(p. 81\)](#).

Important

The user's password is **not** included in the generated email. You must provide them to the customer in a way that complies with your organization's security guidelines.

Creating IAM users (AWS CLI)

You can use the AWS CLI to create an IAM user.

To create an IAM user (AWS CLI)

1. Create a user.
 - [aws iam create-user](#)
2. (Optional) Give the user access to the AWS Management Console. This requires a password. You must also give the user the [URL of your account's sign-in page. \(p. 81\)](#)
 - [aws iam create-login-profile](#)
3. (Optional) Give the user programmatic access. This requires access keys.
 - [aws iam create-access-key](#)
 - Tools for Windows PowerShell: [New-IAMAccessKey](#)
 - IAM API: [CreateAccessKey](#)

Important

This is your only opportunity to view or download the secret access keys, and you must provide this information to your users before they can use the AWS API. Save the user's new access key ID and secret access key in a safe and secure place. **You will not have access to the secret keys again after this step.**

4. Add the user to one or more groups. The groups that you specify should have attached policies that grant the appropriate permissions for the user.
 - [aws iam add-user-to-group](#)

5. (Optional) Attach a policy to the user that defines the user's permissions. **Note:** We recommend that you manage user permissions by adding the user to a group and attaching a policy to the group instead of attaching directly to a user.
 - [aws iam attach-user-policy](#)
6. (Optional) Add custom attributes to the user by attaching tags. For more information, see [Managing tags on IAM entities \(AWS CLI or AWS API\) \(p. 292\)](#).
7. (Optional) Give the user permission to manage their own security credentials. For more information, see [AWS: Allows MFA-authenticated IAM users to manage their own credentials on the My Security Credentials page \(p. 393\)](#).

Creating IAM users (AWS API)

You can use the AWS API to create an IAM user.

To create an IAM user from the (AWS API)

1. Create a user.
 - [CreateUser](#)
2. (Optional) Give the user access to the AWS Management Console. This requires a password. You must also give the user the [URL of your account's sign-in page. \(p. 81\)](#)
 - [CreateLoginProfile](#)
3. (Optional) Give the user programmatic access. This requires access keys.
 - [CreateAccessKey](#)

Important
This is your only opportunity to view or download the secret access keys, and you must provide this information to your users before they can use the AWS API. Save the user's new access key ID and secret access key in a safe and secure place. **You will not have access to the secret keys again after this step.**
4. Add the user to one or more groups. The groups that you specify should have attached policies that grant the appropriate permissions for the user.
 - [AddUserToGroup](#)
5. (Optional) Attach a policy to the user that defines the user's permissions. **Note:** We recommend that you manage user permissions by adding the user to a group and attaching a policy to the group instead of attaching directly to a user.
 - [AttachUserPolicy](#)
6. (Optional) Add custom attributes to the user by attaching tags. For more information, see [Managing tags on IAM entities \(AWS CLI or AWS API\) \(p. 292\)](#).
7. (Optional) Give the user permission to manage their own security credentials. For more information, see [AWS: Allows MFA-authenticated IAM users to manage their own credentials on the My Security Credentials page \(p. 393\)](#).

Controlling user access to the AWS Management Console

Users with permission who sign in to your AWS account through the AWS Management Console can access your AWS resources. The following list shows the ways that you can grant users access to your

AWS account resources through the AWS Management Console. It also shows how users can access other AWS account features through the AWS website.

Note

There is no charge to use IAM.

The AWS Management Console

You create a password for each user who needs access to the AWS Management Console. Users access the console via your IAM-enabled AWS account sign-in page. For information about accessing the sign-in page, see [Signing in to the AWS Management Console as an IAM user or root user \(p. 64\)](#). For information about creating passwords, see [Managing user passwords in AWS \(p. 92\)](#).

Your AWS resources, such as Amazon EC2 instances, Amazon S3 buckets, and so on

Even if your users have passwords, they still need permission to access your AWS resources. When you create a user, that user has no permissions by default. To give your users the permissions they need, you attach policies to them. If you have many users who will be performing the same tasks with the same resources, you can assign those users to a group. Then assign the permissions to that group. For information about creating users and groups, see [IAM Identities \(users, groups, and roles\) \(p. 72\)](#). For information about using policies to set permissions, see [Access management for AWS resources \(p. 350\)](#).

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. Only that user can use that nickname in the AWS Discussion Forums.

Your AWS account billing and usage information

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

Your AWS account profile information

Users cannot access your AWS account profile information.

Your AWS account security credentials

Users cannot access your AWS account security credentials.

Note

IAM policies control access regardless of the interface. For example, you could provide a user with a password to access the AWS Management Console. 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. The policies would control which actions the user could call through a library or client that uses those access keys for authentication.

How IAM users sign in to AWS

To sign in to the AWS Management Console as an IAM user, you must provide your account ID or account alias in addition to your user name and password. When your administrator [created your IAM user in the console \(p. 77\)](#), they should have sent you your sign-in credentials, including your user name and the URL to your account sign-in page that includes your account ID or account alias.

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

Tip

To create a bookmark for your account sign-in page in your web browser, you should manually type the sign-in URL for your account in the bookmark entry. Do not use your web browser bookmark feature because redirects can obscure the sign-in URL.

You can also sign in at the following general sign-in endpoint and type your account ID or account alias manually:

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

For convenience, the AWS sign-in page uses a browser cookie to remember the IAM user name and account information. The next time the user goes to any page in the AWS Management Console, the console uses the cookie to redirect the user to the account sign-in page.

You have access only to the AWS resources that your administrator specifies in the policy that is attached to your IAM user identity. To work in the console, you must have permissions to perform the actions that the console performs, such as listing and creating AWS resources. For more information, see [Access management for AWS resources \(p. 350\)](#) and [Example IAM identity-based policies \(p. 389\)](#).

Note

If your organization has an existing identity system, you might want to create a single sign-on (SSO) option. SSO gives users access to the AWS Management Console for your account without requiring them to have an IAM user identity. SSO also eliminates the need for users to sign in to your organization's site and to AWS separately. For more information, see [Enabling custom identity broker access to the AWS console \(p. 206\)](#).

Logging sign-in details in CloudTrail

If you enable CloudTrail to log sign-in events to your logs, you need to be aware of how CloudTrail chooses where to log the events.

- If your users sign-in directly to a console, they are redirected to either a global or a regional sign-in endpoint, based on whether the selected service console supports regions. For example, the main console home page supports regions, so if you sign in to the following URL:

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

you are redirected to a regional sign-in endpoint such as `https://us-east-2.signin.aws.amazon.com`, resulting in a regional CloudTrail log entry in the user's region's log:

On the other hand, the Amazon S3 console does not support regions, so if you sign in to the following URL

```
https://alias.signin.aws.amazon.com/console/s3
```

AWS redirects you to the global sign-in endpoint at `https://signin.aws.amazon.com`, resulting in a global CloudTrail log entry.

- You can manually request a certain regional sign-in endpoint by signing in to the region-enabled main console home page using a URL syntax like the following:

```
https://alias.signin.aws.amazon.com/console?region=ap-southeast-1
```

AWS redirects you to the `ap-southeast-1` regional sign-in endpoint and results in a regional CloudTrail log event.

For more information about CloudTrail and IAM, see [Logging IAM Events with AWS CloudTrail](#).

If users need programmatic access to work with your account, you can create an access key pair (an access key ID and a secret access key) for each user, as described in [Managing access keys \(console\)](#) (p. 104).

Using MFA devices with your IAM sign-in page

IAM users who are configured with [multi-factor authentication \(MFA\)](#) (p. 111) devices must use their MFA devices to sign in to the AWS Management Console. After the user enters the user name and password, AWS checks the user's account to see if MFA is required for that user. The following sections provide information on how users complete signing in when MFA is required.

Topics

- [Signing in with a virtual MFA device](#) (p. 83)
- [Signing in with a U2F security key](#) (p. 83)
- [Signing in with a hardware MFA device](#) (p. 83)

[Signing in with a virtual MFA device](#)

If MFA is required for the user, a second sign-in page appears. In the **MFA code** box, the user must enter the numeric code provided by the MFA application.

If the MFA code is correct, the user can access the AWS Management Console. If the code is incorrect, the user can try again with another code.

A virtual MFA device can go out of sync. If a user cannot sign in to the AWS Management Console after several tries, the user is prompted to synchronize the virtual MFA device. The user can follow the on-screen prompts to synchronize the virtual MFA device. For information about how you can synchronize a device on behalf of a user in your AWS account, see [Resynchronizing virtual and hardware MFA devices](#) (p. 130).

[Signing in with a U2F security key](#)

If MFA is required for the user, a second sign-in page appears. The user needs to tap the U2F security key.

Unlike other MFA devices, U2F security keys do not go out of sync. Administrators can deactivate a U2F security key if it's lost or broken. For more information, see [Deactivating MFA devices \(console\)](#) (p. 134).

For information on browsers that support U2F and U2F devices that AWS supports, see [Supported configurations for using U2F security keys](#) (p. 121).

[Signing in with a hardware MFA device](#)

If MFA is required for the user, a second sign-in page appears. In the **MFA code** box, the user must enter the numeric code provided by a hardware MFA device.

If the MFA code is correct, the user can access the AWS Management Console. If the code is incorrect, the user can try again with another code.

A hardware MFA device can go out of sync. If a user cannot sign in to the AWS Management Console after several tries, the user is prompted to synchronize the MFA token device. The user can follow the on-screen prompts to synchronize the MFA token device. For information about how you can synchronize a device on behalf of a user in your AWS account, see [Resynchronizing virtual and hardware MFA devices](#) (p. 130).

Managing IAM users

Amazon Web Services offers multiple tools for managing the IAM users in your AWS account. You can list the IAM users in your account or in a group, or list all groups that a user is a member of. You can rename or change the path of an IAM user. You can also delete an IAM user from your AWS account.

For more information about adding, changing, or removing managed policies for an IAM user, see [Changing permissions for an IAM user \(p. 87\)](#). For information about managing inline policies for IAM users, see [Adding and removing IAM identity permissions \(p. 454\)](#), [Editing IAM policies \(p. 465\)](#), and [Deleting IAM policies \(p. 469\)](#). As a best practice, use managed policies instead of inline policies.

For information about managing IAM user passwords, see [Managing passwords for IAM users \(p. 96\)](#).

Topics

- [View user access \(p. 84\)](#)
- [Listing IAM users \(p. 84\)](#)
- [Renaming an IAM user \(p. 85\)](#)
- [Deleting an IAM user \(p. 85\)](#)

View user access

Before you delete a user, you should review its recent service-level activity. This is important because you don't want to remove access from a principal (person or application) who is using it. For more information about viewing last accessed information, see [Refining permissions in AWS using last accessed information \(p. 472\)](#).

Listing IAM users

You can list the IAM users in your AWS account or in a specific IAM group, and list all the groups that a user is in. For information about the permissions that you need in order to list users, see [Permissions required to access IAM resources \(p. 516\)](#).

To list all the users in the account

- [AWS Management Console](#): In the navigation pane, choose **Users**. The console displays the users in your AWS account.
- [AWS CLI](#): `aws iam list-users`
- [AWS API](#): `ListUsers`

To list the users in a specific group

- [AWS Management Console](#): In the navigation pane, choose **Groups**, choose the name of the group, and then choose the **Users** tab.
- [AWS CLI](#): `aws iam get-group`
- [AWS API](#): `GetGroup`

To list all the groups that a user is in

- [AWS Management Console](#): In the navigation pane, choose **Users**, choose the user name, and then choose the **Groups** tab.
- [AWS CLI](#): `aws iam list-groups-for-user`

- AWS API: [ListGroupForUser](#)

Renaming an IAM user

To change a user's name or path, you must use the AWS CLI, Tools for Windows PowerShell, or AWS API. There is no option in the console to rename a user. For information about the permissions that you need in order to rename a user, see [Permissions required to access IAM resources \(p. 516\)](#).

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

- Any policies attached to the user stay with the user under the new name.
- The user stays in the same groups under the new name.
- The unique ID for the user remains the same. For more information about unique IDs, see [Unique identifiers \(p. 604\)](#).
- Any resource or role policies that refer to the user *as a principal* (the user is being granted access) are automatically updated to use the new name or path. For example, any queue-based policies in Amazon SQS or resource-based policies in Amazon S3 are automatically updated to use the new name and path.

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

```
arn:aws:iam::111122223333:user/division_abc/subdivision_xyz/Richard
```

to this:

```
arn:aws:iam::111122223333:user/division_abc/subdivision_xyz/Rich
```

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

To rename a user

- AWS CLI: [aws iam update-user](#)
- AWS API: [UpdateUser](#)

Deleting an IAM user

You might delete an IAM user from your account if someone quits your company. If the user is only temporarily away from your company, you can disable the user's credentials instead of deleting the user entirely from the AWS account. That way, you can prevent the user from accessing the AWS account's resources during the absence but you can re-enable the user later.

For more information about disabling credentials, see [Managing access keys for IAM users \(p. 102\)](#). For information about the permissions that you need in order to delete a user, see [Permissions required to access IAM resources \(p. 516\)](#).

Topics

- [Deleting an IAM user \(console\) \(p. 86\)](#)
- [Deleting an IAM user \(AWS CLI\) \(p. 86\)](#)

Deleting an IAM user (console)

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

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

Note

Any managed policies attached to the user are detached from the user when the user is deleted. Managed policies are not deleted when you delete a user.

- Any associated MFA device

To delete an IAM user (console)

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, choose **Users**, and then select the check box next to the user name that you want to delete, not the name or row itself.
3. At the top of the page, choose **Delete user**.
4. In the confirmation dialog box, wait for the last accessed information to load before you review the data. The dialog box shows when each of the selected users last accessed an AWS service. If you attempt to delete a user that has been active within the last 30 days, you must select an additional check box to confirm that you want to delete the active user. If you want to proceed, choose **Yes, Delete**.

Deleting an IAM user (AWS CLI)

Unlike the AWS Management Console, when you delete a user with the AWS CLI, you must delete the items attached to the user manually. This procedure illustrates the process.

To delete a user from your account (AWS CLI)

1. Delete the user's password, if the user has one.

```
aws iam delete-login-profile
```

2. Delete the user's access keys, if the user has them.

```
aws iam list-access-keys
```

 (to list the user's access keys) and

```
aws iam delete-access-key
```

3. Delete the user's signing certificate. Note that when you delete a security credential, it's gone forever and can't be retrieved.

```
aws iam list-signing-certificates
```

 (to list the user's signing certificates) and

```
aws iam delete-signing-certificate
```

4. Delete the user's SSH public key, if the user has them.

```
aws iam list-ssh-public-keys
```

 (to list the user's SSH public keys) and

```
aws iam delete-ssh-public-key
```

5. Delete the user's Git credentials.

-
-
-
-
-
6. Deactivate the user's multi-factor authentication (MFA) device, if the user has one.
`aws iam list-mfa-devices` (to list the user's MFA devices), `aws iam deactivate-mfa-device` (to deactivate the device), and `aws iam delete-virtual-mfa-device` (to permanently delete a virtual MFA device)
7. Delete the user's inline policies.
`aws iam list-user-policies` (to list the inline policies for the user) and `aws iam delete-user-policy` (to delete the policy)
8. Detach any managed policies that are attached to the user.
`aws iam list-attached-user-policies` (to list the managed policies attached to the user) and `aws iam detach-user-policy` (to detach the policy)
9. Remove the user from any groups.
`aws iam list-groups-for-user` (to list the groups to which the user belongs) and `aws iam remove-user-from-group`
10. Delete the user.
`aws iam delete-user`

Changing permissions for an IAM user

You can change the permissions for an IAM user in your AWS account by changing its group memberships, by copying permissions from an existing user, by attaching policies directly to a user, or by setting a [permissions boundary \(p. 365\)](#). A permissions boundary controls the maximum permissions that a user can have. Permissions boundaries are an advanced AWS feature.

For information about the permissions that you need in order to modify the permissions for a user, see [Permissions required to access IAM resources \(p. 516\)](#).

Topics

- [View user access \(p. 87\)](#)
- [Adding permissions to a user \(console\) \(p. 87\)](#)
- [Changing permissions for a user \(console\) \(p. 90\)](#)
- [Removing a permissions policy from a user \(console\) \(p. 91\)](#)
- [Removing the permissions boundary from a user \(console\) \(p. 91\)](#)
- [Adding and removing a user's permissions \(AWS CLI or AWS API\) \(p. 91\)](#)

View user access

Before you change the permissions for a user, you should review its recent service-level activity. This is important because you don't want to remove access from a principal (person or application) who is using it. For more information about viewing last accessed information, see [Refining permissions in AWS using last accessed information \(p. 472\)](#).

Adding permissions to a user (console)

IAM offers three ways to add permissions policies to a user:

- **Add user to group** – Make the user a member of a group. The policies from the group are attached to the user.
- **Copy permissions from existing user** – Copy all group memberships, attached managed policies, inline policies, and any existing permissions boundaries from the source user.
- **Attach policies directly to user** – Attach a managed policy directly to the user. As a [best practice \(p. 528\)](#), we recommend that you instead attach your policies to a group and then make users members of the appropriate groups.

Important

If the user has a permissions boundary, then you cannot add more permissions to a user than are allowed by the permissions boundary.

Adding permissions by adding the user to a group

Adding a user to a group affects the user immediately.

To add permissions to a user by adding the user to a group

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, choose **Users**.
3. Review the current group memberships for users in the **Groups** column of the console. If necessary, add the column to the users table by completing the following steps:

1. Above the table on the far right, choose the settings symbol ().
2. In the **Manage Columns** dialog box, select the **Groups** column. Optionally, you can also clear the check box for any column headings that you do not want to appear in the users table.
3. Choose **Close** to return to the list of users.

The **Groups** column tells you to which groups the user belongs. The column includes the group names for up to two groups. If the user is a member of three or more groups, the first two groups are shown (ordered alphabetically), and the number of additional group memberships is included. For example, if the user belongs to Group A, Group B, Group C, and Group D, then the field contains the value **Group A, Group B + 2 more**. To see the total number of groups to which the user belongs, you can add the **Group count** column to the users table.

4. Choose the name of the user whose permissions you want to modify.
5. Choose the **Permissions** tab, and then choose **Add permissions**. Choose **Add user to group**.
6. Select the check box for each group that you want the user to join. The list shows each group's name and the policies that the user receives if made a member of that group.
7. (Optional) In addition to selecting from existing groups, you can choose **Create group** to define a new group:
 - a. In the new tab, for **Group name**, type a name for your new group.

Note

The number and size of IAM resources in an AWS account are limited. For more information, see [IAM and STS quotas \(p. 606\)](#). Group names can be a combination of up to 128 letters, digits, and these characters: plus (+), equal (=), comma (,), period (.), at sign (@), and hyphen (-). Names must be unique within an account. They are not distinguished by case. For example, you cannot create two groups named *TESTGROUP* and *testgroup*.

- b. Select one or more check boxes for the managed policies that you want to attach to the group. You can also create a new managed policy by choosing **Create policy**. If you do, return to this

browser tab or window when the new policy is done; choose **Refresh**; and then choose the new policy to attach it to your group. For more information, see [Creating IAM policies \(p. 439\)](#).

- c. Choose **Create group**.
- d. Return to the original tab, refresh your list of groups. Then select the check box for your new group.
8. Choose **Next: Review** to see the list of group memberships to be added to the user. Then choose **Add permissions**.

Adding permissions by copying from another user

Copying permissions affects the user immediately.

To add permissions to a user by copying permissions from another user

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. Choose **Users** in the navigation pane, choose the name of the user whose permissions you want to modify, and then choose the **Permissions** tab.
3. Choose **Add permissions**, and then choose **Copy permissions from existing user**. The list displays available users along with their group memberships and attached policies. If the full list of groups or policies don't fit on one line, you can choose the link for **and n more**. Doing that opens a new browser tab and see the full list of policies (**Permissions** tab) and groups (**Groups** tab).
4. Select the radio button next to the user whose permissions you want to copy.
5. Choose **Next: Review** to see the list of changes that are to be made to the user. Then choose **Add permissions**.

Adding permissions by attaching policies directly to the user

Attaching policies affects the user immediately.

To add permissions to a user by directly attaching managed policies

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. Choose **Users** in the navigation pane, choose the name of the user whose permissions you want to modify, and then choose the **Permissions** tab.
3. Choose **Add permissions**, and then choose **Attach existing policies directly to user**.
4. Select one or more check boxes for the managed policies that you want to attach to the user. You can also create a new managed policy by choosing **Create policy**. If you do, return to this browser tab or window when the new policy is done. Choose **Refresh**; and then select the check box for the new policy to attach it to your user. For more information, see [Creating IAM policies \(p. 439\)](#).
5. Choose **Next: Review** to see the list of policies that are to be attached to the user. Then choose **Add permissions**.

Setting the permissions boundary for a user

Setting a permissions boundary affects the user immediately.

To set the permissions boundary 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, choose **Users**.
3. Choose the name of the user whose permissions boundary you want to change.
4. Choose the **Permissions** tab. If necessary, open the **Permissions boundary** section and then choose **Set boundary**.
5. Select the policy that you want to use for the permissions boundary.
6. Choose **Set boundary**.

Changing permissions for a user (console)

IAM allows you to change the permissions that are associated with a user in the following ways:

- **Edit a permissions policy** – Edit a user's inline policy, the inline policy of the user's group, or edit a managed policy that is attached to the user directly or from a group. If the user has a permissions boundary, then you cannot provide more permissions than are allowed by the policy that was used as the user's permissions boundary.
- **Changing the permissions boundary** – Change the policy that is used as the permissions boundary for the user. This can expand or restrict the maximum permissions that a user can have.

Editing a permissions policy attached to a user

Changing permissions affects the user immediately.

To edit a user's attached managed policies

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, choose **Users**.
3. Choose the name of the user whose permissions policy you want to change.
4. Choose the **Permissions** tab. If necessary, open the **Permissions policies** section.
5. Choose the name of the policy that you want to edit to view details about the policy. Choose the **Used as** tab to view other entities that might be affected if you edit the policy.
6. Choose the **Permissions tab** and review the permissions granted by the policy. Then choose **Edit policy**.
7. Edit the policy using the **Visual editor** tab or the **JSON** tab. For more information, see [Editing IAM policies \(p. 465\)](#).
8. Choose **Review policy**, review the policy summary, and then choose **Save changes**.

Changing the permissions boundary for a user

Changing a permissions boundary affects the user immediately.

To change the policy used to set the permissions boundary 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, choose **Users**.
3. Choose the name of the user whose permissions boundary you want to change.
4. Choose the **Permissions** tab. If necessary, open the **Permissions boundary** section and then choose **Change boundary**.

5. Select the policy that you want to use for the permissions boundary.
6. Choose **Change boundary**.

Removing a permissions policy from a user (console)

Removing a policy affects the user immediately.

To revoke permissions for IAM users

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, choose **Users**.
3. Choose the name of the user whose permissions boundary you want to remove.
4. Choose the **Permissions** tab.
5. If you want to revoke permissions by removing an existing policy, view the **Policy type** to understand how the user is getting that policy before choosing **X** to remove the policy:
 - If the policy applies because of group membership, then choosing **X** removes the user from the group. Remember that you might have multiple policies attached to a single group. If you remove a user from a group, the user loses access to *all* policies that it received through that group membership.
 - If the policy is a managed policy attached directly to the user, then choosing **X** detaches the policy from the user. This does not affect the policy itself or any other entity that the policy might be attached to.
 - If the policy is an inline embedded policy, then choosing **X** removes the policy from IAM. Inline policies that are attached directly to a user exist only on that user.

Removing the permissions boundary from a user (console)

Removing a permissions boundary affects the user immediately.

To remove the permissions boundary from 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, choose **Users**.
3. Choose the name of the user whose permissions boundary you want to remove.
4. Choose the **Permissions** tab. If necessary, open the **Permissions boundary** section and then choose **Remove boundary**.
5. Choose **Remove** to confirm that you want to remove the permissions boundary.

Adding and removing a user's permissions (AWS CLI or AWS API)

To add or remove permissions programmatically, you must add or remove the group memberships, attach or detach the managed policies, or add or delete the inline policies. For more information, see the following topics:

- [Adding and removing users in an IAM group \(p. 163\)](#)
- [Adding and removing IAM identity permissions \(p. 454\)](#)

Managing user passwords in AWS

You can manage passwords for your AWS account root user and for IAM users in your account. IAM users need passwords in order to access the AWS Management Console. Users do not need passwords to access AWS resources programmatically by using the AWS CLI, Tools for Windows PowerShell, the AWS SDKs or APIs. For those environments, users need [access keys \(p. 102\)](#) instead.

Contents

- [Changing the AWS account root user password \(p. 92\)](#)
- [Setting an account password policy for IAM users \(p. 93\)](#)
- [Managing passwords for IAM users \(p. 96\)](#)
- [Permitting IAM users to change their own passwords \(p. 99\)](#)
- [How an IAM user changes their own password \(p. 101\)](#)

Changing the AWS account root user password

You must be signed in as the AWS account root user and not an IAM user to change the root user password. To learn how to reset a *forgotten* root user password, see [Resetting lost or forgotten passwords or access keys for AWS \(p. 110\)](#).

To change the password for the root user

1. Use your AWS account email address and password to sign in to the [AWS Management Console](#) as the AWS account root user.

Note

If you see three text boxes, then you previously signed in to the console with [IAM user](#) credentials. Your browser might remember this preference and open this account-specific sign-in page every time that you try to sign in. You cannot use the IAM user sign-in page to sign in as the account owner. If you see the [IAM user sign-in page](#), choose **Sign in using root user email** near the bottom of the page. This returns you to the main sign-in page. From there, you can sign in as the root user using your AWS account email address and password.

2. In the upper right corner of the console, choose your account name or number and then choose **My Account**.
3. On the right side of the page, next to the **Account Settings** section, choose **Edit**.
4. On the **Password** line choose **Edit** to change your password.
5. Choose a strong password. Although you can [set an account password policy for IAM users \(p. 93\)](#), that policy does not apply to your AWS account root user.

AWS requires that your password meet these conditions:

- have a minimum of 8 characters and a maximum of 128 characters
- include a minimum of three of the following mix of character types: uppercase, lowercase, numbers, and ! @ # \$ % ^ & * () <> [] {} | _+=- symbols
- not be identical to your AWS account name or email address

Note

AWS is rolling out improvements to the sign-in process. One of those improvements is to enforce a more secure password policy for your account. If your account has been upgraded, you are required to meet the password policy above. If your account has not yet been upgraded, then AWS does not enforce this policy, but highly recommends that you follow its guidelines for a more secure password.

To protect your password, it's important to follow these best practices:

- Change your password periodically and keep your password private, since anyone who knows your password may access your account.
- Use a different password on AWS than you use on other sites.
- Avoid passwords that are easy to guess. These include passwords such as `secret`, `password`, `amazon`, or `123456`. They also include things like a dictionary word, your name, email address, or other personal information that can easily be obtained.

Setting an account password policy for IAM users

You can set a custom password policy on your AWS account to specify complexity requirements and mandatory rotation periods for your IAM users' passwords. If you don't set a custom password policy, IAM user passwords must meet the default AWS password policy. For more information, see [Custom password policy options \(p. 94\)](#).

Topics

- [Rules for setting a password policy \(p. 93\)](#)
- [Permissions required to set a password policy \(p. 93\)](#)
- [Default password policy \(p. 94\)](#)
- [Custom password policy options \(p. 94\)](#)
- [Setting a password policy \(console\) \(p. 95\)](#)
- [Setting a password policy \(AWS CLI\) \(p. 95\)](#)
- [Setting a password policy \(AWS API\) \(p. 96\)](#)

Rules for setting a password policy

The IAM password policy does not apply to the AWS account root user password or IAM user access keys. If a password expires, the IAM user can't sign in to the AWS Management Console but can continue to use their access keys.

When you create or change a password policy, most of the password policy settings are enforced the next time your users change their passwords. However, some of the settings are enforced immediately. For example:

- When the minimum length and character type requirements change, these settings are enforced the next time that your users change their passwords. Users are not forced to change their existing passwords, even if the existing passwords do not adhere to the updated password policy.
- When you set a password expiration period, the expiration period is enforced immediately. For example, assume that you set a password expiration period of 90 days. In that case, the password expires for all IAM users whose existing password is older than 90 days. Those users are required to change their password the next time that they sign in.

You can't create a "lockout policy" to lock a user out of the account after a specified number of failed sign-in attempts. For enhanced security, we recommend that you combine a strong password policy with multi-factor authentication (MFA). For more information about MFA, see [Using multi-factor authentication \(MFA\) in AWS \(p. 111\)](#).

Permissions required to set a password policy

You must configure permissions to allow an IAM entity (user or role) to view or edit their account password policy. You can include the following password policy actions in an IAM policy:

- `iam:GetAccountPasswordPolicy` – Allows the entity to view the password policy for their account
- `iam>DeleteAccountPasswordPolicy` – Allows the entity to delete the custom password policy for their account and revert to the default password policy
- `iam:UpdateAccountPasswordPolicy` – Allows the entity to create or change the custom password policy for their account

The following policy allows full access to view and edit the account password policy. To learn how to create an IAM policy using this example JSON policy document, see [the section called “Creating policies on the JSON tab” \(p. 440\)](#).

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "FullAccessPasswordPolicy",  
            "Effect": "Allow",  
            "Action": [  
                "iam:GetAccountPasswordPolicy",  
                "iam>DeleteAccountPasswordPolicy",  
                "iam:UpdateAccountPasswordPolicy"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

For information about the permissions required for an IAM user to change their own password, see [Permitting IAM users to change their own passwords \(p. 99\)](#).

Default password policy

If an administrator does not set a custom password policy, IAM user passwords must meet the default AWS password policy. The default password policy enforces the following conditions:

- Minimum password length of 8 characters and a maximum length of 128 characters
- Minimum of three of the following mix of character types: uppercase, lowercase, numbers, and ! @ # \$ % ^ & * () _ + - = [] { } | ' symbols
- Not be identical to your AWS account name or email address

Custom password policy options

When you configure a custom password policy for your account, you can specify the following conditions:

- **Password minimum length** – You can specify a minimum of 6 characters and a maximum of 128 characters.
- **Password strength** – You can select any of the following check boxes to define the strength of your IAM user passwords:
 - Require at least one uppercase letter from Latin alphabet (A–Z)
 - Require at least one lowercase letter from Latin alphabet (a–z)
 - Require at least one number
 - Require at least one nonalphanumeric character ! @ # \$ % ^ & * () _ + - = [] { } | '
- **Enable password expiration** – You can select and specify a minimum of 1 and a maximum of 1,095 days that IAM user passwords are valid after they are set. For example, after 90 days a user's password expires and they must set a new password before accessing the AWS Management Console. The AWS

Management Console warns IAM users when they are within 15 days of password expiration. IAM users can change their password at any time if they have permission. When they set a new password, the expiration period for that password starts over. An IAM user can have only one valid password at a time.

- **Password expiration requires administrator reset** – Select this option to prevent IAM users from updating their own passwords after the password expires. Before you select this option, confirm that your AWS account has more than one user with administrative permissions to reset IAM user passwords. Also consider providing access keys to allow administrators to reset IAM user passwords programmatically. If you clear this check box, IAM users with expired passwords must still set a new password before they can access the AWS Management Console.
- **Allow users to change their own password** – You can permit all IAM users in your account to use the IAM console to change their own passwords, as described in [Permitting IAM users to change their own passwords \(p. 99\)](#). Alternatively, you can allow only some users to manage passwords, either for themselves or for others. To do so, you clear this check box. For more information about using policies to limit who can manage passwords, see [Permitting IAM users to change their own passwords \(p. 99\)](#).
- **Prevent password reuse** – You can prevent IAM users from reusing a specified number of previous passwords. You can specify a minimum number of 1 and a maximum number of 24 previous passwords that can't be repeated.

Setting a password policy (console)

You can use the AWS Management Console to create, change, or delete a custom password policy.

To create a custom password policy (console)

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, choose **Account settings**.
3. In the **Password policy** section, choose **Change password policy**.
4. Select the options that you want to apply to your password policy and choose **Save changes**.

To change a custom password policy (console)

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, choose **Account settings**.
3. In the **Password policy** section, choose **Change**.
4. Select the options that you want to apply to your password policy and choose **Save changes**.

To delete a password policy (console)

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, choose **Account settings**.
3. In the **Password policy** section, choose **Delete**.
4. Confirm that you want to delete the custom password policy by choosing **Delete custom**.

Setting a password policy (AWS CLI)

You can use the AWS Command Line Interface to set a password policy.

To manage the custom account password policy from the AWS CLI

Run the following commands:

- To create or change the custom password policy: `aws iam update-account-password-policy`
- To view the password policy: `aws iam get-account-password-policy`
- To delete the custom password policy: `aws iam delete-account-password-policy`

Setting a password policy (AWS API)

You can use AWS API operations to set a password policy.

To manage the custom account password policy from the AWS API

Call the following operations:

- To create or change the custom password policy: `UpdateAccountPasswordPolicy`
- To view the password policy: `GetAccountPasswordPolicy`
- To delete the custom password policy: `DeleteAccountPasswordPolicy`

Managing passwords for IAM users

IAM users who use the AWS Management Console to work with AWS resources must have a password in order to sign in. You can create, change, or delete a password for an IAM user in your AWS account.

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

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

For more information about how IAM users sign in to the AWS Management Console, see [Signing in to the AWS Management Console as an IAM user or root user \(p. 64\)](#).

Even if your users have their own passwords, they still need permissions to access your AWS resources. By default, a user has no permissions. To give your users the permissions they need, you assign policies to them or to the groups they belong to. For information about creating users and groups, see [IAM Identities \(users, groups, and roles\) \(p. 72\)](#). For information about using policies to set permissions, see [Changing permissions for an IAM user \(p. 87\)](#).

You can grant users permission to change their own passwords. For more information, see [Permitting IAM users to change their own passwords \(p. 99\)](#). For information about how users access your account sign-in page, see [Signing in to the AWS Management Console as an IAM user or root user \(p. 64\)](#).

Topics

- [Creating, changing, or deleting an IAM user password \(console\) \(p. 96\)](#)
- [Creating, changing, or deleting an IAM user password \(AWS CLI\) \(p. 98\)](#)
- [Creating, changing, or deleting an IAM user password \(AWS API\) \(p. 99\)](#)

Creating, changing, or deleting an IAM user password (console)

You can use the AWS Management Console to manage passwords for your IAM users.

When users leave your organization or no longer need AWS access, it is important to find the credentials that they were using and ensure that they are no longer operational. Ideally, you delete credentials if

they are no longer needed. You can always recreate them at a later date if the need arises. At the very least, you should change the credentials so that the former users no longer have access.

To add a password for an IAM user (console)

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, choose **Users**.
3. Choose the name of the user whose password you want to create.
4. Choose the **Security credentials** tab, and then under **Sign-in credentials**, choose **Manage password** next to **Console password**.
5. In **Manage console access**, for **Console access** choose **Enable** if not already selected. If console access is disabled, then no password is needed.
6. For **Set password**, choose whether to have IAM generate a password or create a custom password:
 - To have IAM generate a password, choose **Autogenerated password**.
 - To create a custom password, choose **Custom password**, and type the password.

Note

The password that you create must meet the account's [password policy \(p. 93\)](#).

7. To require the user to create a new password when signing in, choose **Require password reset**. Then choose **Apply**.

Important

If you select the **Require password reset** option, make sure that the user has permission to change his or her password. For more information, see [Permitting IAM users to change their own passwords \(p. 99\)](#).

8. If you choose the option to generate a password, choose **Show** in the **New password** dialog box. This lets you view the password so you can share it with the user.

Important

For security reasons, you cannot access the password after completing this step, but you can create a new password at any time.

To change the password for an IAM user (console)

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, choose **Users**.
3. Choose the name of the user whose password you want to change.
4. Choose the **Security credentials** tab, and then under **Sign-in credentials**, choose **Manage password** next to **Console password**.
5. In **Manage console access**, for **Console access** choose **Enable** if not already selected. If console access is disabled, then no password is needed.
6. For **Set password**, choose whether to have IAM generate a password or create a custom password:
 - To have IAM generate a password, choose **Autogenerated password**.
 - To create a custom password, choose **Custom password**, and type the password.

Note

The password that you create must meet the account's [password policy \(p. 93\)](#), if one is currently set.

7. To require the user to create a new password when signing in, choose **Require password reset**. Then choose **Apply**.

Important

If you select the **Require password reset** option, make sure that the user has permission to change his or her password. For more information, see [Permitting IAM users to change their own passwords \(p. 99\)](#).

8. If you choose the option to generate a password, choose **Show** in the **New password** dialog box. This lets you view the password so you can share it with the user.

Important

For security reasons, you cannot access the password after completing this step, but you can create a new password at any time.

To delete (disable) an IAM user's password (console)

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, choose **Users**.
3. Choose the name of the user whose password you want to delete.
4. Choose the **Security credentials** tab, and then under **Sign-in credentials**, choose **Manage password** next to **Console password**.
5. For **Console access**, choose **Disable**, and then choose **Apply**.

Important

When you delete a user's password, the user can no longer sign in to the AWS Management Console. If the user has active access keys, they continue to function and allow access through the AWS CLI, Tools for Windows PowerShell, or AWS API function calls.

Creating, changing, or deleting an IAM user password (AWS CLI)

You can use the AWS CLI API to manage passwords for your IAM users.

To create a password (AWS CLI)

1. (Optional) To determine whether a user has a password, run this command: `aws iam get-login-profile`
2. To create a password, run this command: `aws iam create-login-profile`

To change a user's password (AWS CLI)

1. (Optional) To determine whether a user has a password, run this command: `aws iam get-login-profile`
2. To change a password, run this command: `aws iam update-login-profile`

To delete (disable) a user's password (AWS CLI)

1. (Optional) To determine whether a user has a password, run this command: `aws iam get-login-profile`
2. (Optional) To determine when a password was last used, run this command: `aws iam get-user`
3. To delete a password, run this command: `aws iam delete-login-profile`

Important

When you delete a user's password, the user can no longer sign in to the AWS Management Console. If the user has active access keys, they continue to function and allow access through

the AWS CLI, Tools for Windows PowerShell, or AWS API function calls. When you use the AWS CLI, Tools for Windows PowerShell, or AWS API to delete a user from your AWS account, you must first delete the password using this operation. For more information, see [Deleting an IAM user \(AWS CLI\) \(p. 86\)](#).

Creating, changing, or deleting an IAM user password (AWS API)

You can use the AWS API to manage passwords for your IAM users.

To create a password (AWS API)

1. (Optional) To determine whether a user has a password, call this operation: [GetLoginProfile](#)
2. To create a password, call this operation: [CreateLoginProfile](#)

To change a user's password (AWS API)

1. (Optional) To determine whether a user has a password, call this operation: [GetLoginProfile](#)
2. To change a password, call this operation: [UpdateLoginProfile](#)

To delete (disable) a user's password (AWS API)

1. (Optional) To determine whether a user has a password, run this command: [GetLoginProfile](#)
2. (Optional) To determine when a password was last used, run this command: [GetUser](#)
3. To delete a password, run this command: [DeleteLoginProfile](#)

Important

When you delete a user's password, the user can no longer sign in to the AWS Management Console. If the user has active access keys, they continue to function and allow access through the AWS CLI, Tools for Windows PowerShell, or AWS API function calls. When you use the AWS CLI, Tools for Windows PowerShell, or AWS API to delete a user from your AWS account, you must first delete the password using this operation. For more information, see [Deleting an IAM user \(AWS CLI\) \(p. 86\)](#).

Permitting IAM users to change their own passwords

You can grant IAM users the permission to change their own passwords for signing in to the AWS Management Console. You can do this in one of two ways:

- [Allow all IAM users in the account to change their own passwords \(p. 99\)](#).
- [Allow only selected IAM users to change their own passwords \(p. 100\)](#). In this scenario, you disable the option for all users to change their own passwords and you use an IAM policy to grant permissions to only some users. This approach allows those users to change their own passwords and optionally other credentials like their own access keys.

Important

We recommend that you [set a custom password policy \(p. 93\)](#) that requires IAM users to create strong passwords.

To allow all IAM users change their own passwords

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 **Account settings**.

3. In the **Password policy** section, choose **Change password policy** if your account uses the default password policy. If your account uses a custom password policy, choose **Change**.
4. Select **Allow users to change their own password**, and then click **save changes**.
5. Provide users with the following instructions for changing their passwords: [How an IAM user changes their own password \(p. 101\)](#).

For information about the AWS CLI, Tools for Windows PowerShell, and API commands that you can use to change the account's password policy (which includes letting all users change their own passwords), see [Setting a password policy \(AWS CLI\) \(p. 95\)](#).

To allow selected IAM users change their own passwords

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 **Account settings**.
3. In the **Password policy** section, make sure that **Allow users to change their own password** is not selected. If this check box is selected, all users can change their own passwords. (See the previous procedure.)
4. Create the users who should be allowed to change their own password, if they do not already exist. For details, see [Creating an IAM user in your AWS account \(p. 76\)](#).
5. Create an IAM group for the users who should be allowed to change their passwords, and then add the users from the previous step to the group. For details, see [Creating your first IAM admin user and group \(p. 20\)](#) and [Managing IAM groups \(p. 162\)](#).

This step is optional, but it's a best practice to use groups to manage permissions. That way, you can add and remove users and change the permissions for the group as a whole.

6. Assign the following policy to the group. For more information, see [Managing IAM policies \(p. 438\)](#).

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "iam:GetAccountPasswordPolicy",  
            "Resource": "*"  
        },  
        {  
            "Effect": "Allow",  
            "Action": "iam:ChangePassword",  
            "Resource": "arn:aws:iam::account-id-without-hyphens:user/${aws:username}"  
        }  
    ]  
}
```

This policy grants access to the [ChangePassword](#) action, which lets users change only their own passwords from the console, the AWS CLI, Tools for Windows PowerShell, or the API. It also grants access to the [GetAccountPasswordPolicy](#) action, which lets the user view the current password policy; this permission is required so that the user can view the account password policy on the **Change password** page. The user must be allowed to read the current password policy to ensure that the changed password meets the requirements of the policy.

7. Provide users with the following instructions for changing their passwords: [How an IAM user changes their own password \(p. 101\)](#).

For more information

For more information on managing credentials, see the following topics:

- [Permitting IAM users to change their own passwords \(p. 99\)](#)
- [Managing user passwords in AWS \(p. 92\)](#)
- [Setting an account password policy for IAM users \(p. 93\)](#)
- [Managing IAM policies \(p. 438\)](#)
- [How an IAM user changes their own password \(p. 101\)](#)

How an IAM user changes their own password

If you have been granted permission to change your own IAM user password, you can use a special page in the AWS Management Console to do this. You can also use the AWS CLI or AWS API.

Topics

- [Permissions required \(p. 101\)](#)
- [How IAM users change their own password \(console\) \(p. 101\)](#)
- [How IAM users change their own password \(AWS CLI or AWS API\) \(p. 102\)](#)

Permissions required

To change the password for your own IAM user, you must have the permissions from the following policy: [AWS: Allows IAM users to change their own console password on the My Security Credentials page \(p. 401\)](#).

How IAM users change their own password (console)

The following procedure describes how IAM users can use the AWS Management Console to change their own password.

To change your own IAM user password (console)

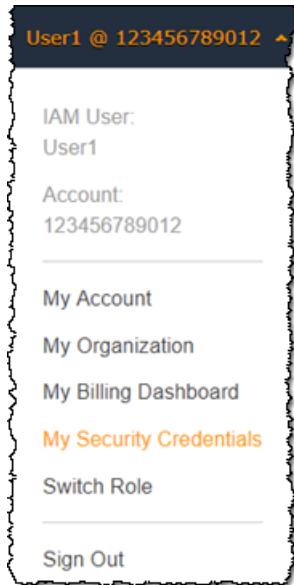
1. Use your AWS account ID or account alias, your IAM user name, and your password to sign in to the [IAM console](#).

Note

For your convenience, the AWS sign-in page uses a browser cookie to remember your IAM user name and account information. If you previously signed in as a different user, choose **Sign in to a different account** near the bottom of the page to return to the main sign-in page. From there, you can type your AWS account ID or account alias to be redirected to the IAM user sign-in page for your account.

To get your AWS account ID, contact your administrator.

2. In the navigation bar on the upper right, choose your user name, and then choose **My Security Credentials**.



3. On the **AWS IAM Credentials** tab, choose **Change password**.
4. For **Current password**, enter your current password. Enter a new password for **New password** and **Confirm new password**. Then choose **Change password**.

Note

The new password must meet the requirements of the account password policy. For more information, see [Setting an account password policy for IAM users \(p. 93\)](#).

How IAM users change their own password (AWS CLI or AWS API)

The following procedure describes how IAM users can use the AWS CLI or AWS API to change their own password.

To change your own IAM password, use the following:

- AWS CLI: `aws iam change-password`
- AWS API: `ChangePassword`

Managing access keys for IAM users

[Follow us on Twitter](#)

Note

If you found this page because you are looking for information about the Product Advertising API to sell Amazon products on your website, see the [Product Advertising API 5.0 Documentation](#).

Access keys are long-term credentials for an IAM user or the AWS account root user. You can use access keys to sign programmatic requests to the AWS CLI or AWS API (directly or using the AWS SDK). For more information, see [Signing AWS API Requests](#) in the *Amazon Web Services General Reference*.

Access keys consist of two parts: an access key ID (for example, `AKIAIOSFODNN7EXAMPLE`) and a secret access key (for example, `wJalrXutnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). Like a user name and password, you must use both the access key ID and secret access key together to authenticate your requests. Manage your access keys as securely as you do your user name and password.

Important

Do not provide your access keys to a third party, even to help [find your canonical user ID](#). By doing this, you might give someone permanent access to your account.

As a best practice, use temporary security credentials (IAM roles) instead of access keys, and disable any AWS account root user access keys. For more information, see [Best Practices for Managing AWS Access Keys](#) in the *Amazon Web Services General Reference*.

If you still need to use long-term access keys, you can create, modify, view, or rotate your access keys (access key IDs and secret access keys). You can have a maximum of two access keys. This allows you to rotate the active keys according to best practices.

When you create an access key pair, save the access key ID and secret access key in a secure location. The secret access key is available only at the time you create it. If you lose your secret access key, you must delete the access key and create a new one. For more details, see [Resetting lost or forgotten passwords or access keys for AWS \(p. 110\)](#).

Topics

- [Permissions required \(p. 103\)](#)
- [Managing access keys \(console\) \(p. 104\)](#)
- [Managing access keys \(AWS CLI\) \(p. 106\)](#)
- [Managing access keys \(AWS API\) \(p. 106\)](#)
- [Rotating access keys \(p. 106\)](#)
- [Auditing access keys \(p. 109\)](#)

Permissions required

To create access keys for your own IAM user, you must have the permissions from the following policy:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "CreateOwnAccessKeys",  
            "Effect": "Allow",  
            "Action": [  
                "iam:CreateAccessKey",  
                "iam:GetUser",  
                "iam>ListAccessKeys"  
            ],  
            "Resource": "arn:aws:iam::*:user/${aws:username}"  
        }  
    ]  
}
```

To rotate access keys for your own IAM user, you must have the permissions from the following policy:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "ManageOwnAccessKeys",  
            "Effect": "Allow",  
            "Action": [  
                "iam:CreateAccessKey",  
                "iam>DeleteAccessKey",  
                "iam:GetAccessKeyLastUsed",  
            ]  
        }  
    ]  
}
```

```
        "iam:GetUser",
        "iam>ListAccessKeys",
        "iam:UpdateAccessKey"
    ],
    "Resource": "arn:aws:iam::*:user/${aws:username}"
}
}
```

Managing access keys (console)

You can use the AWS Management Console to manage an IAM user's access keys.

To create, modify, or delete your own IAM user access keys (console)

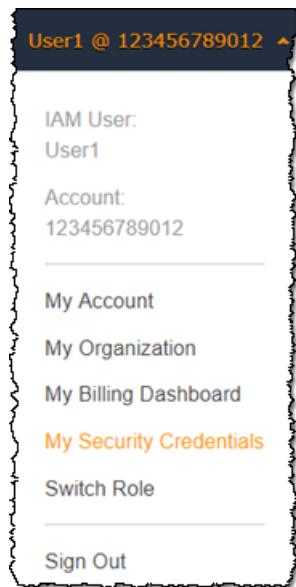
1. Use your AWS account ID or account alias, your IAM user name, and your password to sign in to the [IAM console](#).

Note

For your convenience, the AWS sign-in page uses a browser cookie to remember your IAM user name and account information. If you previously signed in as a different user, choose **Sign in to a different account** near the bottom of the page to return to the main sign-in page. From there, you can type your AWS account ID or account alias to be redirected to the IAM user sign-in page for your account.

To get your AWS account ID, contact your administrator.

2. In the navigation bar on the upper right, choose your user name, and then choose **My Security Credentials**.



3. Expand the **Access keys (access key ID and secret access key)** section.
4. Do any of the following:
 - To create an access key, choose **Create New Access Key**. If this feature is disabled, then you must delete one of the existing keys before you can create a new one. A warning explains that you have only this one opportunity to view or download the secret access key. To copy the key to paste it somewhere else for safekeeping, choose **Show Access Key**. To save the access key ID and secret access key to a .csv file to a secure location on your computer, choose **Download Key File**.
 - To disable an active access key, choose **Make Inactive**.

- To reenable an inactive access key, choose **Make Active**.
- To delete your access key, choose **Delete**. AWS recommends that before you do this, you first deactivate the key and test that it's no longer in use. When you use the AWS Management Console, you must deactivate your key before deleting it.

To create, modify, or delete another IAM user's access keys (console)

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, choose **Users**.
3. Choose the name of the user whose access keys you want to manage, and then choose the **Security credentials** tab.
4. In the **Access keys** section, do any of the following:
 - To create an access key, choose **Create access key**. Then choose **Download .csv file** to save the access key ID and secret access key to a CSV file on your computer. Store the file in a secure location. You will not have access to the secret access key again after this dialog box closes. After you download the CSV file, choose **Close**. When you create an access key, the key pair is active by default, and you can use the pair right away.
 - To disable an active access key, choose **Make inactive**.
 - To reenable an inactive access key, choose **Make active**.
 - To delete your access key, choose **Delete**. AWS recommends that before you do this, you first deactivate the key and test that it's no longer in use. When you use the AWS Management Console, you must deactivate your key before deleting it.

To list the access keys for an IAM user (console)

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, choose **Users**.
3. Choose the name of the intended user, and then choose the **Security credentials** tab. The user's access keys and the status of each key is displayed.

Note

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

To list the access key IDs for multiple IAM users (console)

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, choose **Users**.
3. If necessary, add the **Access key ID** column to the users table by completing the following steps:
 - a. Above the table on the far right, choose the settings icon ().
 - b. In **Manage columns**, select **Access key ID**.
 - c. Choose **Close** to return to the list of users.
4. The **Access key ID** column shows each access key ID, followed by its state; for example, **23478207027842073230762374023 (Active)** or **22093740239670237024843420327 (Inactive)**.

You can use this information to view and copy the access keys for users with one or two access keys. The column displays **None** for users with no access key.

Note

Only the user's access key ID and status is visible. The secret access key can only be retrieved when the key is created.

To find which IAM user owns a specific access key (console)

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, choose **Users**.
3. In the search box, type or paste the access key ID of the user you want to find.
4. If necessary, add the **Access key ID** column to the users table by completing the following steps:
 - a. Above the table on the far right, choose the settings icon ().
 - b. In **Manage columns**, select **Access key ID**.
 - c. Choose **Close** to return to the list of users and confirm that the filtered user owns the specified access key.

Managing access keys (AWS CLI)

To manage an IAM user's access keys from the AWS CLI, run the following commands.

- To create an access key: `aws iam create-access-key`
- To disable or reenable an access key: `aws iam update-access-key`
- To list a user's access keys: `aws iam list-access-keys`
- To determine when an access key was most recently used: `aws iam get-access-key-last-used`
- To delete an access key: `aws iam delete-access-key`

Managing access keys (AWS API)

To manage an IAM user's access keys from the AWS API, call the following operations.

- To create an access key: `CreateAccessKey`
- To disable or reenable an access key: `UpdateAccessKey`
- To list a user's access keys: `ListAccessKeys`
- To determine when an access key was most recently used: `GetAccessKeyLastUsed`
- To delete an access key: `DeleteAccessKey`

Rotating access keys

As a security best practice, we recommend that you regularly rotate (change) IAM user access keys. If your administrator granted you the necessary permissions, you can rotate your own access keys.

Administrators, for details about granting your users permissions to rotate their own access keys, see [AWS: Allows IAM users to manage their own password, access keys, and SSH public keys on the My Security Credentials page \(p. 402\)](#). You can also apply a password policy to your account to require that all of your IAM users periodically rotate their passwords. You can choose how often they must do so. For more information, see [Setting an account password policy for IAM users \(p. 93\)](#).

Important

As a best practice, do not use your AWS account root user. If you use the AWS account root user credentials, we recommend that you also regularly rotate them. The account password policy does not apply to the root user credentials. IAM users cannot manage credentials for the AWS account root user, so you must use the root user credentials (not a user's) to change the root user credentials. Note that we recommend against using the root user for everyday work in AWS.

Topics

- [Rotating IAM user access keys \(console\) \(p. 107\)](#)
- [Rotating access keys \(AWS CLI\) \(p. 108\)](#)
- [Rotating access keys \(AWS API\) \(p. 109\)](#)

Rotating IAM user access keys (console)

You can rotate access keys from the AWS Management Console.

To rotate access keys for an IAM user without interrupting your applications (console)

1. While the first access key is still active, create a second access key.
 - a. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
 - b. In the navigation pane, choose **Users**.
 - c. Choose the name of the intended user, and then choose the **Security credentials** tab.
 - d. Choose **Create access key** and then choose **Download .csv file** to save the access key ID and secret access key to a .csv file on your computer. Store the file in a secure location. You will not have access to the secret access key again after this closes. After you have downloaded the .csv file, choose **Close**.

The new access key is active by default. At this point, the user has two active access keys.

2. Update all applications and tools to use the new access key.
3. Determine whether the first access key is still in use by reviewing the **Last used** column for the oldest access key. One approach is to wait several days and then check the old access key for any use before proceeding.
4. Even if the **Last used** column value indicates that the old key has never been used, we recommend that you do not immediately delete the first access key. Instead, choose **Make inactive** to deactivate the first access key.
5. Use only the new access key to confirm that your applications are working. Any applications and tools that still use the original access key will stop working at this point because they no longer have access to AWS resources. If you find such an application or tool, you can choose **Make active** to reenable the first access key. Then return to [Step 3 \(p. 107\)](#) and update this application to use the new key.
6. After you wait some period of time to ensure that all applications and tools have been updated, you can delete the first access key:
 - a. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
 - b. In the navigation pane, choose **Users**.
 - c. Choose the name of the intended user, and then choose the **Security credentials** tab.
 - d. Locate the access key to delete and choose its **X** button at the far right of the row. Enter the access key ID to confirm the deletion and then choose **Delete**.

To determine when access keys need rotating (console)

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, choose **Users**.
3. If necessary, add the **Access key age** column to the users table by completing the following steps:
 - a. Above the table on the far right, choose the settings icon ().
 - b. In **Manage columns**, select **Access key age**.
 - c. Choose **Close** to return to the list of users.
4. The **Access key age** column shows the number of days since the oldest active access key was created. You can use this information to find users with access keys that need rotating. The column displays **None** for users with no access key.

Rotating access keys (AWS CLI)

You can rotate access keys from the AWS Command Line Interface.

To rotate access keys without interrupting your applications (AWS CLI)

1. While the first access key is still active, create a second access key, which is active by default. Run the following command:
 - `aws iam create-access-key`

At this point, the user has two active access keys.
2. Update all applications and tools to use the new access key.
3. Determine whether the first access key is still in use by using this command:
 - `aws iam get-access-key-last-used`

One approach is to wait several days and then check the old access key for any use before proceeding.
4. Even if step **Step 3** indicates no use of the old key, we recommend that you do not immediately delete the first access key. Instead, change the state of the first access key to **Inactive** using this command:
 - `aws iam update-access-key`
5. Use only the new access key to confirm that your applications are working. Any applications and tools that still use the original access key will stop working at this point because they no longer have access to AWS resources. If you find such an application or tool, you can switch its state back to **Active** to reenable the first access key. Then return to step **Step 2** and update this application to use the new key.
6. After you wait some period of time to ensure that all applications and tools have been updated, you can delete the first access key with this command:
 - `aws iam delete-access-key`

For more information, see the following:

- [How to Rotate Access Keys for IAM Users](#). This entry on the *AWS Security Blog* provides more information on key rotation.

- [Security best practices in IAM \(p. 527\)](#). This page provides general recommendations for helping to secure your AWS resources.

Rotating access keys (AWS API)

You can rotate access keys using the AWS API.

To rotate access keys without interrupting your applications (AWS API)

1. While the first access key is still active, create a second access key, which is active by default. Call the following operation:
 - [CreateAccessKey](#)At this point, the user has two active access keys.
2. Update all applications and tools to use the new access key.
3. Determine whether the first access key is still in use by calling this operation:
 - [GetAccessKeyLastUsed](#)One approach is to wait several days and then check the old access key for any use before proceeding.
4. Even if step [Step 3](#) indicates no use of the old key, we recommend that you do not immediately delete the first access key. Instead, change the state of the first access key to `Inactive` calling this operation:
 - [UpdateAccessKey](#)
5. Use only the new access key to confirm that your applications are working. Any applications and tools that still use the original access key will stop working at this point because they no longer have access to AWS resources. If you find such an application or tool, you can switch its state back to `Active` to reenable the first access key. Then return to step [Step 2](#) and update this application to use the new key.
6. After you wait some period of time to ensure that all applications and tools have been updated, you can delete the first access key calling this operation:
 - [DeleteAccessKey](#)

For more information, see the following:

- [How to Rotate Access Keys for IAM Users](#). This entry on the *AWS Security Blog* provides more information on key rotation.
- [Security best practices in IAM \(p. 527\)](#). This page provides general recommendations for helping to secure your AWS resources.

Auditing access keys

You can review the AWS access keys in your code to determine whether the keys are from an account that you own. You can pass an access key ID using the [aws sts get-access-key-info](#) AWS CLI command or the [GetAccessKeyInfo](#) AWS API operation.

The AWS CLI and AWS API operations return the ID of the AWS account to which the access key belongs. Access key IDs beginning with `AKIA` are long-term credentials for an IAM user or an AWS account root user. Access key IDs beginning with `ASIA` are temporary credentials that are created using AWS STS operations. If the account in the response belongs to you, you can sign in as the root user and review

your root user access keys. Then, you can pull a [credentials report \(p. 148\)](#) to learn which IAM user owns the keys. To learn who requested the temporary credentials for an ASIA access key, view the AWS STS events in your CloudTrail logs.

For security purposes, you can [review AWS CloudTrail logs \(p. 340\)](#) to learn who used the temporary credentials to perform an action in AWS. You can use the `aws:RoleSessionName` condition key in the role trust policy to require users to specify a session name when they assume a role. For example, you can require that IAM users specify their own user name as their session name. For more information, see [aws:RoleSessionName \(p. 718\)](#).

This operation does not indicate the state of the access key. The key might be active, inactive, or deleted. Active keys might not have permissions to perform an operation. Providing a deleted access key might return an error that the key doesn't exist.

Resetting lost or forgotten passwords or access keys for AWS

Having trouble signing in to AWS? Make sure that you're on the correct [AWS sign-in page \(p. 64\)](#) for your type of user. If you are the AWS account root user (account owner), you can sign in to AWS using the credentials that you set up when you created the AWS account. If you are an IAM user, your account administrator can give you the credentials that you can use to sign in to AWS. If you need to request support, do not use the feedback link on this page, as the form is received by the AWS Documentation team, not AWS Support. Instead, on the [Contact Us](#) page, expand **I cannot login to my AWS Account** and choose **Request Support for AWS Account Credentials**.

On the main sign-in page, you must enter your email address to sign in as the root user, or enter your account ID to sign in as an IAM user. You can provide your password only on the sign-in page that matches your user type. For more information, see [Signing in to the AWS Management Console as an IAM user or root user \(p. 64\)](#).

If you are on the correct sign-in page and lose or forget your passwords or access keys, you *cannot* retrieve them from IAM. Instead, you can reset them using the following methods:

- **AWS account root user password** – If you forget your root user password, you can reset the password from the AWS Management Console. For details, see [the section called "Resetting a lost or forgotten root user password" \(p. 111\)](#) later in this topic.
- **AWS account access keys** – If you forget your account access keys, you can create new access keys without disabling the existing access keys. If you are not using the existing keys, you can delete those. For details, see [Creating access keys for the root user \(p. 335\)](#) and [Deleting access keys for the root user \(p. 335\)](#).
- **IAM user password** – If you are an IAM user and you forget your password, you must ask your administrator to reset your password. To learn how an administrator can manage your password, see [Managing passwords for IAM users \(p. 96\)](#).
- **IAM user access keys** – If you are an IAM user and you forget your access keys, you will need new access keys. If you have permission to create your own access keys, you can find instructions for creating a new one at [Managing access keys \(console\) \(p. 104\)](#). If you do not have the required permissions, you must ask your administrator to create new access keys. If you are still using your old keys, ask your administrator not to delete the old keys. To learn how an administrator can manage your access keys, see [Managing access keys for IAM users \(p. 102\)](#).

You should follow the AWS [best practice \(p. 532\)](#) of periodically changing your password and AWS access keys. In AWS, you change access keys by *rotating* them. This means that you create a new one, configure your applications to use the new key, and then delete the old one. You are allowed to have two access key pairs active at the same time for just this reason. For more information, see [Rotating access keys \(p. 106\)](#).

Resetting a lost or forgotten root user password

When you first created your AWS account, you provided an email address and password. These are your AWS account root user credentials. If you forget your root user password, you can reset the password from the AWS Management Console.

To reset your root user password:

1. Use your AWS account email address to begin signing in to the [AWS Management Console](#) as the root user and then choose **Next**.

Note

If you are signed in to the [AWS Management Console](#) with *IAM user* credentials, then you must sign out before you can reset the root user password. If you see the account-specific IAM user sign-in page, choose **Sign-in using root account credentials** near the bottom of the page. If necessary, provide your account email address and choose **Next** to access the [Root user sign in](#) page.

2. Choose **Forgot your password?**.
3. Provide the email address that is associated with the account. Then provide the CAPTCHA text and choose **Continue**.
4. Check the email that is associated with your AWS account for a message from Amazon Web Services. The email will come from an address ending in @amazon.com or @aws.amazon.com. Follow the directions in the email. If you don't see the email in your account, check your spam folder. If you no longer have access to the email, see [I don't have access to the email for my AWS account \(p. 70\)](#).

Using multi-factor authentication (MFA) in AWS



For increased security, we recommend that you configure multi-factor authentication (MFA) to help protect your AWS resources. You can enable MFA for IAM users or the AWS account root user. When you enable MFA for the root user, it affects only the root user credentials. IAM users in the account are distinct identities with their own credentials, and each identity has its own MFA configuration.

Topics

- [What is MFA? \(p. 111\)](#)
- [Enabling MFA devices for users in AWS \(p. 112\)](#)
- [Checking MFA status \(p. 129\)](#)
- [Resynchronizing virtual and hardware MFA devices \(p. 130\)](#)
- [Deactivating MFA devices \(p. 134\)](#)
- [What if an MFA device is lost or stops working? \(p. 136\)](#)
- [Configuring MFA-protected API access \(p. 138\)](#)
- [Sample code: Requesting credentials with multi-factor authentication \(p. 143\)](#)

What is MFA?

MFA adds extra security because it requires users to provide unique authentication from an AWS supported MFA mechanism in addition to their regular sign-in credentials when they access AWS websites or services:

- **Virtual MFA devices.** A software app that runs on a phone or other device and emulates a physical device. The device generates a six-digit numeric code based upon a time-synchronized one-time

password algorithm. The user must type a valid code from the device on a second webpage during sign-in. Each virtual MFA device assigned to a user must be unique. A user cannot type a code from another user's virtual MFA device to authenticate. Because they can run on unsecured mobile devices, virtual MFA might not provide the same level of security as U2F devices or hardware MFA devices. We do recommend that you use a virtual MFA device while waiting for hardware purchase approval or while you wait for your hardware to arrive. For a list of a few supported apps that you can use as virtual MFA devices, see [Multi-Factor Authentication](#). For instructions on setting up a virtual MFA device with AWS, see [Enabling a virtual multi-factor authentication \(MFA\) device \(console\) \(p. 114\)](#).

- **U2F security key.** A device that you plug into a USB port on your computer. U2F is an open authentication standard hosted by the [FIDO Alliance](#). When you enable a U2F security key, you sign in by entering your credentials and then tapping the device instead of manually entering a code. For information on supported AWS U2F security keys, see [Multi-Factor Authentication](#). For instructions on setting up a U2F security key with AWS, see [Enabling a U2F security key \(console\) \(p. 117\)](#).
- **Hardware MFA device.** A hardware device that generates a six-digit numeric code based upon a time-synchronized one-time password algorithm. The user must type a valid code from the device on a second webpage during sign-in. Each MFA device assigned to a user must be unique. A user cannot type a code from another user's device to be authenticated. For information on supported hardware MFA devices, see [Multi-Factor Authentication](#). For instructions on setting up a hardware MFA device with AWS, see [Enabling a hardware MFA device \(console\) \(p. 122\)](#).
- **SMS text message-based MFA.** A type of MFA in which the IAM user settings include the phone number of the user's SMS-compatible mobile device. When the user signs in, AWS sends a six-digit numeric code by SMS text message to the user's mobile device. The user is required to type that code on a second webpage during sign-in. Note that SMS-based MFA is available only for IAM users. You cannot use this type of MFA with the AWS account root user. For more information about enabling SMS text messaging-based MFA, see [PREVIEW – Enabling SMS text message MFA devices \(p. 127\)](#).

Note

AWS will soon end support for SMS multi-factor authentication (MFA). We are not allowing new customers to preview this feature. We recommend that existing customers switch to one of the following alternative methods of MFA: [virtual \(software-based\) MFA device \(p. 114\)](#), [U2F security key \(p. 117\)](#), or [hardware MFA device \(p. 122\)](#). You can view users in your account with an assigned SMS MFA device. To do so, go to the IAM console, choose **Users** from the navigation pane, and look for users with **SMS** in the **MFA** column of the table.

For answers to commonly asked questions about AWS MFA, go to the [AWS Multi-Factor Authentication FAQs](#).

Enabling MFA devices for users in AWS

The steps for configuring MFA depend on the type of MFA device you are using.

Topics

- [General steps for enabling MFA devices \(p. 112\)](#)
- [Enabling a virtual multi-factor authentication \(MFA\) device \(console\) \(p. 114\)](#)
- [Enabling a U2F security key \(console\) \(p. 117\)](#)
- [Enabling a hardware MFA device \(console\) \(p. 122\)](#)
- [PREVIEW – Enabling SMS text message MFA devices \(p. 127\)](#)
- [Enabling and managing virtual MFA devices \(AWS CLI or AWS API\) \(p. 128\)](#)

General steps for enabling MFA devices

The following overview procedure describes how to set up and use MFA and provides links to related information.

1. *Get an MFA device such as one of the following.* You can enable only one MFA device per AWS account root user or IAM user.
 - A virtual MFA device, which is a software app that is compliant with [RFC 6238, a standards-based TOTP \(time-based one-time password\) algorithm](#). You can install the app on a phone or other device. For a list of a few supported apps that you can use as virtual MFA devices, see [Multi-Factor Authentication](#).
 - A U2F security key with an [AWS supported configuration \(p. 121\)](#), such as one of the U2F devices discussed on the [Multi-Factor Authentication](#) page.
 - A hardware-based MFA device, such as one of the AWS supported hardware token devices discussed on the [Multi-Factor Authentication](#) page.
 - A mobile phone that can receive standard short message service (SMS) text messages.

Notes

- If you choose to use SMS-based MFA, text messaging charges from your mobile device's carrier may apply.
- SMS-based MFA is only available for IAM users and cannot be used for the root user.

2. *Enable the MFA device.*

- IAM users with virtual or hardware MFA devices: Enable from the AWS Management Console, AWS CLI, or the IAM API.
- IAM users with U2F security keys or a mobile phone that can receive SMS text messages: Enable from the AWS Management Console only.
- AWS account root users with any type of MFA device (except SMS MFA, which is not supported for root users): Enable from the AWS Management Console only.

For information about enabling each type of MFA device, see the following pages:

- Virtual MFA device: [Enabling a virtual multi-factor authentication \(MFA\) device \(console\) \(p. 114\)](#)
- U2F security key: [Enabling a U2F security key \(console\) \(p. 117\)](#)
- Hardware MFA device: [Enabling a hardware MFA device \(console\) \(p. 122\)](#)
- SMS MFA device: [PREVIEW – Enabling SMS text message MFA devices \(p. 127\)](#)

3. *Use the MFA device when you log in to or access AWS resources.* Note the following:

- U2F security keys: To access an AWS website, enter your credentials and then tap the U2F security key when prompted.
- Virtual MFA devices, hardware MFA devices, and SMS MFA devices: To access an AWS website, you need an MFA code from the device in addition to your user name and password. If AWS determines that the IAM user you sign in as is MFA-enabled with SMS, then it automatically sends the MFA code to the configured phone number.

To access MFA-protected API operations, you need the following:

- An MFA code
- The identifier for the MFA device (the device serial number of a physical device or the ARN of a virtual or SMS device defined in AWS)
- The usual access key ID and secret access key

Notes

- You cannot pass the MFA information for a U2F security key or SMS MFA device to AWS STS API operations to request temporary credentials.
- You cannot use AWS CLI commands or AWS API operations to enable [U2F security keys \(p. 117\)](#).

For more information, see [Using MFA devices with your IAM sign-in page \(p. 83\)](#).

Enabling a virtual multi-factor authentication (MFA) device (console)

You can use a phone or other device as a virtual multi-factor authentication (MFA) device. To do this, install a mobile app that is compliant with [RFC 6238, a standards-based TOTP \(time-based one-time password\) algorithm](#). These apps generate a six-digit authentication code. Because they can run on unsecured mobile devices, virtual MFA might not provide the same level of security as U2F devices or hardware MFA devices. We do recommend that you use a virtual MFA device while waiting for hardware purchase approval or while you wait for your hardware to arrive.

Most virtual MFA apps support creating multiple virtual devices, allowing you to use the same app for multiple AWS accounts or users. However, you can enable only one MFA device per user.

For a list of virtual MFA apps that you can use, see [Multi-Factor Authentication](#). Note that AWS requires a virtual MFA app that produces a six-digit OTP.

Topics

- [Permissions required \(p. 114\)](#)
- [Enable a virtual MFA device for an IAM user \(console\) \(p. 114\)](#)
- [Enable a virtual MFA device for your AWS account root user \(console\) \(p. 115\)](#)
- [Replace or "rotate" a virtual MFA device \(p. 117\)](#)

Permissions required

To manage virtual MFA devices for your IAM user, you must have the permissions from the following policy: [AWS: Allows MFA-authenticated IAM users to manage their own MFA device on the My Security Credentials page \(p. 399\)](#).

Enable a virtual MFA device for an IAM user (console)

You can use IAM in the AWS Management Console to enable and manage a virtual MFA device for an IAM user in your account. To enable and manage an MFA device using the AWS CLI or AWS API, see [Enabling and managing virtual MFA devices \(AWS CLI or AWS API\) \(p. 128\)](#).

Note

You must have physical access to the hardware that will host the user's virtual MFA device in order to configure MFA. For example, you might configure MFA for a user who will use a virtual MFA device running on a smartphone. In that case, you must have the smartphone available in order to finish the wizard. Because of this, you might want to let users configure and manage their own virtual MFA devices. In that case, you must grant users the permissions to perform the necessary IAM actions. For more information and for an example of an IAM policy that grants these permissions, see [AWS: Allows MFA-authenticated IAM users to manage their own MFA device on the My Security Credentials page \(p. 399\)](#).

To enable a virtual MFA device for an IAM user (console)

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, choose **Users**.
3. In the **User Name** list, choose the name of the intended MFA user.
4. Choose the **Security credentials** tab. Next to **Assigned MFA device**, choose **Manage**.
5. In the **Manage MFA Device** wizard, choose **Virtual MFA device**, and then choose **Continue**.

IAM generates and displays configuration information for the virtual MFA device, including a QR code graphic. The graphic is a representation of the "secret configuration key" that is available for manual entry on devices that do not support QR codes.

6. Open your virtual MFA app. For a list of apps that you can use for hosting virtual MFA devices, see [Multi-Factor Authentication](#).

If the virtual MFA app supports multiple virtual MFA devices or accounts, choose the option to create a new virtual MFA device or account.

7. Determine whether the MFA app supports QR codes, and then do one of the following:

- From the wizard, choose **Show QR code**, and then use the app to scan the QR code. For example, you might choose the camera icon or choose an option similar to **Scan code**, and then use the device's camera to scan the code.
- In the **Manage MFA Device** wizard, choose **Show secret key**, and then type the secret key into your MFA app.

When you are finished, the virtual MFA device starts generating one-time passwords.

8. In the **Manage MFA Device** wizard, in the **MFA code 1** box, type the one-time password that currently appears in the virtual MFA device. Wait up to 30 seconds for the device to generate a new one-time password. Then type the second one-time password into the **MFA code 2** box. Choose **Assign MFA**.

Important

Submit your request immediately after generating the codes. If you generate the codes and then wait too long to submit the request, the MFA device successfully associates with the user but the MFA device is out of sync. This happens because time-based one-time passwords (TOTP) expire after a short period of time. If this happens, you can [resync the device \(p. 130\)](#).

The virtual MFA device is now ready for use with AWS. For information about using MFA with the AWS Management Console, see [Using MFA devices with your IAM sign-in page \(p. 83\)](#).

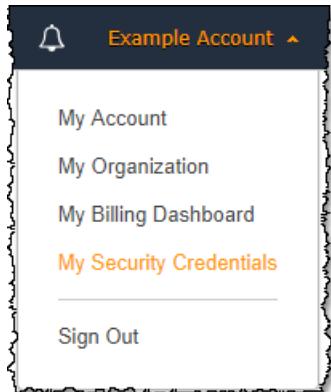
[Enable a virtual MFA device for your AWS account root user \(console\)](#)

You can use the AWS Management Console to configure and enable a virtual MFA device for your root user. To enable MFA devices for the AWS account, you must be signed in to AWS using your root user credentials.

Before you enable MFA for your root user, review your account settings and contact information to make sure that you have access to the email and phone number. If your MFA device is lost, stolen, or not working, you can still sign in as the root user by verifying your identity using that email and phone number. To learn about signing in using these alternative factors of authentication, see [What if an MFA device is lost or stops working? \(p. 136\)](#).

[To configure and enable a virtual MFA device for use with your root user \(console\)](#)

1. Sign in to the AWS Management Console.
2. On the right side of the navigation bar, choose your account name, and choose **My Security Credentials**. If necessary, choose **Continue to Security Credentials**. Then expand the **Multi-Factor Authentication (MFA)** section on the page.



3. Choose **Activate MFA**.
4. In the wizard, choose **Virtual MFA device**, and then choose **Continue**.

IAM generates and displays configuration information for the virtual MFA device, including a QR code graphic. The graphic is a representation of the secret configuration key that is available for manual entry on devices that do not support QR codes.

5. Open the virtual MFA app on the device.
If the virtual MFA app supports multiple virtual MFA devices or accounts, choose the option to create a new virtual MFA device or account.
6. The easiest way to configure the app is to use the app to scan the QR code. If you cannot scan the code, you can type the configuration information manually. The QR code and secret configuration key generated by IAM are tied to your AWS account and cannot be used with a different account. They can, however, be reused to configure a new MFA device for your account in case you lose access to the original MFA device.
 - To use the QR code to configure the virtual MFA device, from the wizard, choose **Show QR code**. Then follow the app instructions for scanning the code. For example, you might need to choose the camera icon or choose a command like **Scan account barcode**, and then use the device's camera to scan the QR code.
 - In the **Manage MFA Device** wizard, choose **Show secret key**, and then type the secret key into your MFA app.

Important

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

The device starts generating six-digit numbers.

7. In the **Manage MFA Device** wizard, in the **MFA Code 1** box, enter the six-digit number that's currently displayed by the MFA device. Wait up to 30 seconds for the device to generate a new number, and then type the new six-digit number into the **MFA Code 2** box.

Important

Submit your request immediately after generating the codes. If you generate the codes and then wait too long to submit the request, the MFA device successfully associates with the user but the MFA device is out of sync. This happens because time-based one-time passwords (TOTP) expire after a short period of time. If this happens, you can [resync the device \(p. 130\)](#).

8. Choose **Assign MFA**, and then choose **Finish**.

The device is ready for use with AWS. For information about using MFA with the AWS Management Console, see [Using MFA devices with your IAM sign-in page \(p. 83\)](#).

Replace or "rotate" a virtual MFA device

You can have only one MFA device assigned to a user at a time. If the user loses a device or needs to replace it for any reason, you must first deactivate the old device. Then you can add the new device for the user.

- To deactivate the device currently associated with another IAM user, see [Deactivating MFA devices \(p. 134\)](#).
- To add a replacement virtual MFA device for another IAM user, follow the steps in the procedure [Enable a virtual MFA device for an IAM user \(console\) \(p. 114\)](#) above.
- To add a replacement virtual MFA device for the AWS account root user, follow the steps in the procedure [Enable a virtual MFA device for your AWS account root user \(console\) \(p. 115\)](#) earlier in this topic.

Enabling a U2F security key (console)

Universal 2nd Factor (U2F) security keys are a type of [MFA device \(p. 111\)](#) that you can use to protect your AWS resources. You plug your U2F security key into a USB port on your computer and enable it using the instructions that follow. After you enable it, you tap it when prompted to securely complete the sign-in process. If you already use a U2F security key with other services, and it has an [AWS supported configuration \(p. 121\)](#) (for example, the Yubikey 4 or 5 from Yubico), you can also use it with AWS. Otherwise, you need to purchase a U2F security key if you want to use U2F for MFA in AWS. For specifications and purchase information, see [Multi-Factor Authentication](#).

U2F is an open authentication standard hosted by the [FIDO Alliance](#). When you enable a U2F key in AWS, the U2F security key creates a new key pair for use with only AWS. First, you enter your credentials. When prompted, you tap the U2F security key, which responds to the authentication challenge issued by AWS. To learn more about the U2F standard, see [Universal 2nd Factor](#).

You can enable **one** MFA device (of any kind) per root user or IAM user.

Topics

- [Permissions required \(p. 117\)](#)
- [Enable a U2F security key for your own IAM user \(console\) \(p. 118\)](#)
- [Enable a U2F security key for another IAM user \(console\) \(p. 119\)](#)
- [Enable a U2F security key for the AWS account root user \(console\) \(p. 120\)](#)
- [Replace a U2F security key \(p. 121\)](#)
- [Supported configurations for using U2F security keys \(p. 121\)](#)

Permissions required

To manage a U2F security key for your own IAM user while protecting sensitive MFA-related actions, you must have the permissions from the following policy:

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

```
{  
    "Sid": "AllowManageOwnUserMFA",  
    "Effect": "Allow",  
    "Action": [  
        "iam:DeactivateMFADevice",  
        "iam:EnableMFADevice",  
        "iam:GetUser",  
        "iam>ListMFADevices",  
        "iam:ResyncMFADevice"  
    ],  
    "Resource": "arn:aws:iam::*:user/${aws:username}"  
},  
{  
    "Sid": "DenyAllExceptListedIfNoMFA",  
    "Effect": "Deny",  
    "NotAction": [  
        "iam:EnableMFADevice",  
        "iam:GetUser",  
        "iam>ListMFADevices",  
        "iam:ResyncMFADevice"  
    ],  
    "Resource": "arn:aws:iam::*:user/${aws:username}",  
    "Condition": {  
        "BoolIfExists": {  
            "aws:MultiFactorAuthPresent": "false"  
        }  
    }  
}  
]  
}
```

Enable a U2F security key for your own IAM user (console)

You can enable a U2F security key for your own IAM user from the AWS Management Console only, not from the AWS CLI or AWS API.

Note

Before you can enable a U2F security key, you must have physical access to the device.

To enable a U2F security key for your own IAM user (console)

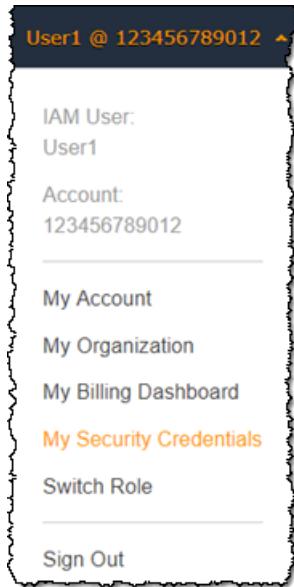
1. Use your AWS account ID or account alias, your IAM user name, and your password to sign in to the [IAM console](#).

Note

For your convenience, the AWS sign-in page uses a browser cookie to remember your IAM user name and account information. If you previously signed in as a different user, choose **Sign in to a different account** near the bottom of the page to return to the main sign-in page. From there, you can type your AWS account ID or account alias to be redirected to the IAM user sign-in page for your account.

To get your AWS account ID, contact your administrator.

2. In the navigation bar on the upper right, choose your user name, and then choose **My Security Credentials**.



3. On the **AWS IAM credentials** tab, in the **Multi-factor authentication** section, choose **Manage MFA device**.
4. In the **Manage MFA device** wizard, choose **U2F security key**, and then choose **Continue**.
5. Insert the U2F security key into your computer's USB port.



6. Tap the U2F security key, and then choose **Close** when U2F setup is complete.

The U2F security key is ready for use with AWS. For information about using MFA with the AWS Management Console, see [Using MFA devices with your IAM sign-in page \(p. 83\)](#).

[Enable a U2F security key for another IAM user \(console\)](#)

You can enable a U2F security key for another IAM user from the AWS Management Console only, not from the AWS CLI or AWS API.

To enable a U2F security key for another IAM user (console)

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, choose **Users**.
3. Choose the name of the user for whom you want to enable MFA, and then choose the **Security credentials** tab.
4. Next to **Assigned MFA device**, choose **Manage**.
5. In the **Manage MFA device** wizard, choose **U2F security key**, and then choose **Continue**.
6. Insert the U2F security key into your computer's USB port.



7. Tap the U2F security key, and then choose **Close** when U2F setup is complete.

The U2F security key is ready for use with AWS. For information about using MFA with the AWS Management Console, see [Using MFA devices with your IAM sign-in page \(p. 83\)](#).

[Enable a U2F security key for the AWS account root user \(console\)](#)

You can configure and enable a virtual MFA device for your root user from the AWS Management Console only, not from the AWS CLI or AWS API.

If your U2F security key is lost, stolen, or not working, you can still sign in using alternative factors of authentication. To learn about signing in using alternative factors of authentication, see [What if an MFA device is lost or stops working? \(p. 136\)](#). To disable this feature, contact [AWS Support](#).

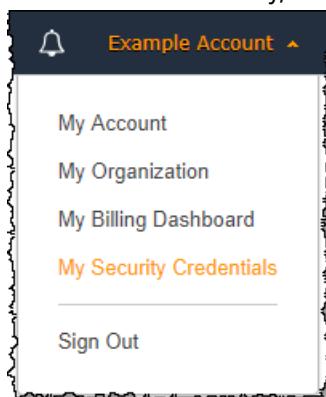
To enable the U2F key for your root user (console)

1. Sign in to the [IAM console](#) as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

Note

If you see three text boxes, then you previously signed in to the console with [IAM user](#) credentials. Your browser might remember this preference and open this account-specific sign-in page every time that you try to sign in. You cannot use the IAM user sign-in page to sign in as the account owner. If you see the [IAM user sign-in page](#), choose **Sign in using root user email** near the bottom of the page. This returns you to the main sign-in page. From there, you can sign in as the root user using your AWS account email address and password.

2. On the right side of the navigation bar, choose on your account name, and then choose **My Security Credentials**. If necessary, choose **Continue to Security Credentials**.



3. Expand the **Multi-factor authentication (MFA)** section.
4. Choose **Manage MFA** or **Activate MFA**, depending on which option you chose in the preceding step.
5. In the wizard, choose **U2F security key** and then choose **Continue**.
6. Insert the U2F security key into your computer's USB port.



7. Tap the U2F security key, and then choose **Close** when U2F setup is complete.

The U2F security key is ready for use with AWS. The next time you use your root user credentials to sign in, you must tap your U2F security key to complete the sign-in process.

[Replace a U2F security key](#)

You can have only one MFA device (virtual, U2F security key, or hardware) assigned to a user at a time. If the user loses a U2F key or needs to replace it for any reason, you must first deactivate the old U2F key. Then you can add a new MFA device for the user.

- To deactivate the device currently associated with a user, see [Deactivating MFA devices \(p. 134\)](#).
- To add a new U2F security for an IAM user, see [Enabling a U2F security key \(console\) \(p. 117\)](#).

If you don't have access to a new U2F security key, you can enable a new virtual MFA device or hardware MFA device. See one of the following for instructions:

- [Enabling a virtual multi-factor authentication \(MFA\) device \(console\) \(p. 114\)](#)
- [Enabling a hardware MFA device \(console\) \(p. 122\)](#)

[Supported configurations for using U2F security keys](#)

You can use U2F as a multi-factor authentication (MFA) method in AWS using currently supported configurations. These include U2F devices supported by AWS and browsers that support U2F.

[U2F devices supported by AWS](#)

AWS currently supports U2F-compliant security devices that plug into USB ports on your computer.

Note

AWS requires access to the physical USB port on your computer to verify your U2F device. U2F MFA will not work with a virtual machine or remote connection.

For information on purchasing a supported device, see [Multi-Factor Authentication](#).

[Browsers that support U2F](#)

The following browsers currently support the use of U2F security keys:

- Google Chrome, version 38 and later.
- Opera, version 40 and later.
- Mozilla Firefox, version 57 and later.

Note

Most Firefox versions that currently support U2F do not enable support by default.

For instructions on enabling U2F support in Firefox, see [Troubleshooting U2F security keys \(p. 584\)](#).

Browser plugins

AWS currently supports only browsers that natively support the U2F standard. AWS does not support using plugins to add U2F browser support. Also note that some browser plugins are incompatible with the U2F standard and can cause unexpected results with U2F security keys.

For information on disabling browser plugins and other troubleshooting tips, see [I can't enable my U2F security key \(p. 584\)](#).

Mobile environments

AWS does not currently support the use of U2F security keys with mobile browsers or non-USB U2F devices.

The AWS Console Mobile App does not currently support using U2F security keys for MFA.

AWS CLI and AWS API

AWS currently supports using U2F security keys only in the AWS Management Console. Using U2F security keys for MFA is not currently supported in the [AWS CLI](#) and [AWS API](#), or for access to [MFA-protected API operations \(p. 138\)](#).

Additional resources

- For more information on using U2F security keys in AWS, see [Enabling a U2F security key \(console\) \(p. 117\)](#).
- For help with troubleshooting U2F in AWS, see [Troubleshooting U2F security keys \(p. 584\)](#).
- For general industry information on U2F support, see [Universal 2nd Factor](#).

Enabling a hardware MFA device (console)

A hardware MFA device generates a six-digit numeric code based upon a time-synchronized one-time password algorithm. The user must type a valid code from the device when prompted during the sign-in process. Each MFA device assigned to a user must be unique; a user cannot type a code from another user's device to be authenticated.

Hardware MFA devices and [U2F security keys \(p. 117\)](#) are both physical devices that you purchase. The difference is that hardware MFA devices generate a code that you view and then enter when prompted when signing it to AWS. With a U2F security key, you don't see or type an authentication code. Instead, the U2F security key generates a response without presenting it to the user and the service validates it. For specifications and purchase information for both device types, see [Multi-Factor Authentication](#).

You can enable a hardware MFA device for an IAM user from the AWS Management Console, the command line, or the IAM API. To enable an MFA device for your AWS account root user, see [Enable a hardware MFA device for the AWS account root user \(console\) \(p. 125\)](#).

You can enable **one** MFA device (of any kind) per root user or IAM user.

Note

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

Topics

- [Permissions required \(p. 123\)](#)
- [Enable a hardware MFA device for your own IAM user \(console\) \(p. 123\)](#)
- [Enable a hardware MFA device for another IAM user \(console\) \(p. 124\)](#)

- [Enable a hardware MFA device for the AWS account root user \(console\) \(p. 125\)](#)
- [Replace or "rotate" a physical MFA device \(p. 126\)](#)

Permissions required

To manage a hardware MFA device for your own IAM user while protecting sensitive MFA-related actions, you must have the permissions from the following policy:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowManageOwnUserMFA",  
            "Effect": "Allow",  
            "Action": [  
                "iam:DeactivateMFADevice",  
                "iam:EnableMFADevice",  
                "iam:GetUser",  
                "iam>ListMFADevices",  
                "iam:ResyncMFADevice"  
            ],  
            "Resource": "arn:aws:iam::*:user/${aws:username}"  
        },  
        {  
            "Sid": "DenyAllExceptListedIfNoMFA",  
            "Effect": "Deny",  
            "NotAction": [  
                "iam:EnableMFADevice",  
                "iam:GetUser",  
                "iam>ListMFADevices",  
                "iam:ResyncMFADevice"  
            ],  
            "Resource": "arn:aws:iam::*:user/${aws:username}",  
            "Condition": {  
                "BoolIfExists": {  
                    "aws:MultiFactorAuthPresent": "false"  
                }  
            }  
        }  
    ]  
}
```

Enable a hardware MFA device for your own IAM user (console)

You can enable your own hardware MFA device from the AWS Management Console.

Note

Before you can enable a hardware MFA device, you must have physical access to the device.

To enable a hardware MFA device for your own IAM user (console)

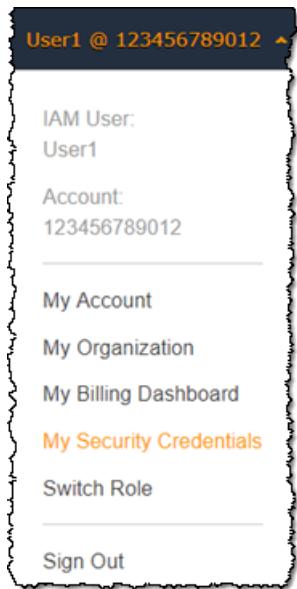
1. Use your AWS account ID or account alias, your IAM user name, and your password to sign in to the [IAM console](#).

Note

For your convenience, the AWS sign-in page uses a browser cookie to remember your IAM user name and account information. If you previously signed in as a different user, choose **Sign in to a different account** near the bottom of the page to return to the main sign-in page. From there, you can type your AWS account ID or account alias to be redirected to the IAM user sign-in page for your account.

To get your AWS account ID, contact your administrator.

2. In the navigation bar on the upper right, choose your user name, and then choose **My Security Credentials**.



3. On the **AWS IAM credentials** tab, in the **Multi-factor authentication** section, choose **Manage MFA device**.
4. In the **Manage MFA device** wizard, choose **Hardware MFA device** and then choose **Continue**.
5. Type the device serial number. The serial number is usually on the back of the device.
6. In the **MFA 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 the code, and then type the next six-digit number into the **MFA code 2** box. You might need to press the button on the front of the device again to display the second number.
8. Choose **Assign MFA**.

Important

Submit your request immediately after generating the authentication codes. If you generate the codes and then wait too long to submit the request, the MFA device successfully associates with the user but the MFA device becomes out of sync. This happens because time-based one-time passwords (TOTP) expire after a short period of time. If this happens, you can [resync the device \(p. 130\)](#).

The device is ready for use with AWS. For information about using MFA with the AWS Management Console, see [Using MFA devices with your IAM sign-in page \(p. 83\)](#).

Enable a hardware MFA device for another IAM user (console)

You can enable a hardware MFA device for another IAM user from the AWS Management Console.

To enable a hardware MFA device for another IAM user (console)

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, choose **Users**.
3. Choose the name of the user for whom you want to enable MFA, and then choose the **Security credentials** tab.
4. Next to **Assigned MFA device**, choose **Manage**.
5. In the **Manage MFA device** wizard, choose **Hardware MFA device** and then choose **Continue**.
6. Type the device serial number. The serial number is usually on the back of the device.
7. In the **MFA code 1** box, type the six-digit number displayed by the MFA device. You might need to press the button on the front of the device to display the number.



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

Important

Submit your request immediately after generating the authentication codes. If you generate the codes and then wait too long to submit the request, the MFA device successfully associates with the user but the MFA device becomes out of sync. This happens because time-based one-time passwords (TOTP) expire after a short period of time. If this happens, you can [resync the device](#) (p. 130).

The device is ready for use with AWS. For information about using MFA with the AWS Management Console, see [Using MFA devices with your IAM sign-in page](#) (p. 83).

[Enable a hardware MFA device for the AWS account root user \(console\)](#)

You can configure and enable a virtual MFA device for your root user from the AWS Management Console only, not from the AWS CLI or AWS API.

If your MFA device is lost, stolen, or not working, you can still sign in using alternative factors of authentication. If you can't sign in with your MFA device, you can sign in by verifying your identity using the email and phone that are registered with your account. Before you enable MFA for your root user, review your account settings and contact information to make sure that you have access to the email and phone number. To learn about signing in using alternative factors of authentication, see [What if an MFA device is lost or stops working?](#) (p. 136). To disable this feature, contact [AWS Support](#).

Note

You might see different text, such as **Sign in using MFA** and **Troubleshoot your authentication device**. However, the same features are provided. In either case, if you cannot verify your account email address and phone number using alternative factors of authentication, contact [AWS Support](#) to deactivate your MFA setting.

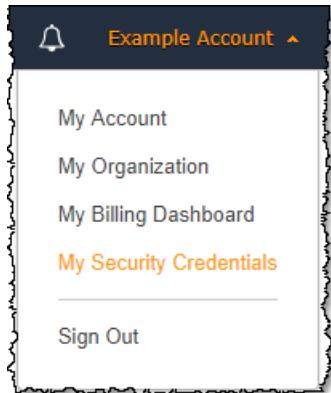
[To enable the MFA device for your root user \(console\)](#)

1. Sign in to the [IAM console](#) as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

Note

If you see three text boxes, then you previously signed in to the console with [IAM user](#) credentials. Your browser might remember this preference and open this account-specific sign-in page every time that you try to sign in. You cannot use the IAM user sign-in page to sign in as the account owner. If you see the [IAM user sign-in page](#), choose **Sign in using root user email** near the bottom of the page. This returns you to the main sign-in page. From there, you can sign in as the root user using your AWS account email address and password.

2. On the right side of the navigation bar, choose on your account name, and then choose **My Security Credentials**. If necessary, choose **Continue to Security Credentials**.



3. Expand the **Multi-factor authentication (MFA)** section.
4. Choose **Manage MFA** or **Activate MFA**, depending on which option you chose in the preceding step.
5. In the wizard, choose **Hardware MFA device** and then choose **Continue**.
6. In the **Serial number** box, type the serial number that is found on the back of the MFA device.
7. In the **MFA code 1** box, type the six-digit number displayed by the MFA device. You might need to press the button on the front of the device to display the number.



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

Important

Submit your request immediately after generating the authentication codes. If you generate the codes and then wait too long to submit the request, the MFA device successfully associates with the user but the MFA device becomes out of sync. This happens because time-based one-time passwords (TOTP) expire after a short period of time. If this happens, you can [resync the device \(p. 130\)](#).

The next time you use your root user credentials to sign in, you must type a code from the MFA device.

Replace or "rotate" a physical MFA device

You can have only one MFA device assigned to a user at a time. If the user loses a device or needs to replace it for any reason, you must first deactivate the old device. Then you can add the new device for the user.

- To deactivate the device currently associated with a user, see [Deactivating MFA devices \(p. 134\)](#).
- To add a replacement hardware MFA device for an IAM user, follow the steps in the procedure [Enable a hardware MFA device for another IAM user \(console\) \(p. 124\)](#) earlier in this topic.
- To add a replacement virtual MFA device for the AWS account root user, follow the steps in the procedure [Enable a hardware MFA device for the AWS account root user \(console\) \(p. 125\)](#) earlier in this topic.

PREVIEW – Enabling SMS text message MFA devices

AWS will soon end support for SMS multi-factor authentication (MFA). We are not allowing new customers to preview this feature. We recommend that existing customers switch to one of the following alternative methods of MFA:

- A virtual (software-based) (p. 114) MFA device
- A U2F security key (p. 117)
- A hardware-based (p. 122) MFA device

Tip

You can view users in your account with an assigned SMS MFA device. In the IAM console, choose **Users** from the navigation pane, and look for users with **SMS** in the **MFA** column of the table.

An SMS (short message service) MFA device can be any mobile device with a phone number that can receive standard [SMS text messages](#). When an MFA code is needed, AWS sends it to the phone number that is configured for the IAM user.

Note

SMS MFA can be used only with IAM users. It cannot be used with the AWS account root user. To protect the root user with MFA, you must use a virtual MFA device, U2F security key, or hardware MFA device.

Enable an SMS MFA device for an IAM user (console)

You can use IAM in the AWS Management Console to configure an IAM user with a phone number to enable SMS MFA.

Note

Currently, you can manage SMS MFA only in the AWS Management Console.

To enable SMS MFA for an IAM user (console)

1. Use your AWS account ID or account alias, your IAM user name, and your password to sign in to the [IAM console](#).
Note
For your convenience, the AWS sign-in page uses a browser cookie to remember your IAM user name and account information. If you previously signed in as a different user, choose **Sign in to a different account** near the bottom of the page to return to the main sign-in page. From there, you can type your AWS account ID or account alias to be redirected to the IAM user sign-in page for your account.
2. In the navigation pane, choose **Users**.
3. In the **User Name** list, choose the name (not the check box) of the intended MFA user.
4. Choose the **Security credentials** tab. Next to **Assigned MFA device**, choose **Manage**.
5. In the **Manage MFA Device** wizard, choose **An SMS MFA device**, and then choose **Continue**.
6. Type the phone number to which you want to send MFA codes for this IAM user, and then choose **Continue**.
7. A six-digit authentication code is immediately sent to the specified phone number for verification. Type the six-digit code and then choose **Continue**. If the code does not arrive in a reasonable amount of time, choose **Resend Code**. Note that SMS is not a service with a guaranteed delivery time.
8. If AWS successfully verifies the code, the wizard ends. Otherwise, choose **Finish** to close the wizard.

Change the phone number for SMS MFA for an IAM user

To change the phone number of the SMS MFA device assigned to an IAM user, you must delete the current MFA device. Then create a new device with the new phone number. To learn how to delete a device, see [Deactivating MFA devices \(p. 134\)](#).

Enabling and managing virtual MFA devices (AWS CLI or AWS API)

You can use AWS CLI commands or AWS API operations to enable a virtual MFA device for an IAM user. You cannot enable an MFA device for the AWS account root user with the AWS CLI, AWS API, Tools for Windows PowerShell, or any other command line tool. However, you can use the AWS Management Console to enable an MFA device for the root user.

When you enable an MFA device from the AWS Management Console, the console performs multiple steps for you. If you instead create a virtual device using the AWS CLI, Tools for Windows PowerShell, or AWS API, then you must perform the steps manually and in the correct order. For example, to create a virtual MFA device, you must create the IAM object and extract the code as either a string or a QR code graphic. Then you must sync the device and associate it with an IAM user. See the **Examples** section of [New-IAMVirtualMFADevice](#) for more details. For a physical device, you skip the creation step and go directly to syncing the device and associating it with the user.

To create the virtual device entity in IAM to represent a virtual MFA device

These commands provide an ARN for the device that is used in place of a serial number in many of the following commands.

- AWS CLI: `aws iam create-virtual-mfa-device`
- AWS API: `CreateVirtualMFADevice`

To enable an MFA device for use with AWS

These commands synchronize the device with AWS and associate it with a user or the root user. If the device is virtual, use the ARN of the virtual device as the serial number.

Important

Submit your request immediately after generating the authentication codes. If you generate the codes and then wait too long to submit the request, the MFA device successfully associates with the user but the MFA device becomes out of sync. This happens because time-based one-time passwords (TOTP) expire after a short period of time. If this happens, you can resynchronize the device using the commands described below.

- AWS CLI: `aws iam enable-mfa-device`
- AWS API: `EnableMFADevice`

To deactivate a device

Use these commands to disassociate the device from the user and deactivate it. If the device is virtual, use the ARN of the virtual device as the serial number. You must also separately delete the virtual device entity.

- AWS CLI: `aws iam deactivate-mfa-device`
- AWS API: `DeactivateMFADevice`

To list virtual MFA device entities

Use these commands to list virtual MFA device entities.

- AWS CLI: `aws iam list-virtual-mfa-devices`
- AWS API: `ListVirtualMFADevices`

To resynchronize an MFA device

Use these commands if the device is generating codes that are not accepted by AWS. If the device is virtual, use the ARN of the virtual device as the serial number.

- AWS CLI: `aws iam resync-mfa-device`
- AWS API: `ResyncMFADevice`

To delete a virtual MFA device entity in IAM

After the device is disassociated from the user, you can delete the device entity.

- AWS CLI: `aws iam delete-virtual-mfa-device`
- AWS API: `DeleteVirtualMFADevice`

To recover a virtual MFA device that is lost or not working

Sometimes, an IAM user's device that hosts the virtual MFA app is lost, replaced, or not working. When this happens, the user can't recover it on their own. IAM users must contact an administrator to deactivate the device. For more information, see [What if an MFA device is lost or stops working? \(p. 136\)](#).

Checking MFA status

Use the IAM console to check whether an AWS account root user or IAM user has a valid MFA device enabled.

To check the MFA status of a root user

1. Sign in to the AWS Management Console with your root user credentials and then open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation bar on the upper right, choose your user name, and then choose **My Security Credentials**.
3. Check under **Multi-factor Authentication (MFA)** to see whether MFA is enabled or disabled. If MFA has not been activated, an alert symbol () is displayed.

If you want to enable MFA for the account, see one of the following:

- [Enable a virtual MFA device for your AWS account root user \(console\) \(p. 115\)](#)
- [Enable a U2F security key for the AWS account root user \(console\) \(p. 120\)](#)
- [Enable a hardware MFA device for the AWS account root user \(console\) \(p. 125\)](#)

To check the MFA status of IAM users

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Users**.
3. If necessary, add the **MFA** column to the users table by completing the following steps:

- a. Above the table on the far right, choose the settings icon ().
 - b. In **Manage Columns**, select **MFA**.
 - c. (Optional) Clear the check box for any column headings that you do not want to appear in the users table.
 - d. Choose **Close** to return to the list of users.
4. The **MFA** column tells you about the MFA device that is enabled. If no MFA device is active for the user, the console displays **Not enabled**. If the user has an MFA device enabled, the **MFA** column shows the type of device that is enabled with a value of **Virtual**, **U2F Security Key**, **Hardware**, or **SMS**.
 5. To view additional information about the MFA device for a user, choose the name of the user whose MFA status you want to check. Then choose the **Security credentials** tab.
 6. If no MFA device is active for the user, the console displays **No** next to **Assigned MFA device**. If the user has an MFA device enabled, the **Assigned MFA device** item shows a value for the device:
 - The device serial number of a hardware device (usually the number from the back of the device), such as `GAHT12345678`
 - The ARN in AWS for an SMS device, such as `arn:aws:iam::123456789012:sms-mfa/username`
 - The ARN in AWS for a virtual device, such as `arn:aws:iam::123456789012:mfa/username`

If you want to change the current setting, choose **Manage** next to **Assigned MFA Device**.

For more information on enabling MFA, see the following:

- [Enabling a virtual multi-factor authentication \(MFA\) device \(console\) \(p. 114\)](#)
- [Enabling a U2F security key \(console\) \(p. 117\)](#)
- [Enabling a hardware MFA device \(console\) \(p. 122\)](#)
- [PREVIEW – Enabling SMS text message MFA devices \(p. 127\)](#)

Resynchronizing virtual and hardware MFA devices

You can use AWS to resynchronize your virtual and hardware multi-factor authentication (MFA) devices. If your device is not synchronized when you try to use it, the sign-in attempt fails and IAM prompts you to resynchronize the device.

Note

U2F security keys do not go out of sync. If a U2F security key is lost or broken, you can deactivate it. For instructions on deactivating any MFA device type, see [To deactivate an MFA device for another IAM user \(console\) \(p. 134\)](#).

As an AWS administrator, you can resynchronize your IAM users' virtual and hardware MFA devices if they get out of synchronization.

If your AWS account root user MFA device is not working, you can resynchronize your device using the IAM console with or without completing the sign-in process.

Topics

- [Permissions required \(p. 131\)](#)
- [Resynchronizing virtual and hardware MFA devices \(IAM console\) \(p. 131\)](#)
- [Resynchronizing virtual and hardware MFA devices \(AWS CLI\) \(p. 133\)](#)
- [Resynchronizing virtual and hardware MFA devices \(AWS API\) \(p. 134\)](#)

Permissions required

To resynchronize virtual or hardware MFA devices for your own IAM user, you must have the permissions from the following policy: This policy does not allow you to create or deactivate a device.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowListActions",  
            "Effect": "Allow",  
            "Action": [  
                "iam>ListVirtualMFADevices"  
            ],  
            "Resource": "*"  
        },  
        {  
            "Sid": "AllowUserToViewAndManageTheirOwnUserMFA",  
            "Effect": "Allow",  
            "Action": [  
                "iam>ListMFADevices",  
                "iam>ResyncMFADevice"  
            ],  
            "Resource": "arn:aws:iam::*:user/${aws:username}"  
        },  
        {  
            "Sid": "BlockAllExceptListedIfNoMFA",  
            "Effect": "Deny",  
            "NotAction": [  
                "iam>ListMFADevices",  
                "iam>ListVirtualMFADevices",  
                "iam>ResyncMFADevice"  
            ],  
            "Resource": "*",  
            "Condition": {  
                "BoolIfExists": {  
                    "aws:MultiFactorAuthPresent": "false"  
                }  
            }  
        }  
    ]  
}
```

Resynchronizing virtual and hardware MFA devices (IAM console)

You can use the IAM console to resynchronize virtual and hardware MFA devices.

To resynchronize a virtual or hardware MFA device for your own IAM user (console)

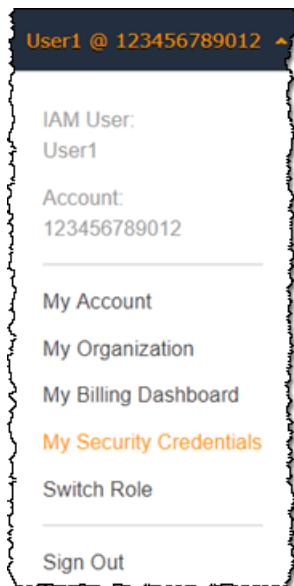
1. Use your AWS account ID or account alias, your IAM user name, and your password to sign in to the [IAM console](#).

Note

For your convenience, the AWS sign-in page uses a browser cookie to remember your IAM user name and account information. If you previously signed in as a different user, choose **Sign in to a different account** near the bottom of the page to return to the main sign-in page. From there, you can type your AWS account ID or account alias to be redirected to the IAM user sign-in page for your account.

To get your AWS account ID, contact your administrator.

2. In the navigation bar on the upper right, choose your user name, and then choose **My Security Credentials**.



3. On the **AWS IAM credentials** tab, in the **Multi-factor authentication** section, choose **Manage MFA device**.
4. In the **Manage MFA device** wizard, choose **Resync**, and then choose **Continue**.
5. Type the next two sequentially generated codes from the device into **MFA code 1** and **MFA code 2**. Then choose **Continue**.

Important

Submit your request immediately after generating the codes. If you generate the codes and then wait too long to submit the request, the request appears to work but the device remains out of sync. This happens because time-based one-time passwords (TOTP) expire after a short period of time.

To resynchronize a virtual or hardware MFA device for another IAM user (console)

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, choose **Users**, and then choose the name of the user whose MFA device needs to be resynchronized.
3. Choose the **Security credentials** tab. Next to **Assigned MFA device**, choose **Manage**.
4. In the **Manage MFA device** wizard, choose **Resync**, and then choose **Continue**.
5. Type the next two sequentially generated codes from the device into **MFA code 1** and **MFA code 2**. Then choose **Continue**.

Important

Submit your request immediately after generating the codes. If you generate the codes and then wait too long to submit the request, the request appears to work but the device remains out of sync. This happens because time-based one-time passwords (TOTP) expire after a short period of time.

To resynchronize your root user MFA before signing in (console)

1. On the **Amazon Web Services Sign In With Authentication Device** page, choose **Having problems with your authentication device? Click here**.

Note

You might see different text, such as **Sign in using MFA** and **Troubleshoot your authentication device**. However, the same features are provided.

2. In the **Re-Sync With Our Servers** section, type the next two sequentially generated codes from the device into **MFA code 1** and **MFA code 2**. Then choose **Re-sync authentication device**.
3. If necessary, type your password again and choose **Sign in**. Then complete the sign-in using your MFA device.

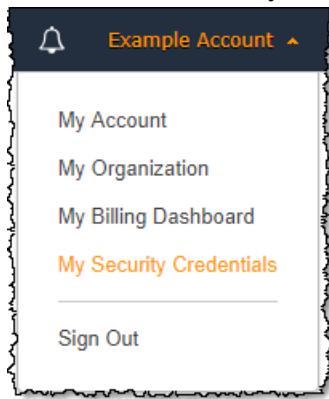
To resynchronize your root user MFA device after signing in (console)

1. Sign in to the [IAM console](#) as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

Note

If you see three text boxes, then you previously signed in to the console with [IAM user](#) credentials. Your browser might remember this preference and open this account-specific sign-in page every time that you try to sign in. You cannot use the IAM user sign-in page to sign in as the account owner. If you see the [IAM user sign-in page](#), choose **Sign in using root user email** near the bottom of the page. This returns you to the main sign-in page. From there, you can sign in as the root user using your AWS account email address and password.

2. On the right side of the navigation bar, choose on your account name, and then choose **My Security Credentials**. If necessary, choose **Continue to Security Credentials**.



3. Expand the **Multi-factor authentication (MFA)** section on the page.
4. Next to your active MFA device, choose **Resync**.
5. In the **Manage MFA device** dialog box, type the next two sequentially generated codes from the device into **MFA code 1** and **MFA code 2**. Then choose **Continue**.

Important

Submit your request immediately after generating the codes. If you generate the codes and then wait too long to submit the request, the MFA device is successfully associated with the user, but the MFA device is out of sync. This happens because time-based one-time passwords (TOTP) expire after a short period of time.

Resynchronizing virtual and hardware MFA devices (AWS CLI)

You can resynchronize virtual and hardware MFA devices from the AWS CLI.

To resynchronize a virtual or hardware MFA device for an IAM user (AWS CLI)

At a command prompt, issue the [aws iam resync-mfa-device](#) command:

- Virtual MFA device: Specify Amazon Resource Name (ARN) of device as the serial number.

```
aws iam resync-mfa-device --user-name Richard --serial-number  
arn:aws:iam::123456789012:mfa/RichardsMFA --authentication-code1 123456 --  
authentication-code2 987654
```

- Hardware MFA device: Specify hardware device's serial number as serial number. The format is vendor-specific. For example, you can purchase a gemalto token from Amazon. Its serial number is typically four letters followed by four numbers.

```
aws iam resync-mfa-device --user-name Richard --serial-number ABCD12345678 --  
authentication-code1 123456 --authentication-code2 987654
```

Important

Submit your request immediately after generating the codes. If you generate the codes and then wait too long to submit the request, the request fails because the codes expire after a short time.

Resynchronizing virtual and hardware MFA devices (AWS API)

IAM has an API call that performs synchronization. In this case, we recommend that you give your virtual and hardware MFA device users permission to access this API call. Then build a tool based on that API call so your users can resynchronize their devices whenever they need to.

To resynchronize a virtual or hardware MFA device for an IAM user (AWS API)

- Send the [ResyncMFADevice](#) request.

Deactivating MFA devices

If you have trouble signing in with a multi-factor authentication (MFA) device as an IAM user, contact your administrator for help.

As an administrator, you can deactivate the device for another IAM user. This allows the user to sign in without using MFA. You might do this as a temporary solution while the MFA device is replaced, or if the device is temporarily unavailable. However, we recommend that you enable a new device for the user as soon as possible. To learn how to enable a new MFA device, see [the section called “Enabling MFA devices” \(p. 112\)](#).

Note

If you use the API or AWS CLI to delete a user from your AWS account, you must deactivate or delete the user's MFA device. You make this change as part of the process of removing the user. For more information about deleting users, see [Managing IAM users \(p. 84\)](#).

Topics

- [Deactivating MFA devices \(console\) \(p. 134\)](#)
- [Deactivating MFA devices \(AWS CLI\) \(p. 135\)](#)
- [Deactivating MFA devices \(AWS API\) \(p. 136\)](#)

Deactivating MFA devices (console)

To deactivate an MFA device for another IAM user (console)

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, choose **Users**.
3. To deactivate the MFA device for a user, choose the name of the user whose MFA you want to remove.
4. Choose the **Security credentials** tab. Next to **Assigned MFA device**, choose **Manage**.
5. In the **Manage MFA device** wizard, choose **Deactivate MFA device**, and then choose **Continue**.

The device is removed from AWS. It cannot be used to sign in or authenticate requests until it is reactivated and associated with an AWS user or AWS account root user.

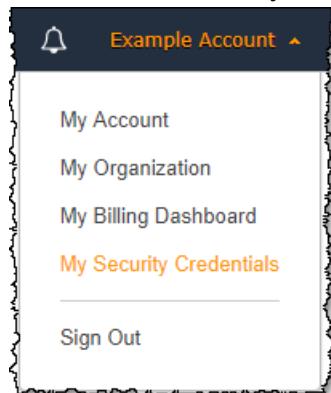
To deactivate the MFA device for your AWS account root user (console)

1. Sign in to the [IAM console](#) as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

Note

If you see three text boxes, then you previously signed in to the console with [IAM user](#) credentials. Your browser might remember this preference and open this account-specific sign-in page every time that you try to sign in. You cannot use the IAM user sign-in page to sign in as the account owner. If you see the [IAM user sign-in page](#), choose **Sign in using root user email** near the bottom of the page. This returns you to the main sign-in page. From there, you can sign in as the root user using your AWS account email address and password.

2. On the right side of the navigation bar, choose on your account name, and then choose **My Security Credentials**. If necessary, choose **Continue to Security Credentials**.



3. Expand the **Multi-factor authentication (MFA)** section.
4. In the row for the MFA device that you want to deactivate, choose **Deactivate**.

The MFA device is deactivated for the AWS account. Check the email that is associated with your AWS account for a confirmation message from Amazon Web Services. The email informs you that your Amazon Web Services multi-factor authentication (MFA) has been deactivated. The message will come from an address ending in @amazon.com or @aws.amazon.com.

Deactivating MFA devices (AWS CLI)

To deactivate an MFA device for an IAM user (AWS CLI)

- Run this command: `aws iam deactivate-mfa-device`

Deactivating MFA devices (AWS API)

To deactivate an MFA device for an IAM user (AWS API)

- Call this operation: [DeactivateMFADevice](#)

What if an MFA device is lost or stops working?

If your [virtual MFA device \(p. 114\)](#) or [hardware MFA device \(p. 122\)](#) appears to be functioning properly, but you cannot use it to access your AWS resources, it might be out of synchronization with AWS. For information about synchronizing a virtual MFA device or hardware MFA device, see [Resynchronizing virtual and hardware MFA devices \(p. 130\)](#). [U2F security keys \(p. 117\)](#) do not go out of sync.

If your AWS account root user [multi-factor authentication \(MFA\) device \(p. 111\)](#) is lost, damaged, or not working, you can recover access to your account. IAM users must contact an administrator to deactivate the device.

Recovering a root user MFA device

If your AWS account root user [multi-factor authentication \(MFA\) device \(p. 111\)](#) is lost, damaged, or not working, you can sign in using alternative methods of authentication. This means that if you can't sign in with your MFA device, you can sign in by verifying your identity using the email and phone that are registered with your account.

Before you sign in as a root user using alternative factors of authentication, make sure that you have access to the email and phone number that are associated with your account. If you no longer have access to the email or phone, you must contact [AWS Support](#). They can disable your MFA device so that you can sign in and add a new one.

To sign in using alternative factors of authentication as an AWS account root user

1. Sign in to the [AWS Management Console](#) as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.
2. On the [Amazon Web Services Sign In Using MFA](#) page, choose **Having problems with your authentication device? Click here.**

Note

You might see different text, such as **Sign in using MFA** and **Troubleshoot your authentication device**. However, the same features are provided. In either case, if you cannot verify your account email address and phone number using alternative factors of authentication, contact [AWS Support](#) to deactivate your MFA device.

3. If required, type your password again and choose **Sign in**.
4. In the **Sign In Using Alternative Factors of Authentication** section, choose **Sign in using alternative factors**.
5. To authenticate your account by verifying the email address, choose **Send verification email**.
6. Check the email that is associated with your AWS account for a message from Amazon Web Services (no-reply-aws@amazon.com). Follow the directions in the email.

If you don't see the email in your account, check your spam folder, or return to your browser and choose **Resend the email**.

7. After you verify your email address, you can continue authenticating your account. To verify your phone number, choose **Call me now**.
8. Answer the call from AWS and, when prompted, enter the 6-digit number from the AWS website on your phone keypad.

If you don't receive a call from AWS, choose **Sign in** to sign in to the console again and start over. Or choose **AWS Support** to contact support for help.

9. After you verify your phone number, you can sign in to your account by choosing **Sign in to the console**.
10. The next step varies depending on the type of MFA you are using:
 - For a virtual MFA device, remove the account from your device. Then go to the [AWS Security Credentials](#) page and delete the old MFA virtual device entity before you create a new one.
 - For a U2F security key, go to the [AWS Security Credentials](#) page and deactivate the old U2F key before enabling a new one.
 - For a hardware MFA device, contact the third-party provider for help fixing or replacing the device. You can continue to sign in using alternative factors of authentication until you receive your new device. After you have the new hardware MFA device, go to the [AWS Security Credentials](#) page and delete the old MFA hardware device entity before you create a new one.

Note

You don't have to replace a lost or stolen MFA device with the same type of device. For example, if you break your U2F security key and order a new one, you can use virtual MFA or a hardware MFA device until you receive a new U2F security key.

11. If your MFA device is missing or stolen, also [change your AWS password \(p. 92\)](#) in case an attacker has stolen the authentication device and might also have your current password.

Recovering an IAM user MFA device

If you are an IAM user and your device is lost or stops working, you can't recover it by yourself. You must contact an administrator to deactivate the device. Then you can enable a new device.

To get help for an MFA device as an IAM user

1. Contact the AWS administrator or other person who gave you the user name and password for the IAM user. The administrator must deactivate the MFA device as described in [Deactivating MFA devices \(p. 134\)](#) so that you can sign in.
2. The next step varies depending on the type of MFA you are using:
 - For a virtual MFA device, remove the account from your device. Then enable the virtual device as described in [Enabling a virtual multi-factor authentication \(MFA\) device \(console\) \(p. 114\)](#).
 - For a U2F security key, contact the third-party provider for help replacing the device. When you receive the new U2F security key, enable it as described in [Enabling a U2F security key \(console\) \(p. 117\)](#).
 - For a hardware MFA device, contact the third-party provider for help fixing or replacing the device. After you have the new physical MFA device, enable the device as described in [Enabling a hardware MFA device \(console\) \(p. 122\)](#).

Note

You don't have to replace a lost or stolen MFA device with the same type of device. For example, if you break your U2F security key and order a new one, you can use virtual MFA or a hardware MFA device until you receive a new U2F security key.

3. If your MFA device is missing or stolen, also [change your password \(p. 101\)](#) in case an attacker has stolen the authentication device and might also have your current password.

Configuring MFA-protected API access

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

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

Topics

- [Overview \(p. 138\)](#)
- [Scenario: MFA protection for cross-account delegation \(p. 140\)](#)
- [Scenario: MFA protection for access to API operations in the current account \(p. 142\)](#)
- [Scenario: MFA protection for resources that have resource-based policies \(p. 142\)](#)

Overview

Adding MFA protection to API operations involves these tasks:

1. The administrator configures an AWS MFA device for each user who needs to make API requests that require MFA authentication. This process is described at [Enabling MFA devices for users in AWS \(p. 112\)](#).
2. The administrator creates policies for the users that include a `Condition` element that checks whether the user authenticated with an AWS MFA device.
3. The user calls one of the AWS STS API operations that support the MFA parameters `AssumeRole` or `GetSessionToken`, depending on the scenario for MFA protection, as explained later. As part of the call, the user includes the device identifier for the device that's associated with the user. The user also includes the time-based one-time password (TOTP) that the device generates. In either case, the user gets back temporary security credentials that the user can then use to make additional requests to AWS.

Note

MFA protection for a service's API operations is available only if the service supports temporary security credentials. For a list of these services, see [Using Temporary Security Credentials to Access AWS](#).

If authorization fails, 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 API operations specified in the policies if the user attempts to call an API operation without valid MFA authentication. The operation is also denied if the time stamp of the request for the API operation is outside of the allowed range specified in the policy. The user must be reauthenticated with MFA by requesting new temporary security credentials with an MFA code and device serial number.

IAM policies with MFA conditions

Policies with MFA conditions can be attached to the following:

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

You can use an MFA condition in a policy to check the following properties:

- Existence—To simply verify that the user did authenticate with MFA, check that the `aws:MultiFactorAuthPresent` key is `True` in a `Bool` condition. The key is only present when the user authenticates with short-term credentials. Long-term credentials, such as access keys, do not include this key.
- Duration—if you want to grant access only within a specified time after MFA authentication, use a numeric condition type to compare the `aws:MultiFactorAuthAge` key's age to a value (such as 3600 seconds). Note that the `aws:MultiFactorAuthAge` key is not present if MFA was not used.

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

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

For more information on the condition types for MFA, see [AWS global condition context keys \(p. 692\)](#), [Numeric condition operators \(p. 646\)](#), and [Condition operator to check existence of condition keys \(p. 651\)](#).

Choosing between GetSessionToken and AssumeRole

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

Use `GetSessionToken` for the following scenarios:

- Call API operations that access resources in the same AWS account as the IAM user who makes the request. Note that temporary credentials from a `GetSessionToken` request can access IAM and AWS STS API operations *only* if you include MFA information in the request for credentials. Because temporary credentials returned by `GetSessionToken` include MFA information, you can check for MFA in individual API operations made by the credentials.
- Access to resources that are protected with resource-based policies that include an MFA condition.

The purpose of the `GetSessionToken` operation is to authenticate the user using MFA. You cannot use policies to control authentication operations.

Use `AssumeRole` for the following scenarios:

- Call API operations that access resources in the same or a different AWS account. The API calls can include any IAM or AWS STS API. Note that to protect access you enforce MFA at the time when the user assumes the role. The temporary credentials returned by `AssumeRole` do not include MFA information in the context, so you cannot check individual API operations for MFA. This is why you must use `GetSessionToken` to restrict access to resources protected by resource-based policies.

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

Important points about MFA-protected API access

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

- MFA protection is available only with temporary security credentials, which must be obtained with `AssumeRole` or `GetSessionToken`.
- You cannot use MFA-protected API access with AWS account root user credentials.
- You cannot use MFA-protected API access with U2F security keys.
- Federated users cannot be assigned an MFA device for use with AWS services, so they cannot access AWS resources controlled by MFA. (See next point.)
- Other AWS STS API operations that return temporary credentials do not support MFA. For `AssumeRoleWithWebIdentity` and `AssumeRoleWithSAML`, the user is authenticated by an external provider and AWS cannot determine whether that provider required MFA. For `GetFederationToken`, MFA is not necessarily associated with a specific user.
- Similarly, long-term credentials (IAM user access keys and root user access keys) cannot be used with MFA-protected API access because they don't expire.
- `AssumeRole` and `GetSessionToken` can also be called without MFA information. In that case, the caller gets back temporary security credentials, but the session information for those temporary credentials does not indicate that the user authenticated with MFA.
- To establish MFA protection for API operations, you add MFA conditions to policies. A policy must include the `aws:MultiFactorAuthPresent` condition key to enforce the use of MFA. For cross-account delegation, the role's trust policy must include the condition key.
- When you allow another AWS account to access resources in your account, the security of your resources depends on the configuration of the trusted account (the other account, not yours). This is true even when you require multi-factor authentication. Any identity in the trusted account that has permission to create virtual MFA devices can construct an MFA claim to satisfy that part of your role's trust policy. Before you allow members of another account access to your AWS resources that require multi-factor authentication, you should ensure that the trusted account's owner follows security best practices. For example, the trusted account should restrict access to sensitive API operations, such as MFA device-management API operations, to specific, trusted identities.
- If a policy includes an MFA condition, a request is denied if users have not been MFA authenticated, or if they provide an invalid MFA device identifier or invalid TOTP.

Scenario: MFA protection for cross-account delegation

In this scenario, you want to delegate access to IAM users in another account, but only if the users are authenticated with an AWS MFA device. (For more information about cross-account delegation, see [Roles terms and concepts \(p. 168\)](#).)

Imagine that you have account A (the trusting account that owns the resource to be accessed), with the IAM user Anaya, who has administrator permission. She wants to grant access to user Richard in account B (the trusted account), but wants to make sure that Richard is authenticated with MFA before he assumes the role.

1. In the trusting account A, Anaya creates an IAM role named `CrossAccountRole` and sets the principal in the role's trust policy to the account ID of account B. The trust policy grants permission to the AWS STS `AssumeRole` action. Anaya also adds an MFA condition to the trust policy, as in the following example.

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

```

        "Action": "sts:AssumeRole",
        "Condition": {"Bool": {"aws:MultiFactorAuthPresent": "true"}}
    }
}

```

2. Anaya adds a permissions policy to the role that specifies what the role is allowed to do. The permissions policy for a role with MFA protection is no different than any other role-permission policy. The following example shows the policy that Anaya adds to the role; it allows an assuming user to perform any Amazon DynamoDB action on the table Books in account A. This policy also allows the dynamodb>ListTables action, which is required to perform actions in the console.

Note

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

```

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

```

3. In trusted account B, the administrator makes sure that IAM user Richard is configured with an AWS MFA device and that he knows the ID of the device. The device ID is the serial number if it's a hardware MFA device, or the device's ARN if it's a virtual MFA device.
4. In account B, the administrator attaches the following policy to user Richard (or a group that he's a member of) that allows him to call the AssumeRole action. The resource is set to the ARN of the role that Anaya created in step 1. Notice that this policy does not contain an MFA condition.

```

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

```

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

When Richard calls AssumeRole, AWS determines whether he has valid credentials, including the requirement for MFA. If so, Richard successfully assumes the role and can perform any DynamoDB action on the table named Books in account A while using the role's temporary credentials.

For an example of a program that calls AssumeRole, see [Calling AssumeRole with MFA authentication \(Python\) \(p. 145\)](#).

Scenario: MFA protection for access to API operations in the current account

In this scenario, you should ensure that a user in your AWS account can access sensitive API operations only when the user is authenticated using an AWS MFA device.

Imagine that you have account A that contains a group of developers who need to work with EC2 instances. Ordinary developers can work with the instances, but they are not granted permissions for the `ec2:StopInstances` or `ec2:TerminateInstances` actions. You want to limit those "destructive" privileged actions to just a few trusted users, so you add MFA protection to the policy that allows these sensitive Amazon EC2 actions.

In this scenario, one of those trusted users is user Sofía. User Anaya is an administrator in account A.

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

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

4. **Note**

For this policy to take effect, users must first sign out and then sign in again.

If user Sofía needs to stop or terminate an Amazon EC2 instance, she (or an application that she is running) calls `GetSessionToken`. This API operation passes the ID of the MFA device and the current TOTP that Sofía gets from her device.

5. User Sofía (or an application that Sofía is using) uses the temporary credentials provided by `GetSessionToken` to call the Amazon EC2 `StopInstances` or `TerminateInstances` action.

For an example of a program that calls `GetSessionToken`, see [Calling `GetSessionToken` with MFA authentication \(Python and C#\) \(p. 144\)](#) later in this document.

Scenario: MFA protection for resources that have resource-based policies

In this scenario, you are the owner of an S3 bucket, an SQS queue, or an SNS topic. You want to make sure that any user from any AWS account who accesses the resource is authenticated by an AWS MFA device.

This scenario illustrates a way to provide cross-account MFA protection without requiring users to assume a role first. In this case, the user can access the resource if three conditions are met: The user must be authenticated by MFA, be able to get temporary security credentials from `GetSessionToken`, and be in an account that is trusted by the resource's policy.

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

In this scenario, user Anaya is an administrator in account A. User Nikhil is an IAM user in account C.

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

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

Note

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

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

For an example of a program that calls `GetSessionToken`, see [Calling `GetSessionToken` with MFA authentication \(Python and C#\) \(p. 144\)](#) later in this document.

Note

The temporary credentials that `AssumeRole` returns won't work in this case. Although the user can provide MFA information to assume a role, the temporary credentials returned by `AssumeRole` don't include the MFA information. That information is required in order to meet the MFA condition in the policy.

Sample code: Requesting credentials with multi-factor authentication

The following examples show how to call `GetSessionToken` and `AssumeRole` operations and pass MFA authentication parameters. No permissions are required to call `GetSessionToken`, but you must have a policy that allows you to call `AssumeRole`. The credentials returned are then used to list all S3 buckets in the account.

Calling GetSessionToken with MFA authentication (Python and C#)

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

The policy attached to the user who runs this code (or to a group that the user is in) provides the permissions for the returned temporary credentials. For this example code, the policy must grant the user permission to request the Amazon S3 `ListBuckets` operation.

Using Python

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

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

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

sts_connection = STSConnection()

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

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

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

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

Using C#

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

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

```

GetSessionTokenRequest getSessionTokenRequest = new GetSessionTokenRequest();
getSessionTokenRequest.DurationSeconds = 3600;

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

GetSessionTokenResponse getSessionTokenResponse =
    stsClient.GetSessionToken(getSessionTokenRequest);

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

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

```

Calling AssumeRole with MFA authentication (Python)

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

For more information about this scenario, see [Scenario: MFA protection for cross-account delegation \(p. 140\)](#).

```

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

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

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

sts_connection = STSConnection()

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

```

```
    role_session_name="AssumeRoleSession1",
    mfa_serial_number="arn:aws:iam::ACCOUNT-NUMBER-WITHOUT-HYPHENS:mfa/MFA-DEVICE-ID",
    mfa_token=mfa_TOTP
)

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

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

Finding unused credentials

To increase the security of your AWS account, remove IAM user credentials (that is, passwords and access keys) that are not needed. For example, when users leave your organization or no longer need AWS access, find the credentials that they were using and ensure that they are no longer operational. Ideally, you delete credentials if they are no longer needed. You can always recreate them at a later date if the need arises. At the very least, you should change the password or deactivate the access keys so that the former users no longer have access.

Of course, the definition of *unused* can vary and usually means a credential that has not been used within a specified period of time.

Finding unused passwords

You can use the AWS Management Console to view password usage information for your users. If you have a large number of users, you can use the console to download a credential report with information about when each user last used their console password. You can also access the information from the AWS CLI or the IAM API.

To find unused passwords (console)

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, choose **Users**.
3. If necessary, add the **Console last sign-in** column to the users table:
 - a. Above the table on the far right, choose the settings icon ().
 - b. In **Manage Columns**, select **Console last sign-in**.
 - c. Choose **Close** to return to the list of users.
4. The **Console last sign-in** column shows the date when the user last signed in to AWS through the console. You can use this information to find users with passwords who have not signed in for more than a specified period of time. The column displays **Never** for users with passwords that have never signed in. **None** indicates users with no passwords. Passwords that have not been used recently might be good candidates for removal.

Important

Due to a service issue, password last used data does not include password use from May 3rd 2018 22:50 PDT to May 23rd 2018 14:08 PDT. This affects **last sign-in** dates shown in the IAM console and password last used dates in the [IAM credential report](#), and returned by the

GetUser API operation. If users signed in during the affected time, the password last used date that is returned is the date the user last signed in before May 3rd 2018. For users that signed in after May 23rd 2018 14:08 PDT, the returned password last used date is accurate. If you use password last used information to identify unused credentials for deletion, such as deleting users who did not sign in to AWS in the last 90 days, we recommend that you adjust your evaluation window to include dates after May 23rd 2018. Alternatively, if your users use access keys to access AWS programmatically you can refer to access key last used information because it is accurate for all dates.

To find unused passwords by downloading the credentials report (console)

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, choose **Credential report**.
3. Choose **Download Report** to download a comma-separated value (CSV) file named `status_reports_<date>T<time>.csv`. The fifth column contains the `password_last_used` column with the dates or one of the following:
 - **N/A** – Users that do not have a password assigned at all.
 - **no_information** – Users that have not used their password since IAM began tracking password age on October 20, 2014.

To find unused passwords (AWS CLI)

Run the following command to find unused passwords:

- `aws iam list-users` returns a list of users, each with a `PasswordLastUsed` value. If the value is missing, then the user either has no password or the password has not been used since IAM began tracking password age on October 20, 2014.

To find unused passwords (AWS API)

Call the following operation to find unused passwords:

- `ListUsers` returns a collection of users, each of which has a `<PasswordLastUsed>` value. If the value is missing, then the user either has no password or the password has not been used since IAM began tracking password age on October 20, 2014.

For information about the commands to download the credentials report, see [Getting credential reports \(AWS CLI\) \(p. 152\)](#).

Finding unused access keys

You can use the AWS Management Console to view access key usage information for your users. If you have a large number of users, you can use the console to download a credentials report to find when each user last used their access keys. You can also access the information from the AWS CLI or the IAM API.

To find unused access keys (console)

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, choose **Users**.
3. If necessary, add the **Access key last used** column to the users table:

- a. Above the table on the far right, choose the settings icon ().
 - b. In **Manage Columns**, select **Access key last used**.
 - c. Choose **Close** to return to the list of users.
4. The **Access key last used** column shows the number of days since the user last accessed AWS programmatically. You can use this information to find users with access keys that have not been used for more than a specified period of time. The column displays **None** for users with no access keys. Access keys that have not been used recently might be good candidates for removal.

To find unused access keys by downloading the credentials report (console)

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, choose **Credential Report**.
3. Choose **Download Report** to download a comma-separated value (CSV) file named `status_reports_<date>T<time>.csv`. Columns 11 through 13 contain the last used date, Region, and service information for access key 1. Columns 16 through 18 contain the same information for access key 2. The value is **N/A** if the user does not have an access key or the user has not used the access key since IAM began tracking access key age on April 22, 2015.

To find unused access keys (AWS CLI)

Run the following commands to find unused access keys:

- `aws iam list-access-keys` returns information about the access keys for a user, including the `AccessKeyId`.
- `aws iam get-access-key-last-used` takes an access key ID and returns output that includes the `LastUsedDate`, the `Region` in which the access key was last used, and the `ServiceName` of the last service requested. If `LastUsedDate` is missing, then the access key has not been used since IAM began tracking access key age on April 22, 2015.

To find unused access keys (AWS API)

Call the following operations to find unused access keys:

- `ListAccessKeys` returns a list of `AccessKeyId` values for access keys that are associated with the specified user.
- `GetAccessKeyLastUsed` takes an access key ID and returns a collection of values. Included are the `LastUsedDate`, the `Region` in which the access key was last used, and the `ServiceName` of the last service requested. If the value is missing, then either the user has no access key or the access key has not been used since IAM began tracking access key age on April 22, 2015.

For information about the commands to download the credentials report, see [Getting credential reports \(AWS CLI\) \(p. 152\)](#).

Getting credential reports for your AWS account

You can generate and download a *credential report* that lists all users in your account and the status of their various credentials, including passwords, access keys, and MFA devices. You can get a credential report from the AWS Management Console, the [AWS SDKs](#) and [Command Line Tools](#), or the IAM API.

You can use credential reports to assist in your auditing and compliance efforts. You can use the report to audit the effects of credential lifecycle requirements, such as password and access key rotation. You

can provide the report to an external auditor, or grant permissions to an auditor so that he or she can download the report directly.

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

Topics

- [Required permissions \(p. 149\)](#)
- [Understanding the report format \(p. 149\)](#)
- [Getting credential reports \(console\) \(p. 152\)](#)
- [Getting credential reports \(AWS CLI\) \(p. 152\)](#)
- [Getting credential reports \(AWS API\) \(p. 152\)](#)

Required permissions

The following permissions are needed to create and download reports:

- To create a credential report: `GenerateCredentialReport`
- To download the report: `GetCredentialReport`

Understanding the report format

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

The CSV file contains the following columns:

user

The friendly name of the user.

arn

The Amazon Resource Name (ARN) of the user. For more information about ARNs, see [IAM ARNs \(p. 601\)](#).

user_creation_time

The date and time when the user was created, in [ISO 8601 date-time format](#).

password_enabled

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

password_last_used

The date and time when the AWS account root user or IAM user's password was last used to sign in to an AWS website, in [ISO 8601 date-time format](#). AWS websites that capture a user's last sign-in time are the AWS Management Console, the AWS Discussion Forums, and the AWS Marketplace. When a password is used more than once in a 5-minute span, only the first use is recorded in this field.

- The value in this field is `no_information` in these cases:
 - The user's password has never been used.

- There is no sign-in data associated with the password, such as when user's password has not been used after IAM started tracking this information on October 20, 2014.
- The value in this field is **N/A** (not applicable) when the user does not have a password.

Important

Due to a service issue, password last used data does not include password use from May 3rd 2018 22:50 PDT to May 23rd 2018 14:08 PDT. This affects [last sign-in](#) dates shown in the IAM console and password last used dates in the [IAM credential report](#), and returned by the [GetUser API operation](#). If users signed in during the affected time, the password last used date that is returned is the date the user last signed in before May 3rd 2018. For users that signed in after May 23rd 2018 14:08 PDT, the returned password last used date is accurate.

If you use password last used information to identify unused credentials for deletion, such as deleting users who did not sign in to AWS in the last 90 days, we recommend that you adjust your evaluation window to include dates after May 23rd 2018. Alternatively, if your users use access keys to access AWS programmatically you can refer to access key last used information because it is accurate for all dates.

password_last_changed

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

password_next_rotation

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

mfa_active

When a [multi-factor authentication \(p. 111\)](#) (MFA) device has been enabled for the user, this value is **TRUE**. Otherwise it is **FALSE**.

access_key_1_active

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

access_key_1_last_rotated

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

access_key_1_last_used_date

The date and time, in [ISO 8601 date-time format](#), when the user's access key was most recently used to sign an AWS API request. When an access key is used more than once in a 15-minute span, only the first use is recorded in this field.

The value in this field is **N/A** (not applicable) in these cases:

- The user does not have an access key.
- The access key has never been used.
- The access key has not been used after IAM started tracking this information on April 22, 2015.

access_key_1_last_used_region

The [AWS Region](#) in which the access key was most recently used. When an access key is used more than once in a 15-minute span, only the first use is recorded in this field.

The value in this field is **N/A** (not applicable) in these cases:

- The user does not have an access key.
- The access key has never been used.
- The access key was last used before IAM started tracking this information on April 22, 2015.
- The last used service is not Region-specific, such as Amazon S3.

access_key_1_last_used_service

The AWS service that was most recently accessed with the access key. The value in this field uses the service's namespace—for example, s3 for Amazon S3 and ec2 for Amazon EC2. When an access key is used more than once in a 15-minute span, only the first use is recorded in this field.

The value in this field is **N/A** (not applicable) in these cases:

- The user does not have an access key.
- The access key has never been used.
- The access key was last used before IAM started tracking this information on April 22, 2015.

access_key_2_active

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

Note

Users can have up to two access keys, to make rotation easier. For more information about rotating access keys, see [Rotating access keys \(p. 106\)](#).

access_key_2_last_rotated

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

access_key_2_last_used_date

The date and time, in [ISO 8601 date-time format](#), when the user's second access key was most recently used to sign an AWS API request. When an access key is used more than once in a 15-minute span, only the first use is recorded in this field.

The value in this field is **N/A** (not applicable) in these cases:

- The user does not have a second access key.
- The user's second access key has never been used.
- The user's second access key was last used before IAM started tracking this information on April 22, 2015.

access_key_2_last_used_region

The [AWS Region](#) in which the user's second access key was most recently used. When an access key is used more than once in a 15-minute span, only the first use is recorded in this field. The value in this field is **N/A** (not applicable) in these cases:

- The user does not have a second access key.
- The user's second access key has never been used.
- The user's second access key was last used before IAM started tracking this information on April 22, 2015.
- The last used service is not Region-specific, such as Amazon S3.

access_key_2_last_used_service

The AWS service that was most recently accessed with the user's second access key. The value in this field uses the service's namespace—for example, s3 for Amazon S3 and ec2 for Amazon EC2. When an access key is used more than once in a 15-minute span, only the first use is recorded in this field. The value in this field is **N/A** (not applicable) in these cases:

- The user does not have a second access key.
- The user's second access key has never been used.
- The user's second access key was last used before IAM started tracking this information on April 22, 2015.

cert_1_active

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

cert_1_last_rotated

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

cert_2_active

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

Note

Users can have up to two X.509 signing certificates, to make certificate rotation easier.

cert_2_last_rotated

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

Getting credential reports (console)

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

To download a credential report (console)

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, choose **Credential report**.
3. Choose **Download Report**.

Getting credential reports (AWS CLI)

To download a credentials report (AWS CLI)

1. Generate a credentials report. AWS stores a single report. If a report exists, generating a credentials report overwrites the previous report. `aws iam generate-credential-report`
2. View the last report that was generated: `aws iam get-credential-report`

Getting credential reports (AWS API)

To download a credentials report (AWS API)

1. Generate a credentials report. AWS stores a single report. If a report exists, generating a credentials report overwrites the previous report. `GenerateCredentialReport`
2. View the last report that was generated: `GetCredentialReport`

Using IAM with CodeCommit: Git credentials, SSH keys, and AWS access keys

CodeCommit is a managed version control service that hosts private Git repositories in the AWS cloud. To use CodeCommit, you configure your Git client to communicate with CodeCommit repositories. As part of this configuration, you provide IAM credentials that CodeCommit can use to authenticate you. IAM supports CodeCommit with three types of credentials:

- Git credentials, an IAM -generated user name and password pair you can use to communicate with CodeCommit repositories over HTTPS.
- SSH keys, a locally generated public-private key pair that you can associate with your IAM user to communicate with CodeCommit repositories over SSH.
- [AWS access keys \(p. 102\)](#), which you can use with the credential helper included with the AWS CLI to communicate with CodeCommit repositories over HTTPS.

See the following sections for more information about each option.

Use Git credentials and HTTPS with CodeCommit (recommended)

With Git credentials, you generate a static user name and password pair for your IAM user, and then use those credentials for HTTPS connections. You can also use these credentials with any third-party tool or integrated development environment (IDE) that supports static Git credentials.

Because these credentials are universal for all supported operating systems and compatible with most credential management systems, development environments, and other software development tools, this is the recommended method. You can reset the password for Git credentials at any time. You can also make the credentials inactive or delete them if you no longer need them.

Note

You cannot choose your own user name or password for Git credentials. IAM generates these credentials for you to help ensure they meet the security standards for AWS and secure repositories in CodeCommit. You can download the credentials only once, at the time they are generated. Make sure that you save the credentials in a secure location. If necessary, you can reset the password at any time, but doing so invalidates any connections configured with the old password. You must reconfigure connections to use the new password before you can connect.

See the following topics for more information:

- To create an IAM user, see [Creating an IAM user in your AWS account \(p. 76\)](#).
- To generate and use Git credentials with CodeCommit, see [For HTTPS Users Using Git Credentials](#) in the [AWS CodeCommit User Guide](#).

Note

Changing the name of an IAM user after generating Git credentials does not change the user name of the Git credentials. The user name and password remain the same and are still valid.

To rotate service specific credentials

1. Create a second service-specific credential set in addition to the set currently in use.
2. Update all of your applications to use the new set of credentials and validate that the applications are working.
3. Change the state of the original credentials to "Inactive".

4. Ensure that all of your applications are still working.
5. Delete the inactive service-specific credentials.

Use SSH keys and SSH with CodeCommit

With SSH connections, you create public and private key files on your local machine that Git and CodeCommit use for SSH authentication. You associate the public key with your IAM user and store the private key on your local machine. See the following topics for more information:

- To create an IAM user, see [Creating an IAM user in your AWS account \(p. 76\)](#).
- To create an SSH public key and associate it with an IAM user, see [For SSH Connections on Linux, macOS, or Unix](#) or see [For SSH Connections on Windows](#) in the *AWS CodeCommit User Guide*.

Note

The public key must be encoded in ssh-rsa format or PEM format. The minimum bit-length of the public key is 2048 bits, and the maximum length is 16384 bits. This is separate from the size of the file you upload. For example, you can generate a 2048-bit key, and the resulting PEM file is 1679 bytes long. If you provide your public key in another format or size, you will see an error message stating that the key format is not valid.

Use HTTPS with the AWS CLI credential helper and CodeCommit

As an alternative to HTTPS connections with Git credentials, you can allow Git to use a cryptographically signed version of your IAM user credentials or Amazon EC2 instance role whenever Git needs to authenticate with AWS to interact with CodeCommit repositories. This is the only connection method for CodeCommit repositories that does not require an IAM user. This is also the only method that works with federated access and temporary credentials. Unless your business needs require federated access or the use of temporary credentials, creating and using IAM users for access is strongly recommended. See the following topics for more information:

- To learn more about federated access, see [Identity providers and federation \(p. 176\)](#) and [Providing access to externally authenticated users \(identity federation\) \(p. 174\)](#).
- To learn more about temporary credentials, see [Temporary security credentials in IAM \(p. 301\)](#) and [Temporary Access to CodeCommit Repositories](#).

The AWS CLI credential helper is not compatible with other credential helper systems, such as Keychain Access or Windows Credential Management. There are additional configuration considerations when you configure HTTPS connections with the credential helper. For more information, see [For HTTPS Connections on Linux, macOS, or Unix with the AWS CLI Credential Helper](#) or [HTTPS Connections on Windows with the AWS CLI Credential Helper](#) in the *AWS CodeCommit User Guide*.

Using IAM with Amazon Keyspaces (for Apache Cassandra)

Amazon Keyspaces (for Apache Cassandra) is a scalable, highly available, and managed Apache Cassandra-compatible database service. You can access Amazon Keyspaces using the AWS Management Console, by running a cqlsh client, or using an Apache 2.0 licensed Cassandra driver.

Note

If you plan to interact with Amazon Keyspaces only through the console, you don't need to generate service-specific credentials. For more information, see [Accessing Amazon Keyspaces \(for Apache Cassandra\)](#) in the *Amazon Keyspaces (for Apache Cassandra) Developer Guide*.

You must generate service-specific credentials to allow your users to access Amazon Keyspaces using cqlsh or an Apache 2.0 licensed Cassandra driver. *Service-specific credentials* allow an IAM user to interact with one AWS service, but no others.

For more information about the permissions required to access Amazon Keyspaces, see [Amazon Keyspaces \(for Apache Cassandra\) Identity-Based Policy Examples](#) in the *Amazon Keyspaces (for Apache Cassandra) Developer Guide*.

Generating Amazon Keyspaces credentials (console)

You can use the AWS Management Console to generate Amazon Keyspaces (for Apache Cassandra) credentials for your IAM users.

To generate Amazon Keyspaces service-specific credentials (console)

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, choose **Users** and then choose the name of the user that requires the credentials.
3. On the **Security Credentials** tab beneath **Credentials for Amazon Keyspaces (for Apache Cassandra)**, choose **Generate credentials**.
4. Your service-specific credentials are now available. This is the only time that the password can be viewed or downloaded. You cannot recover it later. However, you can reset your password at any time. Save the user and password in a secure location, because you'll need them later.

Generating Amazon Keyspaces credentials (AWS CLI)

You can use the AWS CLI to generate Amazon Keyspaces (for Apache Cassandra) credentials for your IAM users.

To generate Amazon Keyspaces service-specific credentials (AWS CLI)

- Use the following command:
 - `aws iam create-service-specific-credential`

Generating Amazon Keyspaces credentials (AWS API)

You can use the AWS API to generate Amazon Keyspaces (for Apache Cassandra) credentials for your IAM users.

To generate Amazon Keyspaces service-specific credentials (AWS API)

- Complete the following operation:
 - [CreateServiceSpecificCredential](#)

Managing server certificates in IAM

To enable HTTPS connections to your website or application in AWS, you need an *SSL/TLS server certificate*. For certificates in a Region supported by AWS Certificate Manager (ACM), we recommend that you use ACM to provision, manage, and deploy your server certificates. In unsupported Regions, you must use IAM as a certificate manager. To learn which Regions ACM supports, see [AWS Certificate Manager endpoints and quotas](#) in the *AWS General Reference*.

ACM is the preferred tool to provision, manage, and deploy your server certificates. With ACM you can request a certificate or deploy an existing ACM or external certificate to AWS resources. Certificates provided by ACM are free and automatically renew. In a [supported Region](#), you can use ACM to manage server certificates from the console or programmatically. For more information about using ACM, see the [AWS Certificate Manager User Guide](#). For more information about requesting an ACM certificate, see [Request a Public Certificate](#) or [Request a Private Certificate](#) in the [AWS Certificate Manager User Guide](#). For more information about importing third party certificates into ACM, see [Importing Certificates](#) in the [AWS Certificate Manager User Guide](#).

Use IAM as a certificate manager only when you must support HTTPS connections in a Region that is not [supported by ACM](#). IAM securely encrypts your private keys and stores the encrypted version in IAM SSL certificate storage. IAM supports deploying server certificates in all Regions, but you must obtain your certificate from an external provider for use with AWS. You cannot upload an ACM certificate to IAM. Additionally, you cannot manage your certificates from the IAM Console.

For more information about uploading third party certificates to IAM, see the following topics.

Contents

- [Uploading a server certificate \(AWS API\) \(p. 156\)](#)
- [Retrieving a server certificate \(AWS API\) \(p. 157\)](#)
- [Listing server certificates \(AWS API\) \(p. 157\)](#)
- [Renaming a server certificate or updating its path \(AWS API\) \(p. 157\)](#)
- [Deleting a server certificate \(AWS API\) \(p. 158\)](#)
- [Troubleshooting \(p. 158\)](#)

Uploading a server certificate (AWS API)

To upload a server certificate to IAM, you must provide the certificate and its matching private key. When the certificate is not self-signed, you must also provide a certificate chain. (You don't need a certificate chain when uploading a self-signed certificate.) Before you upload a certificate, ensure that you have all these items and that they meet the following criteria:

- The certificate must be valid at the time of upload. You cannot upload a certificate before its validity period begins (the certificate's `NotBefore` date) or after it expires (the certificate's `NotAfter` date).
- The private key must be unencrypted. You cannot upload a private key that is protected by a password or passphrase. For help decrypting an encrypted private key, see [Troubleshooting \(p. 158\)](#).
- The certificate, private key, and certificate chain must all be PEM-encoded. For help converting these items to PEM format, see [Troubleshooting \(p. 158\)](#).

To use the [IAM API](#) to upload a certificate, send an `UploadServerCertificate` request. The following example shows how to do this with the [AWS Command Line Interface \(AWS CLI\)](#). The example assumes the following:

- The PEM-encoded certificate is stored in a file named `Certificate.pem`.
- The PEM-encoded certificate chain is stored in a file named `CertificateChain.pem`.
- The PEM-encoded, unencrypted private key is stored in a file named `PrivateKey.pem`.

To use the following example command, replace these file names with your own and replace `ExampleCertificate` with a name for your uploaded certificate. Type the command on one continuous line. The following example includes line breaks and extra spaces to make it easier to read.

```
aws iam upload-server-certificate --server-certificate-name ExampleCertificate
```

```
--certificate-body file://Certificate.pem
--certificate-chain file://CertificateChain.pem
--private-key file://PrivateKey.pem
```

When the preceding command is successful, it returns metadata about the uploaded certificate, including its [Amazon Resource Name \(ARN\) \(p. 601\)](#), its friendly name, its identifier (ID), its expiration date, and more.

Note

If you are uploading a server certificate to use with Amazon CloudFront, you must specify a path using the --path option. The path must begin with /cloudfront and must include a trailing slash (for example, /cloudfront/test/).

To use the AWS Tools for Windows PowerShell to upload a certificate, use [Publish-IAMServerCertificate](#).

Retrieving a server certificate (AWS API)

To use the IAM API to retrieve a certificate, send a [GetServerCertificate](#) request. The following example shows how to do this with the AWS CLI. Replace *ExampleCertificate* with the name of the certificate to retrieve.

```
aws iam get-server-certificate --server-certificate-name ExampleCertificate
```

When the preceding command is successful, it returns the certificate, the certificate chain (if one was uploaded), and metadata about the certificate.

Note

You cannot download or retrieve a private key from IAM after you upload it.

To use the AWS Tools for Windows PowerShell to retrieve a certificate, use [Get-IAMServerCertificate](#).

Listing server certificates (AWS API)

To use the IAM API to list your uploaded server certificates, send a [ListServerCertificates](#) request. The following example shows how to do this with the AWS CLI.

```
aws iam list-server-certificates
```

When the preceding command is successful, it returns a list that contains metadata about each certificate.

To use the AWS Tools for Windows PowerShell to list your uploaded server certificates, use [Get-IAMServerCertificates](#).

Renaming a server certificate or updating its path (AWS API)

To use the IAM API to rename a server certificate or update its path, send an [UpdateServerCertificate](#) request. The following example shows how to do this with the AWS CLI.

To use the following example command, replace the old and new certificate names and the certificate path, and type the command on one continuous line. The following example includes line breaks and extra spaces to make it easier to read.

```
aws iam update-server-certificate --server-certificate-name ExampleCertificate
--new-server-certificate-name CloudFrontCertificate
```

```
--new-path /cloudfront/
```

When the preceding command is successful, it does not return any output.

To use the AWS Tools for Windows PowerShell to rename a server certificate or update its path, use [Update-IAMServerCertificate](#).

Deleting a server certificate (AWS API)

To use the IAM API to delete a server certificate, send a [DeleteServerCertificate](#) request. The following example shows how to do this with the AWS CLI.

To use the following example command, replace *ExampleCertificate* with the name of the certificate to delete.

```
aws iam delete-server-certificate --server-certificate-name ExampleCertificate
```

When the preceding command is successful, it does not return any output.

To use the AWS Tools for Windows PowerShell to delete a server certificate, use [Remove-IAMServerCertificate](#).

Troubleshooting

Before you can upload a certificate to IAM, you must make sure that the certificate, private key, and certificate chain are all PEM-encoded. You must also ensure that the private key is unencrypted. See the following examples.

Example Example PEM-encoded certificate

```
-----BEGIN CERTIFICATE-----  
Base64-encoded certificate  
-----END CERTIFICATE-----
```

Example Example PEM-encoded, unencrypted private key

```
-----BEGIN RSA PRIVATE KEY-----  
Base64-encoded private key  
-----END RSA PRIVATE KEY-----
```

Example Example PEM-encoded certificate chain

A certificate chain contains one or more certificates. You can use a text editor, the copy command in Windows, or the Linux cat command to concatenate your certificate files into a chain. When you include multiple certificates, each certificate must certify the preceding certificate. You accomplish this by concatenating the certificates, including the root CA certificate last.

The following example contains three certificates, but your certificate chain might contain more or fewer certificates.

```
-----BEGIN CERTIFICATE-----  
Base64-encoded certificate  
-----END CERTIFICATE-----  
-----BEGIN CERTIFICATE-----
```

```
-----  
-----BEGIN CERTIFICATE-----  
-----  
-----  
-----  
-----END CERTIFICATE-----  
-----  
-----  
-----END CERTIFICATE-----
```

If these items are not in the right format for uploading to IAM, you can use [OpenSSL](#) to convert them to the right format.

To convert a certificate or certificate chain from DER to PEM

Use the [OpenSSL x509 command](#), as in the following example. In the following example command, replace `Certificate.der` with the name of the file that contains your DER-encoded certificate. Replace `Certificate.pem` with the preferred name of the output file to contain the PEM-encoded certificate.

```
openssl x509 -inform DER -in Certificate.der -outform PEM -out Certificate.pem
```

To convert a private key from DER to PEM

Use the [OpenSSL rsa command](#), as in the following example. In the following example command, replace `PrivateKey.der` with the name of the file that contains your DER-encoded private key. Replace `PrivateKey.pem` with the preferred name of the output file to contain the PEM-encoded private key.

```
openssl rsa -inform DER -in PrivateKey.der -outform PEM -out PrivateKey.pem
```

To decrypt an encrypted private key (remove the password or passphrase)

Use the [OpenSSL rsa command](#), as in the following example. To use the following example command, replace `EncryptedPrivateKey.pem` with the name of the file that contains your encrypted private key. Replace `PrivateKey.pem` with the preferred name of the output file to contain the PEM-encoded unencrypted private key.

```
openssl rsa -in EncryptedPrivateKey.pem -out PrivateKey.pem
```

To convert a certificate bundle from PKCS#12 (PFX) to PEM

Use the [OpenSSL pkcs12 command](#), as in the following example. In the following example command, replace `CertificateBundle.p12` with the name of the file that contains your PKCS#12-encoded certificate bundle. Replace `CertificateBundle.pem` with the preferred name of the output file to contain the PEM-encoded certificate bundle.

```
openssl pkcs12 -in CertificateBundle.p12 -out CertificateBundle.pem -nodes
```

To convert a certificate bundle from PKCS#7 to PEM

Use the [OpenSSL pkcs7 command](#), as in the following example. In the following example command, replace `CertificateBundle.p7b` with the name of the file that contains your PKCS#7-encoded

certificate bundle. Replace `CertificateBundle.pem` with the preferred name of the output file to contain the PEM-encoded certificate bundle.

```
openssl pkcs7 -in CertificateBundle.p7b -print_certs -out CertificateBundle.pem
```

IAM groups

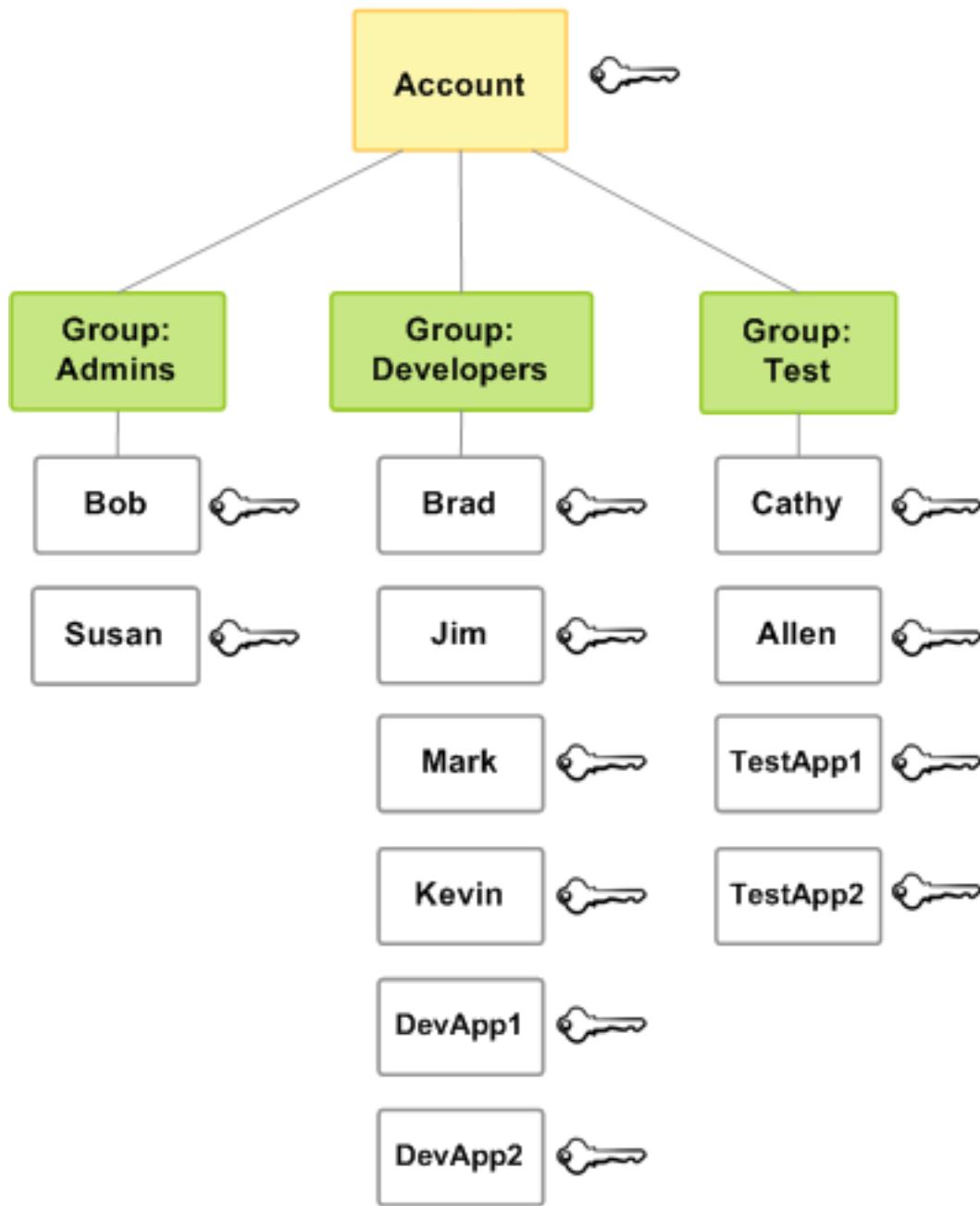
An IAM [group \(p. 160\)](#) is a collection of IAM users. Groups let you specify permissions for multiple users, which can make it easier to manage the permissions for those users. For example, you could have a group called *Admins* and give that group the types of permissions that administrators typically need. Any user in that group automatically has the permissions that are assigned to the group. If a new user joins your organization and needs administrator privileges, you can assign the appropriate permissions by adding the user to that group. Similarly, if a person changes jobs in your organization, instead of editing that user's permissions, you can remove him or her from the old groups and add him or her to the appropriate new groups.

A group cannot be identified as a `Principal` in a resource-based policy. A group is a way to attach policies to multiple users at one time. When you attach an identity-based policy to a group, all of the users in the group receive the permissions from the group. For more information about these policy types, see [Identity-based policies and resource-based policies \(p. 374\)](#).

Following are some important characteristics of groups:

- A group can contain many users, and a user can belong to multiple groups.
- Groups can't be nested; they can contain only users, not other groups.
- There's no default group that automatically includes all users in the AWS account. If you want to have a group like that, you need to create it and assign each new user to it.
- The number and size of IAM resources in an AWS account are limited. For more information, see [IAM and STS quotas \(p. 606\)](#).

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



Creating IAM groups

To set up a group, you need to create the group. Then give the group permissions based on the type of work that you expect the users in the group to do. Finally, add users to the group.

For information about the permissions that you need in order to create a group, see [Permissions required to access IAM resources \(p. 516\)](#).

To create an IAM group and attach policies (console)

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 **Groups** and then click **Create New Group**.
3. In the **Group Name** box, type the name of the group and then click **Next Step**.

Note

The number and size of IAM resources in an AWS account are limited. For more information, see [IAM and STS quotas \(p. 606\)](#). Group names can be a combination of up to 64 letters, digits, and these characters: plus (+), equal (=), comma (,), period (.), at sign (@), underscore (_), and hyphen (-). Names must be unique within an account. They are not distinguished by case. For example, you cannot create groups named both **ADMINS** and **admins**.

4. In the list of policies, select the check box for each policy that you want to apply to all members of the group. Then click **Next Step**.
5. Click **Create Group**.

For an example of how to set up an **Administrators** group, see [Creating your first IAM admin user and group \(p. 20\)](#).

To create IAM groups (AWS CLI or AWS API)

Use one of the following:

- AWS CLI: `aws iam create-group`
- AWS API: `CreateGroup`

Managing IAM groups

Amazon Web Services offers multiple tools for managing IAM groups. For information about the permissions that you need in order to add and remove users in a group, see [Permissions required to access IAM resources \(p. 516\)](#).

Topics

- [Listing IAM groups \(p. 162\)](#)
- [Adding and removing users in an IAM group \(p. 163\)](#)
- [Attaching a policy to an IAM group \(p. 164\)](#)
- [Renaming an IAM group \(p. 165\)](#)
- [Deleting an IAM group \(p. 165\)](#)

Listing IAM groups

You can list all the groups in your account, list the users in a group, and list the groups a user belongs to. If you use the AWS CLI or AWS API, you can list all the groups with a particular path prefix.

To list all the groups in your account

Do any of the following:

- [AWS Management Console](#): In the navigation pane, choose **Groups**.
- AWS CLI: `aws iam list-groups`

- AWS API: [ListGroup](#)

To list the users in a specific group

Do any of the following:

- [AWS Management Console](#): In the navigation pane, choose **Groups**, choose the name of the group, and then choose the **Users** tab.
- AWS CLI: `aws iam get-group`
- AWS API: `GetGroup`

To list all the groups that a user is in

Do any of the following:

- [AWS Management Console](#): In the navigation pane, choose **Users**, choose the user name, and then choose the **Groups** tab.
- AWS CLI: `aws iam list-groups-for-user`
- AWS API: `ListGroupsForUser`

Adding and removing users in an IAM group

Use groups to apply the same permissions policies across multiple users at once. You can then add users to or remove users from an IAM group. This is useful as people enter and leave your organization.

View policy access

Before you change the permissions for a policy, you should review its recent service-level activity. This is important because you don't want to remove access from a principal (person or application) who is using it. For more information about viewing last accessed information, see [Refining permissions in AWS using last accessed information \(p. 472\)](#).

Add or remove a user in a group (console)

You can use the AWS Management Console to add or remove a user from a group.

To add a user to an IAM group (console)

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, choose **Groups** and then choose the name of the group.
3. Choose the **Users** tab and then choose **Add Users to Group**. Select the check box next to the users you want to add.
4. Choose **Add Users**.

To remove a user from an IAM group (console)

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, choose **Groups** and then choose the name of the group.
3. Choose the **Users** tab and then choose **Remove Users from Group**. Select the check box next to the users you want to remove.

4. Choose **Remove Users**.

Add or remove a user in a group (AWS CLI)

You can use the AWS CLI to add or remove a user from a group.

To add a user to an IAM group (AWS CLI)

- Use the following command:
 - [aws iam add-user-to-group](#)

To remove a user from an IAM group (AWS CLI)

- Use the following command:
 - [aws iam remove-user-from-group](#)

Add or remove a user in a group (AWS API)

You can use the AWS API to add or remove a user in a group.

To add a user to an IAM group (AWS API)

- Complete the following operation:
 - [AddUserToGroup](#)

To remove a user from an IAM group (AWS API)

- Complete the following operation:
 - [RemoveUserFromGroup](#)

Attaching a policy to an IAM group

You can attach an [AWS managed policy \(p. 360\)](#)—that is, a prewritten policy provided by AWS—to a group, as explained in the following steps. To attach a customer managed policy—that is, a policy with custom permissions that you create—you must first create the policy. For information about creating customer managed policies, see [Creating IAM policies \(p. 439\)](#).

For more information about permissions and policies, see [Access management for AWS resources \(p. 350\)](#).

To attach a policy to a group (console)

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, select **Policies**.
3. In the list of policies, select the check box next to the name of the policy to attach. You can use the **Filter** menu and the search box to filter the list of policies.
4. Click **Policy actions**, then click **Attach**.
5. For **Filter**, choose **All Types**, then click **Groups**.
6. Select the check box next to the name of the group to attach the policy to, then click **Attach policy**.

To attach a policy to a group (AWS CLI or AWS API)

Do either of the following:

- AWS CLI: [aws iam attach-group-policy](#)
- AWS API: [AttachGroupPolicy](#)

Renaming an IAM group

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

- Any policies attached to the group stay with the group under the new name.
- The group retains all its users under the new name.
- The unique ID for the group remains the same. For more information about unique IDs, see [Unique identifiers \(p. 604\)](#).

Because IAM does not automatically update policies that refer to the group as a resource to use the new name; you must be careful when you rename a group. Before you rename your group, you must manually check all of your policies to find any policies where that group is mentioned by name. For example, let's say Bob is the manager of the testing part of the organization. Bob has a policy attached to his IAM user entity that lets him add and remove users from the Test group. If an administrator changes the name of the group (or changes the group path), the administrator must also update the policy attached to Bob to use the new name or path. Otherwise Bob won't be able to add and remove users from the group.

To find policies that refer to a group as a resource:

1. From the navigation pane of the IAM console, choose **Policies**.
2. From the **Policy type** drop-down list, choose **Customer managed** to filter the policies to show only your custom policies.
3. Choose the arrow next to each policy name to expand the policy summary.
4. Choose **IAM** from the list of services, if it exists.
5. Look for the name of your group in the **Resource** column.
6. Choose **Edit policy** to change the name of your group in the policy.

To change the name of an IAM group

Do any of the following:

- **AWS Management Console:** In the navigation pane, choose **Groups** and then select the check box next to the group name. From the **Group Actions** list at the top of the page, choose **Edit Group Name**. Type the new group name and then choose **Yes, Edit**.
- AWS CLI: [aws iam update-group](#)
- AWS API: [UpdateGroup](#)

Deleting an IAM group

When you delete a group in the AWS Management Console, the console automatically removes all group members, detaches all attached managed policies, and deletes all inline policies. However, because IAM does not automatically delete policies that refer to the group as a resource, you must be careful when you delete a group. Before you delete your group, you must manually check all of your policies to find any policies where that group is mentioned by name. For example, let's say John is the manager of the

testing part of the organization. John has a policy attached to his IAM user entity that lets him add and remove users from the Test group. If an administrator deletes the group, the administrator must also delete the policy attached to John.

To find policies that refer to a group as a resource

1. From the navigation pane of the IAM console, choose **Policies**.
2. From the **Policy type** drop-down list, choose **Customer managed** to filter the policies to show only your custom policies.
3. Choose the arrow next to each policy name to expand the policy summary.
4. Choose **IAM** from the list of services, if it exists.
5. Look for the name of your group in the **Resource** column.
6. Choose **Delete policy** to delete the policy.

In contrast, when you use the AWS CLI, Tools for Windows PowerShell, or AWS API to delete a group, you must first remove the users in the group. Then delete any inline policies embedded in the group. Next, detach any managed policies that are attached to the group. Only then can you delete the group itself.

Deleting an IAM group (console)

You can delete an IAM group from the AWS Management Console.

To delete an IAM group (console)

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, choose **Groups**.
3. In the list of groups, select the check box next to the name of the group to delete. You can use the **Filter** menu and the search box to filter the list of policies.
4. Click **Group Actions**, then click **Delete Group**.
5. In the confirmation box, click **Yes, Delete**.

Deleting an IAM group (AWS CLI)

You can delete an IAM group from the AWS CLI.

To delete an IAM group (AWS CLI)

1. Remove all users from the group.
 - [aws iam get-group](#) (to get the list of users in the group), and [aws iam remove-user-from-group](#) (to remove a user from the group)
2. Delete all inline policies embedded in the group.
 - [aws iam list-group-policies](#) (to get a list of the group's inline policies), and [aws iam delete-group-policy](#) (to delete the group's inline policies)
3. Detach all managed policies attached to the group.
 - [aws iam list-attached-group-policies](#) (to get a list of the managed policies attached to the group), and [aws iam detach-group-policy](#) (to detach a managed policy from the group)
4. Delete the group.
 - [aws iam delete-group](#)

Deleting an IAM group (AWS API)

You can use the AWS API to delete an IAM group.

To delete an IAM group (AWS API)

1. Remove all users from the group.
 - [GetGroup](#) (to get the list of users in the group) and [RemoveUserFromGroup](#) (to remove a user from the group)
2. Delete all inline policies embedded in the group.
 - [ListGroupPolicies](#) (to get a list of the group's inline policies) and [DeleteGroupPolicy](#) (to delete the group's inline policies)
3. Detach all managed policies attached to the group.
 - [ListAttachedGroupPolicies](#) (to get a list of the managed policies attached to the group) and [DetachGroupPolicy](#) (to detach a managed policy from the group)
4. Delete the group.
 - [DeleteGroup](#)

IAM roles

An IAM *role* is an IAM identity that you can create in your account that has specific permissions. An IAM role is similar to an IAM user, in that it is an AWS identity with permission policies that determine what the identity can and cannot do in AWS. However, instead of being uniquely associated with one person, a role is intended to be assumable by anyone who needs it. Also, a role does not have standard long-term credentials such as a password or access keys associated with it. Instead, when you assume a role, it provides you with temporary security credentials for your role session.

You can use roles to delegate access to users, applications, or services that don't normally have access to your AWS resources. For example, you might want to grant users in your AWS account access to resources they don't usually have, or grant users in one AWS account access to resources in another account. Or you might want to allow a mobile app to use AWS resources, but not want to embed AWS keys within the app (where they can be difficult to rotate and where users can potentially extract them). Sometimes you want to give AWS access to users who already have identities defined outside of AWS, such as in your corporate directory. Or, you might want to grant access to your account to third parties so that they can perform an audit on your resources.

For these scenarios, you can delegate access to AWS resources using an *IAM role*. This section introduces roles and the different ways you can use them, when and how to choose among approaches, and how to create, manage, switch to (or assume), and delete roles.

Topics

- [Roles terms and concepts \(p. 168\)](#)
- [Common scenarios for roles: Users, applications, and services \(p. 170\)](#)
- [Identity providers and federation \(p. 176\)](#)
- [Using service-linked roles \(p. 213\)](#)
- [Creating IAM roles \(p. 221\)](#)
- [Using IAM roles \(p. 246\)](#)
- [Managing IAM roles \(p. 272\)](#)
- [How IAM roles differ from resource-based policies \(p. 286\)](#)

Roles terms and concepts

Here are some basic terms to help you get started with roles.

Role

An IAM identity that you can create in your account that has specific permissions. An IAM role has some similarities to an IAM user. Roles and users are both AWS identities with permissions policies that determine what the identity can and cannot do in AWS. However, instead of being uniquely associated with one person, a role is intended to be assumable by anyone who needs it. Also, a role does not have standard long-term credentials such as a password or access keys associated with it. Instead, when you assume a role, it provides you with temporary security credentials for your role session.

Roles can be used by the following:

- An IAM user in the same AWS account as the role
- An IAM user in a different AWS account than the role
- A web service offered by AWS such as Amazon Elastic Compute Cloud (Amazon EC2)
- An external user authenticated by an external identity provider (IdP) service that is compatible with SAML 2.0 or OpenID Connect, or a custom-built identity broker.

AWS service role

A role that a service assumes to perform actions in your account on your behalf. When you set up some AWS service environments, you must define a role for the service to assume. This service role must include all the permissions required for the service to access the AWS resources that it needs. Service roles vary from service to service, but many allow you to choose your permissions, as long as you meet the documented requirements for that service. Service roles provide access only within your account and cannot be used to grant access to services in other accounts. You can create, modify, and delete a service role from within IAM.

AWS service role for an EC2 instance

A special type of service role that an application running on an Amazon EC2 instance can assume to perform actions in your account. This role is assigned to the EC2 instance when it is launched. Applications running on that instance can retrieve temporary security credentials and perform actions that the role allows. For details about using a service role for an EC2 instance, see [Using an IAM role to grant permissions to applications running on Amazon EC2 instances \(p. 263\)](#).

AWS service-linked role

A unique type of service role that is linked directly to an AWS service. Service-linked roles are predefined by the service and include all the permissions that the service requires to call other AWS services on your behalf. The linked service also defines how you create, modify, and delete a service-linked role. A service might automatically create or delete the role. It might allow you to create, modify, or delete the role as part of a wizard or process in the service. Or it might require that you use IAM to create or delete the role. Regardless of the method, service-linked roles make setting up a service easier because you don't have to manually add the necessary permissions.

Note

If you are already using a service when it begins supporting service-linked roles, you might receive an email announcing a new role in your account. In this case, the service automatically created the service-linked role in your account. You don't need to take any action to support this role, and you should not manually delete it. For more information, see [A new role appeared in my AWS account \(p. 587\)](#).

For information about which services support using service-linked roles, see [AWS services that work with IAM \(p. 611\)](#) and look for the services that have Yes in the **Service-Linked Role** column. Choose a Yes with a link to view the service-linked role documentation for that service. If the service does not include documentation for creating, modifying, or deleting the service-linked role,

then you can use the IAM console, AWS CLI, or API. For more information, see [Using service-linked roles \(p. 213\)](#).

Role chaining

Role chaining occurs when you use a role to assume a second role through the AWS CLI or API. For example, assume that `User1` has permission to assume `RoleA` and `RoleB`. Additionally, `RoleA` has permission to assume `RoleB`. You can assume `RoleA` by using `User1`'s long-term user credentials in the `AssumeRole` API operation. This operation returns `RoleA` short-term credentials. To engage in role chaining, you can use `RoleA`'s short-term credentials to assume `RoleB`.

When you assume a role, you can pass a session tag and set the tag as transitive. Transitive session tags are passed to all subsequent sessions in a role chain. To learn more about session tags, see [Passing session tags in AWS STS \(p. 293\)](#).

Role chaining limits your AWS CLI or AWS API role session to a maximum of one hour. When you use the `AssumeRole` API operation to assume a role, you can specify the duration of your role session with the `DurationSeconds` parameter. You can specify a parameter value of up to 43200 seconds (12 hours), depending on the [maximum session duration setting \(p. 248\)](#) for your role. However, if you assume a role using role chaining and provide a `DurationSeconds` parameter value greater than one hour, the operation fails.

AWS does not treat using roles to [grant permissions to applications that run on EC2 instances \(p. 263\)](#) as role chaining.

Delegation

The granting of permissions to someone to allow access to resources that you control. Delegation involves setting up a trust between two accounts. The first is the account that owns the resource (the trusting account). The second is the account that contains the users that need to access the resource (the trusted account). The trusted and trusting accounts can be any of the following:

- The same account.
- Separate accounts that are both under your organization's control.
- Two accounts owned by different organizations.

To delegate permission to access a resource, you [create an IAM role \(p. 221\)](#) in the trusting account that has two [policies \(p. 170\)](#) attached. The *permissions policy* grants the user of the role the needed permissions to carry out the intended tasks on the resource. The *trust policy* specifies which trusted account members are allowed to assume the role.

When you create a trust policy, you cannot specify a wildcard (*) as a principal. The trust policy is attached to the role in the trusting account, and is one-half of the permissions. The other half is a permissions policy attached to the user in the trusted account that [allows that user to switch to, or assume the role \(p. 248\)](#). A user who assumes a role temporarily gives up his or her own permissions and instead takes on the permissions of the role. When the user exits, or stops using the role, the original user permissions are restored. An additional parameter called [external ID \(p. 225\)](#) helps ensure secure use of roles between accounts that are not controlled by the same organization.

Federation

The creation of a trust relationship between an external identity provider and AWS. Users can sign in to a web identity provider, such as [Login with Amazon](#), [Facebook](#), [Google](#), or any IdP that is compatible with [OpenID Connect \(OIDC\)](#). Users can also sign in to an enterprise identity system that is compatible with Security Assertion Markup Language (SAML) 2.0, such as Microsoft Active Directory Federation Services. When you use OIDC and SAML 2.0 to configure a trust relationship between these external identity providers and AWS, the user is assigned to an IAM role. The user also receives temporary credentials that allow the user to access your AWS resources.

Federated user

Instead of creating an IAM user, you can use existing identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as *federated users*. AWS

assigns a role to a federated user when access is requested through an [identity provider \(p. 176\)](#). For more information about federated users, see [Federated Users and Roles \(p. 12\)](#) in the [IAM User Guide](#).

Trust policy

A [JSON policy document \(p. 679\)](#) in which you define the principals that you *trust* to assume the role. A role trust policy is a required [resource-based policy \(p. 352\)](#) that is attached to a role in IAM. The [principals \(p. 631\)](#) that you can specify in the trust policy include users, roles, accounts, and services.

Permissions policy

A permissions document in [JSON](#) format in which you define what actions and resources the role can use. The document is written according to the rules of the [IAM policy language \(p. 627\)](#).

Permissions boundary

An advanced feature in which you use policies to limit the maximum permissions that an identity-based policy can grant to a role. You cannot apply a permissions boundary to a service-linked role. For more information, see [Permissions boundaries for IAM entities \(p. 365\)](#).

Principal

An entity in AWS that can perform actions and access resources. A principal can be an AWS account root user, an IAM user, or a role. You can grant permissions to access a resource in one of two ways:

- You can attach a permissions policy to a user (directly, or indirectly through a group) or to a role.
- For those services that support [resource-based policies \(p. 12\)](#), you can identify the principal in the `Principal` element of a policy attached to the resource.

If you reference an AWS account as principal, it generally means any principal defined within that account.

Note

You cannot use a wildcard (*) in the `Principal` element in a role's trust policy.

Role for cross-account access

A role that grants access to resources in one account to a trusted principal in a different account. Roles are the primary way to grant cross-account access. However, some AWS services allow you to attach a policy directly to a resource (instead of using a role as a proxy). These are called resource-based policies, and you can use them to grant principals in another AWS account access to the resource. Some of these resources include Amazon Simple Storage Service (S3) buckets, S3 Glacier vaults, Amazon Simple Notification Service (SNS) topics, and Amazon Simple Queue Service (SQS) queues. To learn which services support resource-based policies, see [AWS services that work with IAM \(p. 611\)](#). For more information about resource-based policies, see [How IAM roles differ from resource-based policies \(p. 286\)](#).

Common scenarios for roles: Users, applications, and services

As with most AWS features, you generally have two ways to use a role: interactively in the IAM console, or programmatically with the AWS CLI, Tools for Windows PowerShell, or API.

- IAM users in your account using the IAM console can *switch to* a role to temporarily use the permissions of the role in the console. The users give up their original permissions and take on the permissions assigned to the role. When the users exit the role, their original permissions are restored.
- An application or a service offered by AWS (like Amazon EC2) can *assume* a role by requesting temporary security credentials for a role with which to make programmatic requests to AWS. You use

a role this way so that you don't have to share or maintain long-term security credentials (for example, by creating an IAM user) for each entity that requires access to a resource.

Note

This guide uses the phrases *switch to a role* and *assume a role* interchangeably.

The simplest way to use roles is to grant your IAM users permissions to switch to roles that you create within your own or another AWS account. They can switch roles easily using the IAM console to use permissions that you don't ordinarily want them to have, and then exit the role to surrender those permissions. This can help prevent *accidental* access to or modification of sensitive resources.

For more complex uses of roles, such as granting access to applications and services, or federated external users, you can call the `AssumeRole` API. This API call returns a set of temporary credentials that the application can use in subsequent API calls. Actions attempted with the temporary credentials have only the permissions granted by the associated role. An application doesn't have to "exit" the role the way a user in the console does; rather the application simply stops using the temporary credentials and resumes making calls with the original credentials.

Federated users sign in by using credentials from an identity provider (IdP). AWS then provides temporary credentials to the trusted IdP to pass on to the user for including in subsequent AWS resource requests. Those credentials provide the permissions granted to the assigned role.

This section provides overviews of the following scenarios:

- Provide access for an IAM user in one AWS account that you own to access resources in another account that you own (p. 171)
- Provide access to IAM users in AWS accounts owned by third parties (p. 173)
- Provide access for services offered by AWS to AWS resources (p. 174)
- Provide access for externally authenticated users (identity federation) (p. 174)

Providing access to an IAM user in another AWS account that you own

You can grant your IAM users permission to switch to roles within your AWS account or to roles defined in other AWS accounts that you own.

Note

If you want to grant access to an account that you do not own or control, see [Providing access to AWS accounts owned by third parties \(p. 173\)](#) later in this topic.

Imagine that you have Amazon EC2 instances that are critical to your organization. Instead of directly granting your users permission to terminate the instances, you can create a role with those privileges. Then allow administrators to switch to the role when they need to terminate an instance. Doing this adds the following layers of protection to the instances:

- You must explicitly grant your users permission to assume the role.
- Your users must actively switch to the role using the AWS Management Console or assume the role using the AWS CLI or AWS API.
- You can add multi-factor authentication (MFA) protection to the role so that only users who sign in with an MFA device can assume the role. To learn how to configure a role so that users who assume the role must first be authenticated using multi-factor authentication (MFA), see [Configuring MFA-protected API access \(p. 138\)](#).

We recommend using this approach to enforce the *principle of least privilege*. That means restricting the use of elevated permissions to only those times when they are needed for specific tasks. With roles

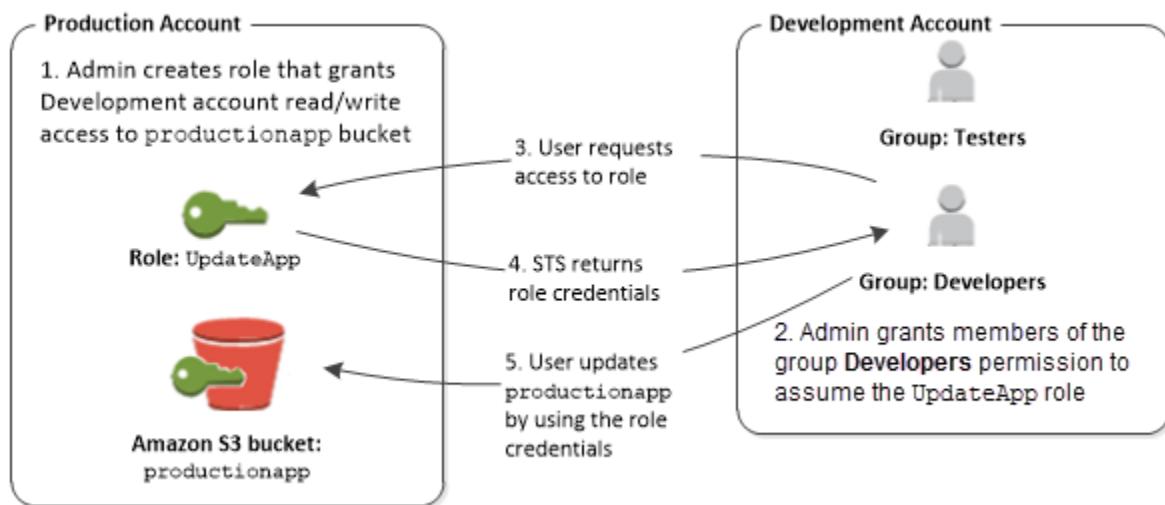
you can help prevent accidental changes to sensitive environments, especially if you combine them with [auditing \(p. 336\)](#) to help ensure that roles are only used when needed.

When you create a role for this purpose, you specify the accounts by ID whose users need access in the `Principal` element of the role's trust policy. You can then grant specific users in those other accounts permissions to switch to the role. To learn whether principals in accounts outside of your zone of trust (trusted organization or account) have access to assume your roles, see [What is IAM Access Analyzer?](#).

A user in one account can switch to a role in the same or a different account. While using the role, the user can perform only the actions and access only the resources permitted by the role; their original user permissions are suspended. When the user exits the role, the original user permissions are restored.

Example scenario using separate development and production accounts

Imagine that your organization has multiple AWS accounts to isolate a development environment from a production environment. Users in the development account might occasionally need to access resources in the production account. For example, you might need cross-account access when you are promoting an update from the development environment to the production environment. Although you could create separate identities (and passwords) for users who work in both accounts, managing credentials for multiple accounts makes identity management difficult. 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.



1. In the production account, an administrator uses IAM to create the `UpdateApp` role in that account. In the role, the administrator defines a trust policy that specifies the development account as a `Principal`, meaning that authorized users from the development account can use the `UpdateApp` role. The administrator also defines a permissions policy for the role that specifies the read and write permissions to the Amazon S3 bucket named `productionapp`.

The administrator then shares the appropriate information with anyone who needs to assume the role. That information is the account number and name of the role (for AWS console users) or the Amazon Resource Name (ARN) (for AWS CLI or AWS API access). The role ARN might look like `arn:aws:iam::123456789012:role/UpdateApp`, where the role is named `updateApp` and the role was created in account number 123456789012.

Note

The administrator can optionally configure the role so that users who assume the role must first be authenticated using multi-factor authentication (MFA). For more information, see [Configuring MFA-protected API access \(p. 138\)](#).

2. In the development account, an administrator grants members of the Developers group permission to switch to the role. This is done by granting the Developers group permission to call the AWS Security Token Service (AWS STS) `AssumeRole` API for the `UpdateApp` role. Any IAM user that belongs to the Developers group in the development account can now switch to the `UpdateApp` role in the production account. Other users who are not in the developer group do not have permission to switch to the role and therefore cannot access the S3 bucket in the production account.
3. The user requests switches to the role:
 - AWS console: The user chooses the account name on the navigation bar and chooses **Switch Role**. The user specifies the account ID (or alias) and role name. Alternatively, the user can click on a link sent in email by the administrator. The link takes the user to the **Switch Role** page with the details already filled in.
 - AWS API/AWS CLI: A user in the Developers group of the development account calls the `AssumeRole` function to obtain credentials for the `UpdateApp` role. The user specifies the ARN of the `UpdateApp` role as part of the call. If a user in the Testers group makes the same request, the request fails because Testers do not have permission to call `AssumeRole` for the `UpdateApp` role ARN.
4. AWS STS returns temporary credentials:
 - AWS console: AWS STS verifies the request with the role's trust policy to ensure that the request is from a trusted entity (which it is: the development account). After verification, AWS STS returns **temporary security credentials** to the AWS console.
 - API/CLI: AWS STS verifies the request against the role's trust policy to ensure that the request is from a trusted entity (which it is: the Development account). After verification, AWS STS returns **temporary security credentials** to the application.
5. The temporary credentials allow access to the AWS resource:
 - AWS console: The AWS console uses the temporary credentials on behalf of the user for all subsequent console actions, in this case, to read and write to the `productionapp` bucket. The console cannot access any other resource in the production account. When the user exits the role, the user's permissions revert to the original permissions held before switching to the role.
 - API/CLI: The application uses the temporary security credentials to update the `productionapp` bucket. With the temporary security credentials, the application can only read from and write to the `productionapp` bucket and cannot access any other resource in the Production account. The application does not have to exit the role, but instead stops using the temporary credentials and uses the original credentials in subsequent API calls.

Providing access to AWS accounts owned by third parties

When third parties require access to your organization's AWS resources, you can use roles to delegate access to them. For example, a third party might provide a service for managing your AWS resources. With IAM roles, you can grant these third parties access to your AWS resources without sharing your AWS security credentials. Instead, the third party can access your AWS resources by assuming a role that you create in your AWS account. To learn whether principals in accounts outside of your zone of trust (trusted organization or account) have access to assume your roles, see [What is IAM Access Analyzer?](#).

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

- The third party's AWS account ID. You specify their AWS account ID as the principal when you define the trust policy for the role.
- An external ID to uniquely associate with the role. The external ID can be any secret identifier that is known by you and the third party. For example, you can use an invoice ID between you and the third party, but do not use something that can be guessed, like the name or phone number of the third party. You must specify this ID when you define the trust policy for the role. The third party must provide this ID when they assume the role. For more information about the external ID, see [How to use an external ID when granting access to your AWS resources to a third party \(p. 225\)](#).

- The permissions that the third party requires to work with your AWS resources. You must specify these permissions when defining the role's permission policy. This policy defines what actions they can take and what resources they can access.

After you create the role, you must provide the role's Amazon Resource Name (ARN) to the third party. They require your role's ARN in order to assume the role.

For details about creating a role to delegate access to a third party, see [How to use an external ID when granting access to your AWS resources to a third party \(p. 225\)](#).

Important

When you grant third parties access to your AWS resources, they can access any resource that you specify in the policy. Their use of your resources is billed to you. Ensure that you limit their use of your resources appropriately.

Providing access to an AWS service

Many AWS services require that you use roles to control what that service can access. A role that a service assumes to perform actions on your behalf is called a [service role \(p. 168\)](#). When a role serves a specialized purpose for a service, it can be categorized as a [service role for EC2 instances \(p. 168\)](#), or a [service-linked role \(p. 168\)](#). See the [AWS documentation](#) for each service to see if it uses roles and to learn how to assign a role for the service to use.

For details about creating a role to delegate access to a service offered by AWS, see [Creating a role to delegate permissions to an AWS service \(p. 229\)](#).

Providing access to externally authenticated users (identity federation)

Your users might already have identities outside of AWS, such as in your corporate directory. If those users need to work with AWS resources (or work with applications that access those resources), then those users also need AWS security credentials. You can use an IAM role to specify permissions for users whose identity is federated from your organization or a third-party identity provider (IdP).

Federating users of a mobile or web-based app with Amazon Cognito

If you create a mobile or web-based app that accesses AWS resources, the app needs security credentials in order to make programmatic requests to AWS. For most mobile application scenarios, we recommend that you use [Amazon Cognito](#). You can use this service with the [AWS Mobile SDK for iOS](#) and the [AWS Mobile SDK for Android and Fire OS](#) to create unique identities for users and authenticate them for secure access to your AWS resources. Amazon Cognito supports the same identity providers as those listed in the next section, and it also supports [developer authenticated identities](#) and unauthenticated (guest) access. Amazon Cognito also provides API operations for synchronizing user data so that it is preserved as users move between devices. For more information, see [Using Amazon Cognito for mobile apps \(p. 177\)](#).

Federating users with public identity service providers or OpenID Connect

Whenever possible, use Amazon Cognito for mobile and web-based application scenarios. Amazon Cognito does most of the behind-the-scenes work with public identity provider services for you. It works with the same third-party services and also supports anonymous sign-ins. However, for more advanced scenarios, you can work directly with a third-party service like Login with Amazon, Facebook, Google, or any IdP that is compatible with OpenID Connect (OIDC). For more information about using web identity federation using one of these services, see [About web identity federation \(p. 177\)](#).

Federating users with SAML 2.0

If your organization already uses an identity provider software package that supports SAML 2.0 (Security Assertion Markup Language 2.0), you can create trust between your organization as an identity provider (IdP) and AWS as the service provider. You can then use SAML to provide your users with federated single-sign on (SSO) to the AWS Management Console or federated access to call AWS API operations. For example, if your company uses Microsoft Active Directory and Active Directory Federation Services, then you can federate using SAML 2.0. For more information about federating users with SAML 2.0, see [About SAML 2.0-based federation \(p. 182\)](#).

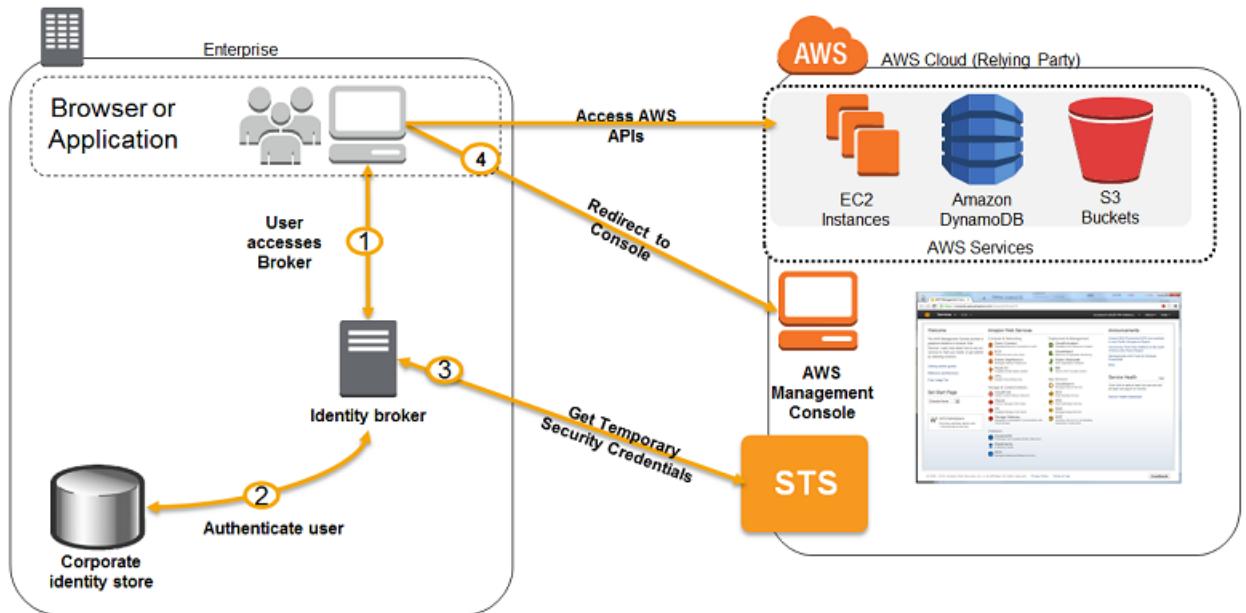
Federating users by creating a custom identity broker application

If your identity store is not compatible with SAML 2.0, then you can build a custom identity broker application to perform a similar function. The broker application authenticates users, requests temporary credentials for users from AWS, and then provides them to the user to access AWS resources.

For example, Example Corp. has many employees who need to run internal applications that access the company's AWS resources. The employees already have identities in the company identity and authentication system, and Example Corp. doesn't want to create a separate IAM user for each company employee.

Bob is a developer at Example Corp. To enable Example Corp. internal applications to access the company's AWS resources, Bob develops a custom identity broker application. The application verifies that employees are signed into the existing Example Corp. identity and authentication system, which might use LDAP, Active Directory, or another system. The identity broker application then obtains temporary security credentials for the employees. This scenario is similar to the previous one (a mobile app that uses a custom authentication system), except that the applications that need access to AWS resources all run within the corporate network, and the company has an existing authentication system.

To get temporary security credentials, the identity broker application calls either `AssumeRole` or `GetFederationToken` to obtain temporary security credentials, depending on how Bob wants to manage the policies for users and when the temporary credentials should expire. (For more information about the differences between these API operations, see [Temporary security credentials in IAM \(p. 301\)](#) and [Controlling permissions for temporary security credentials \(p. 316\)](#).) The call returns temporary security credentials consisting of an AWS access key ID, a secret access key, and a session token. The identity broker application makes these temporary security credentials available to the internal company application. The app can then use the temporary credentials to make calls to AWS directly. The app caches the credentials until they expire, and then requests a new set of temporary credentials. The following figure illustrates this scenario.



This scenario has the following attributes:

- The identity broker application has permissions to access IAM's token service (STS) API to create temporary security credentials.
- The identity broker application is able to verify that employees are authenticated within the existing authentication system.
- Users are able to get a temporary URL that gives them access to the AWS Management Console (which is referred to as single sign-on).

To see a sample application similar to the identity broker application that is described in this scenario, go to [Identity Federation Sample Application for an Active Directory Use Case](#) at [AWS Sample Code & Libraries](#). For information about creating temporary security credentials, see [Requesting temporary security credentials \(p. 303\)](#). For more information about federated users getting access to the AWS Management Console, see [Enabling SAML 2.0 federated users to access the AWS Management Console \(p. 203\)](#).

Identity providers and federation

If you already manage user identities outside of AWS, you can use IAM *identity providers* instead of creating IAM users in your AWS account. With an identity provider (IdP), you can manage your user identities outside of AWS and give these external user identities permissions to use AWS resources in your account. This is useful if your organization already has its own identity system, such as a corporate user directory. It is also useful if you are creating a mobile app or web application that requires access to AWS resources.

When you use an IAM identity provider, you don't have to create custom sign-in code or manage your own user identities. The IdP provides that for you. Your external users sign in through a well-known IdP, such as Login with Amazon, Facebook, or Google. You can give those external identities permissions to use AWS resources in your account. IAM identity providers help keep your AWS account secure because you don't have to distribute or embed long-term security credentials, such as access keys, in your application.

To use an IdP, you create an IAM identity provider entity to establish a trust relationship between your AWS account and the IdP. IAM supports IdPs that are compatible with [OpenID Connect \(OIDC\)](#) or [SAML 2.0 \(Security Assertion Markup Language 2.0\)](#). For more information about using one of these IdPs with AWS, see the following sections:

- [About web identity federation \(p. 177\)](#)
- [About SAML 2.0-based federation \(p. 182\)](#)

For details about creating the IAM identity provider entity to establish a trust relationship between a compatible IdP and AWS, see [Creating IAM identity providers \(p. 186\)](#)

About web identity federation

Imagine that you are creating a mobile app that accesses AWS resources, such as a game that runs on a mobile device and stores player and score information using Amazon S3 and DynamoDB.

When you write such an app, you'll make requests to AWS services that must be signed with an AWS access key. However, we **strongly recommend** that you do **not** embed or distribute long-term AWS credentials with apps that a user downloads to a device, even in an encrypted store. Instead, build your app so that it requests temporary AWS security credentials dynamically when needed using *web identity federation*. The supplied temporary credentials map to an AWS role that has only the permissions needed to perform the tasks required by the mobile app.

With web identity federation, you don't need to create custom sign-in code or manage your own user identities. Instead, users of your app can sign in using a well-known external identity provider (IdP), such as Login with Amazon, Facebook, Google, or any other [OpenID Connect \(OIDC\)](#)-compatible IdP. They can receive an authentication token, and then exchange that token for temporary security credentials in AWS that map to an IAM role with permissions to use the resources in your AWS account. Using an IdP helps you keep your AWS account secure, because you don't have to embed and distribute long-term security credentials with your application.

For most scenarios, we recommend that you use [Amazon Cognito](#) because it acts as an identity broker and does much of the federation work for you. For details, see the following section, [Using Amazon Cognito for mobile apps \(p. 177\)](#).

If you don't use Amazon Cognito, then you must write code that interacts with a web IdP, such as Facebook, and then calls the `AssumeRoleWithWebIdentity` API to trade the authentication token you get from those IdPs for AWS temporary security credentials. If you have already used this approach for existing apps, you can continue to use it.

Topics

- [Using Amazon Cognito for mobile apps \(p. 177\)](#)
- [Using web identity federation API operations for mobile apps \(p. 179\)](#)
- [Identifying users with web identity federation \(p. 180\)](#)
- [Additional resources for web identity federation \(p. 182\)](#)

Using Amazon Cognito for mobile apps

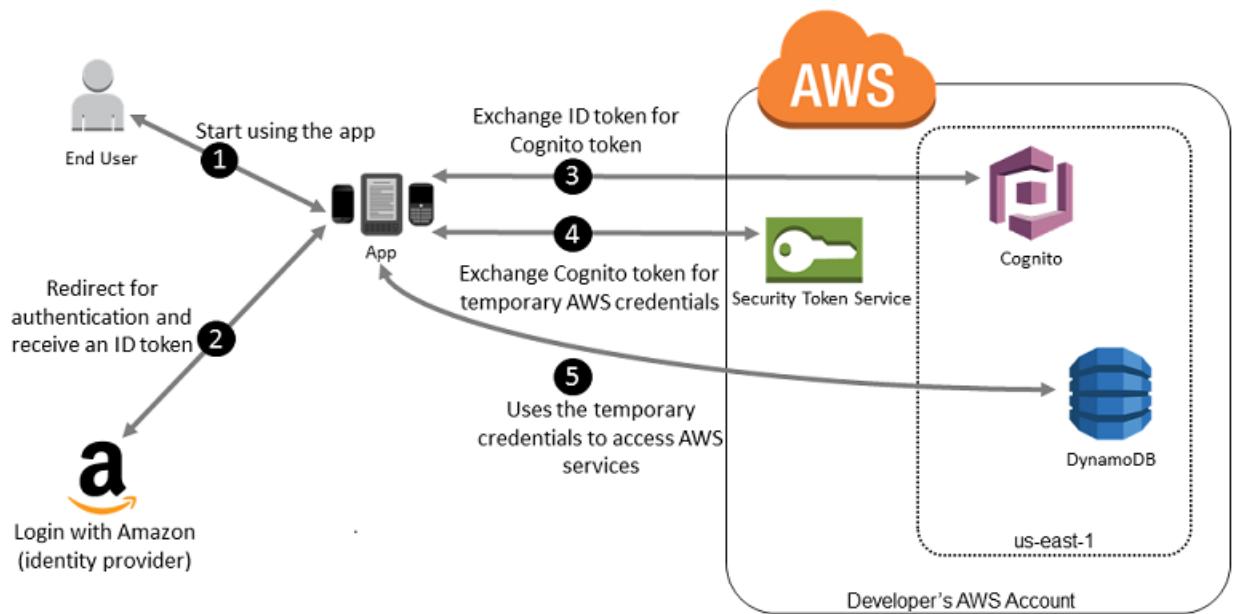
The preferred way to use web identity federation is to use [Amazon Cognito](#). For example, Adele the developer is building a game for a mobile device where user data such as scores and profiles is stored in Amazon S3 and Amazon DynamoDB. Adele could also store this data locally on the device and use Amazon Cognito to keep it synchronized across devices. She knows that for security and maintenance reasons, long-term AWS security credentials should not be distributed with the game. She also knows that the game might have a large number of users. For all of these reasons, she does not want to create new user identities in IAM for each player. Instead, she builds the game so that users can sign in using an identity that they've already established with a well-known external identity provider (IdP), such as [Login with Amazon](#).

with Amazon, Facebook, Google, or any **OpenID Connect** (OIDC)-compatible IdP. Her game can take advantage of the authentication mechanism from one of these providers to validate the user's identity.

To enable the mobile app to access her AWS resources, Adele first registers for a developer ID with her chosen IdPs. She also configures the application with each of these providers. In her AWS account that contains the Amazon S3 bucket and DynamoDB table for the game, Adele uses Amazon Cognito to create IAM roles that precisely define permissions that the game needs. If she is using an OIDC IdP, she also creates an IAM OIDC identity provider entity to establish trust between her AWS account and the IdP.

In the app's code, Adele calls the sign-in interface for the IdP that she configured previously. The IdP handles all the details of letting the user sign in, and the app gets an OAuth access token or OIDC ID token from the provider. Adele's app can trade this authentication information for a set of temporary security credentials that consist of an AWS access key ID, a secret access key, and a session token. The app can then use these credentials to access web services offered by AWS. The app is limited to the permissions that are defined in the role that it assumes.

The following figure shows a simplified flow for how this might work, using Login with Amazon as the IdP. For Step 2, the app can also use Facebook, Google, or any OIDC-compatible IdP, but that's not shown here.



1. A customer starts your app on a mobile device. The app asks the user to sign in.
2. The app uses Login with Amazon resources to accept the user's credentials.
3. The app uses Cognito API operations to exchange the Login with Amazon ID token for a Cognito token.
4. The app requests temporary security credentials from AWS STS, passing the Cognito token.
5. The temporary security credentials can be used by the app to access any AWS resources required by the app to operate. The role associated with the temporary security credentials and its assigned policies determines what can be accessed.

Use the following process to configure your app to use Amazon Cognito to authenticate users and give your app access to AWS resources. For specific steps to accomplish this scenario, consult the documentation for Amazon Cognito.

1. (Optional) Sign up as a developer with Login with Amazon, Facebook, Google, or any other OpenID Connect (OIDC)-compatible IdP and configure one or more apps with the provider. This step is optional because Amazon Cognito also supports unauthenticated (guest) access for your users.
2. Go to [Amazon Cognito in the AWS Management Console](#). Use the Amazon Cognito wizard to create an identity pool, which is a container that Amazon Cognito uses to keep end user identities organized for your apps. You can share identity pools between apps. When you set up an identity pool, Amazon Cognito creates one or two IAM roles (one for authenticated identities, and one for unauthenticated "guest" identities) that define permissions for Amazon Cognito users.
3. Download and integrate the [AWS SDK for iOS](#) or the [AWS SDK for Android](#) with your app, and import the files required to use Amazon Cognito.
4. Create an instance of the Amazon Cognito credentials provider, passing the identity pool ID, your AWS account number, and the Amazon Resource Name (ARN) of the roles that you associated with the identity pool. The Amazon Cognito wizard in the AWS Management Console provides sample code to help you get started.
5. When your app accesses an AWS resource, pass the credentials provider instance to the client object, which passes temporary security credentials to the client. The permissions for the credentials are based on the role or roles that you defined earlier.

For more information, see the following:

- [Amazon Cognito Identity in the AWS Mobile SDK for Android Developer Guide](#).
- [Amazon Cognito Identity in the AWS Mobile SDK for iOS Developer Guide](#).

Using web identity federation API operations for mobile apps

For best results, use Amazon Cognito as your identity broker for almost all web identity federation scenarios. Amazon Cognito is easy to use and provides additional capabilities like anonymous (unauthenticated) access, and synchronizing user data across devices and providers. However, if you have already created an app that uses web identity federation by manually calling the `AssumeRoleWithWebIdentity` API, you can continue to use it and your apps will still work fine.

Note

To help understand how web identity federation works, you can use the [Web Identity Federation Playground](#). This interactive website lets you walk through the process of authenticating via Login with Amazon, Facebook, or Google, getting temporary security credentials, and then using those credentials to make a request to AWS.

The process for using web identity federation **without** Amazon Cognito follows this general outline:

1. Sign up as a developer with the external identity provider (IdP) and configure your app with the IdP, who gives you a unique ID for your app. (Different IdPs use different terminology for this process. This outline uses the term *configure* for the process of identifying your app with the IdP.) Each IdP gives you an app ID that's unique to that IdP, so if you configure the same app with multiple IdPs, your app will have multiple app IDs. You can configure multiple apps with each provider.

The following external links provide information about using some of the commonly used identity providers (IdPs):

- [Login with Amazon Developer Center](#)
- [Add Facebook Login to Your App or Website](#) on the Facebook developers site.
- [Using OAuth 2.0 for Login \(OpenID Connect\)](#) on the Google developers site.

Note

Although Amazon Cognito and Google are based on OIDC technology, you don't have to create an IAM identity provider entity to use them. Support for Amazon Cognito and Google are built-in to AWS.

2. If you use an IdP that is compatible with OIDC, then create an IAM identity provider entity for it.
 3. In IAM, [create one or more roles \(p. 234\)](#). For each role, define who can assume the role (the trust policy) and what permissions the app's users are to have (the permissions policy). Typically, you create one role for each IdP that an app supports. For example, you might create a role that is assumed by an app when the user signs in through Login with Amazon, a second role for the same app where the user signs in through Facebook, and a third role for the app where the user signs in through Google. For the trust relationship, specify the IdP (like Amazon.com) as the Principal (the trusted entity), and include a Condition that matches the IdP assigned app ID. Examples of roles for different providers are described later in this topic.
 4. In your application, authenticate your users with the IdP. The specifics of how to do this vary both according to which IdP you're using (Login with Amazon, Facebook, or Google) and on which platform your app runs. For example, an Android app's method of authentication can differ from that of an iOS app or a JavaScript-based web app.
- Typically, if the user is not already signed in, the IdP takes care of displaying a sign-in page. After the IdP authenticates the user, the IdP returns an authentication token with information about the user to your app. The information included depends on what the IdP exposes and what information the user is willing to share. You can use this information in your app.
5. In your app, make an *unsigned* call to the `AssumeRoleWithWebIdentity` action to request temporary security credentials. In the request, you pass the IdP's authentication token and specify the Amazon Resource Name (ARN) for the IAM role that you created for that IdP. AWS verifies that the token is trusted and valid and if so, returns temporary security credentials to your app that have the permissions for the role that you name in the request. The response also includes metadata about the user from the IdP, such as the unique user ID that the IdP associates with the user.
 6. Using the temporary security credentials from the `AssumeRoleWithWebIdentity` response, your app makes signed requests to AWS API operations. The user ID information from the IdP can distinguish users in your app—for example, you can put objects into Amazon S3 folders that include the user ID as prefixes or suffixes. This lets you create access control policies that lock the folder so only the user with that ID can access it. For more information, see [Identifying users with web identity federation \(p. 180\)](#) later in this topic.
 7. Your app should cache the temporary security credentials so that you do not have to get new ones each time the app needs to make a request to AWS. By default, the credentials are good for one hour. When the credentials expire (or before then), you make another call to `AssumeRoleWithWebIdentity` to obtain a new set of temporary security credentials. Depending on the IdP and how they manage their tokens, you might have to refresh the IdP's token before you make a new call to `AssumeRoleWithWebIdentity`, since the IdP's tokens also usually expire after a fixed time. If you use the AWS SDK for iOS or the AWS SDK for Android, you can use the `AmazonSTSCredentialsProvider` action, which manages the IAM temporary credentials, including refreshing them as required.

Identifying users with web identity federation

When you create access policies in IAM, it's often useful to be able to specify permissions based on configured apps and on the ID of users who have authenticated using an external identity provider (IdP). For example, your mobile app that's using web identity federation might keep information in Amazon S3 using a structure like this:

```
myBucket/app1/user1
myBucket/app1/user2
myBucket/app1/user3
...
myBucket/app2/user1
myBucket/app2/user2
myBucket/app2/user3
...
```

You might also want to additionally distinguish these paths by provider. In that case, the structure might look like the following (only two providers are listed to save space):

```
myBucket/Amazon/app1/user1
myBucket/Amazon/app1/user2
myBucket/Amazon/app1/user3
...
myBucket/Amazon/app2/user1
myBucket/Amazon/app2/user2
myBucket/Amazon/app2/user3

myBucket/Facebook/app1/user1
myBucket/Facebook/app1/user2
myBucket/Facebook/app1/user3
...
myBucket/Facebook/app2/user1
myBucket/Facebook/app2/user2
myBucket/Facebook/app2/user3
...
```

For these structures, app1 and app2 represent different apps, such as different games, and each user of the app has a distinct folder. The values for app1 and app2 might be friendly names that you assign (for example, mynumbersgame) or they might be the app IDs that the providers assign when you configure your app. If you decide to include provider names in the path, those can also be friendly names like Cognito, Amazon, Facebook, and Google.

You can typically create the folders for app1 and app2 through the AWS Management Console, since the application names are static values. That's true also if you include the provider name in the path, since the provider name is also a static value. In contrast, the user-specific folders (**user1**, **user2**, **user3**, etc.) have to be created at run time from the app, using the user ID that's available in the `SubjectFromWebIdentityToken` value that is returned by the request to `AssumeRoleWithWebIdentity`.

To write policies that allow exclusive access to resources for individual users, you can match the complete folder name, including the app name and provider name, if you're using that. You can then include the following provider-specific context keys that reference the user ID that the provider returns:

- `cognito-identity.amazonaws.com:sub`
- `www.amazon.com:user_id`
- `graph.facebook.com:id`
- `accounts.google.com:sub`

For OIDC providers, use the fully qualified URL of the OIDC provider with the subcontext key, like the following example:

- `server.example.com:sub`

The following example shows a permission policy that grants access to a bucket in Amazon S3 only if the prefix for the bucket matches the string:

```
myBucket/Amazon/mynumbersgame/user1
```

The example assumes that the user is signed in using Login with Amazon, and that the user is using an app called mynumbersgame. The user's unique ID is presented as an attribute called `user_id`.

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

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

You would create similar policies for users who sign in using Amazon Cognito, Facebook, Google, or another OpenID Connect-compatible IdP. Those policies would use a different provider name as part of the path as well as different app IDs.

For more information about the web identity federation keys available for condition checks in policies, see [Available keys for AWS web identity federation \(p. 710\)](#).

Additional resources for web identity federation

The following resources can help you learn more about web identity federation:

- [Amazon Cognito Identity](#) in the *AWS Mobile SDK for Android Developer Guide* and [Amazon Cognito Identity](#) in the *AWS Mobile SDK for iOS Developer Guide*.
- The [Web Identity Federation Playground](#) is an interactive website that lets you walk through the process of authenticating via Login with Amazon, Facebook, or Google, getting temporary security credentials, and then using those credentials to make a request to AWS.
- The entry [Web Identity Federation using the AWS SDK for .NET](#) on the AWS .NET Development blog walks through how to use web identity federation with Facebook and includes code snippets in C# that show how to call `AssumeRoleWithWebIdentity` and how to use the temporary security credentials from that API call to access an S3 bucket.
- The [AWS SDK for iOS](#) and the [AWS SDK for Android](#) contain sample apps. These apps include code that shows how to invoke the identity providers and then how to use the information from these providers to get and use temporary security credentials.
- The article [Web Identity Federation with Mobile Applications](#) discusses web identity federation and shows an example of how to use web identity federation to get access to content in Amazon S3.

About SAML 2.0-based federation

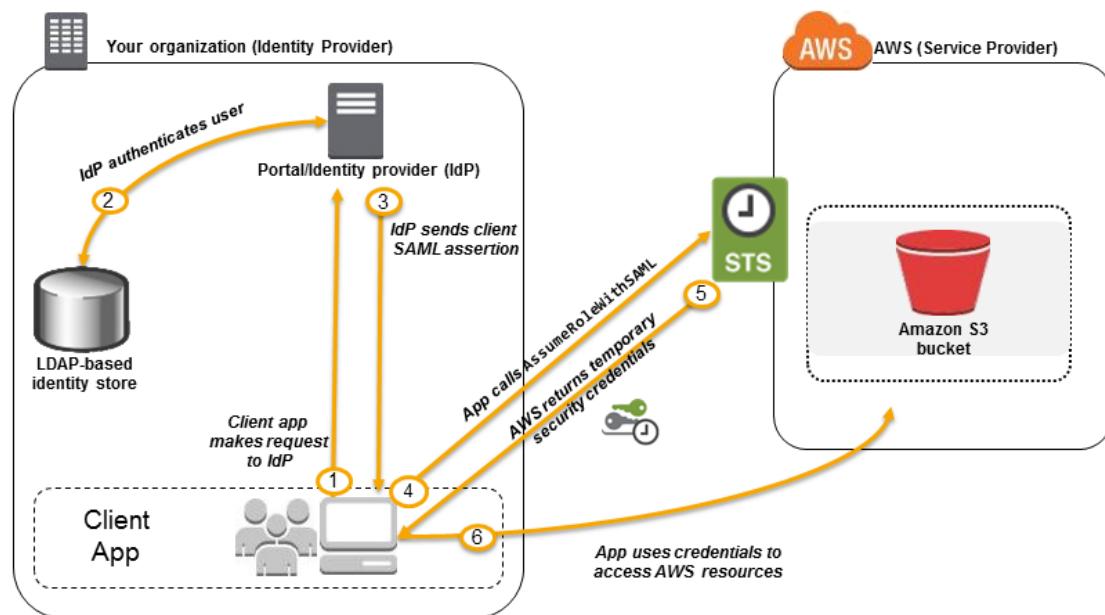
AWS supports identity federation with [SAML 2.0 \(Security Assertion Markup Language 2.0\)](#), an open standard that many identity providers (IdPs) use. This feature enables federated single sign-on (SSO), so users can log into the AWS Management Console or call the AWS API operations without you having to create an IAM user for everyone in your organization. By using SAML, you can simplify the process of configuring federation with AWS, because you can use the IdP's service instead of [writing custom identity proxy code](#).

IAM federation supports these use cases:

- **Federated access to allow a user or application in your organization to call AWS API operations (p. 183).** You use a SAML assertion (as part of the authentication response) that is generated in your organization to get temporary security credentials. This scenario is similar to other federation scenarios that IAM supports, like those described in [Requesting temporary security credentials \(p. 303\)](#) and [About web identity federation \(p. 177\)](#). However, SAML 2.0-based IdPs in your organization handle many of the details at run time for performing authentication and authorization checking. This is the scenario discussed in this topic.
- **Web-based single sign-on (SSO) to the AWS Management Console from your organization (p. 203).** Users can sign in to a portal in your organization hosted by a SAML 2.0-compatible IdP, select an option to go to AWS, and be redirected to the console without having to provide additional sign-in information. You can use a third-party SAML IdP to establish SSO access to the console or you can create a custom IdP to enable console access for your external users. For more information about building a custom IdP, see [Enabling custom identity broker access to the AWS console \(p. 206\)](#).

Using SAML-based federation for API access to AWS

Assume that you want to provide a way for employees to copy data from their computers to a backup folder. You build an application that users can run on their computers. On the back end, the application reads and writes objects in an S3 bucket. Users don't have direct access to AWS. Instead, the following process is used:



1. A user in your organization uses a client app to request authentication from your organization's IdP.
2. The IdP authenticates the user against your organization's identity store.
3. The IdP constructs a SAML assertion with information about the user and sends the assertion to the client app.
4. The client app calls the AWS STS [AssumeRoleWithSAML](#) API, passing the ARN of the SAML provider, the ARN of the role to assume, and the SAML assertion from IdP.
5. The API response to the client app includes temporary security credentials.
6. The client app uses the temporary security credentials to call Amazon S3 API operations.

Overview of configuring SAML 2.0-based federation

Before you can use SAML 2.0-based federation as described in the preceding scenario and diagram, you must configure your organization's IdP and your AWS account to trust each other. The general process for configuring this trust is described in the following steps. Inside your organization, you must have an [IdP that supports SAML 2.0 \(p. 197\)](#), like Microsoft Active Directory Federation Service (AD FS, part of Windows Server), Shibboleth, or another compatible SAML 2.0 provider.

To configure your organization's IdP and AWS to trust each other

1. You begin by registering AWS with your IdP. In your organization's IdP you register AWS as a service provider (SP) by using the SAML metadata document that you get from the following URL:

`https://signin.aws.amazon.com/static/saml-metadata.xml`
2. Using your organization's IdP, you generate an equivalent metadata XML file that can describe your IdP as an IAM identity provider in AWS. It must include the issuer name, a creation date, an expiration date, and keys that AWS can use to validate authentication responses (assertions) from your organization.
3. In the IAM console, you create a SAML identity provider entity. As part of this process, you upload the SAML metadata document that was produced by the IdP in your organization in [Step 2](#). For more information, see [Creating IAM SAML identity providers \(p. 193\)](#).
4. In IAM, you create one or more IAM roles. In the role's trust policy, you set the SAML provider as the principal, which establishes a trust relationship between your organization and AWS. The role's permission policy establishes what users from your organization are allowed to do in AWS. For more information, see [Creating a role for a third-party Identity Provider \(federation\) \(p. 234\)](#).
5. In your organization's IdP, you define assertions that map users or groups in your organization to the IAM roles. Note that different users and groups in your organization might map to different IAM roles. The exact steps for performing the mapping depend on what IdP you're using. In the [earlier scenario \(p. 183\)](#) of an Amazon S3 folder for users, it's possible that all users will map to the same role that provides Amazon S3 permissions. For more information, see [Configuring SAML assertions for the authentication response \(p. 199\)](#).

If your IdP enables SSO to the AWS console, then you can configure the maximum duration of the console sessions. For more information, see [Enabling SAML 2.0 federated users to access the AWS Management Console \(p. 203\)](#).

Note

The AWS implementation of SAML 2.0 federation does not support encrypted SAML assertions between the IAM identity provider and AWS. However, the traffic between the customer's systems and AWS is transmitted over an encrypted (TLS) channel.

6. In the application that you're creating, you call the AWS Security Token Service `AssumeRoleWithSAML` API, passing it the ARN of the SAML provider you created in [Step 3](#), the ARN of the role to assume that you created in [Step 4](#), and the SAML assertion about the current user that you get from your IdP. AWS makes sure that the request to assume the role comes from the IdP referenced in the SAML provider.

For more information, see [AssumeRoleWithSAML](#) in the [AWS Security Token Service API Reference](#).

7. If the request is successful, the API returns a set of temporary security credentials, which your application can use to make signed requests to AWS. Your application has information about the current user and can access user-specific folders in Amazon S3, as described in the previous scenario.

Overview of the role to allow SAML-federated access to your AWS resources

The role or roles that you create in IAM define what federated users from your organization are allowed to do in AWS. When you create the trust policy for the role, you specify the SAML provider that you created earlier as the `Principal`. You can additionally scope the trust policy with a `Condition` to allow

only users that match certain SAML attributes to access the role. For example, you can specify that only users whose SAML affiliation is `staff` (as asserted by `https://openidp.feide.no`) are allowed to access the role, as illustrated by the following sample policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {"Federated": "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:saml-provider/ExampleOrgSSOProvider"},
            "Action": "sts:AssumeRoleWithSAML",
            "Condition": {
                "StringEquals": {
                    "saml:aud": "https://signin.aws.amazon.com/saml",
                    "saml:iss": "https://openidp.feide.no"
                },
                "ForAllValues:StringLike": {"saml:edupersonaffiliation": ["staff"]}
            }
        }
    ]
}
```

For more information about the SAML keys that you can check in a policy, see [Available keys for SAML-based AWS STS federation \(p. 713\)](#).

For the permission policy in the role, you specify permissions as you would for any role. For example, if users from your organization are allowed to administer Amazon Elastic Compute Cloud instances, you must explicitly allow Amazon EC2 actions in the permissions policy, such as those in the [AmazonEC2FullAccess](#) managed policy.

Uniquely identifying users in SAML-based federation

When you create access policies in IAM, it's often useful to be able to specify permissions based on the identity of users. For example, for users who have been federated using SAML, an application might want to keep information in Amazon S3 using a structure like this:

```
myBucket/app1/user1
myBucket/app1/user2
myBucket/app1/user3
```

You can create the bucket (`myBucket`) and folder (`app1`) through the Amazon S3 console or the AWS CLI, since those are static values. However, the user-specific folders (`user1`, `user2`, `user3`, etc.) have to be created at run time using code, since the value that identifies the user isn't known until the first time the user signs in through the federation process.

To write policies that reference user-specific details as part of a resource name, the user identity has to be available in SAML keys that can be used in policy conditions. The following keys are available for SAML 2.0-based federation for use in IAM policies. You can use the values returned by the following keys to create unique user identifiers for resources like Amazon S3 folders.

- `saml:namequalifier`. A hash value based on the concatenation of the `Issuer` response value (`saml:iss`) and a string with the AWS account ID and the friendly name (the last part of the ARN) of the SAML provider in IAM. The concatenation of the account ID and friendly name of the SAML provider is available to IAM policies as the key `saml:doc`. The account ID and provider name must be separated by a '/' as in "123456789012/provider_name". For more information, see the `saml:doc` key at [Available keys for SAML-based AWS STS federation \(p. 713\)](#).

The combination of `NameQualifier` and `Subject` can be used to uniquely identify a federated user. The following pseudocode shows how this value is calculated. In this pseudocode `+` indicates concatenation, `SHA1` represents a function that produces a message digest using SHA-1, and `Base64` represents a function that produces Base-64 encoded version of the hash output.

```
Base64 ( SHA1 ( "https://example.com/saml" + "123456789012" + "/" + "MySAMLIdP" ) )
```

For more information about the policy keys that are available for SAML-based federation, see [Available keys for SAML-based AWS STS federation \(p. 713\)](#).

- **saml:sub** (string). This is the subject of the claim, which includes a value that uniquely identifies an individual user within an organization (for example, `_cbb88bf52c2510eabe00c1642d4643f41430fe25e3`).
- **saml:sub_type** (string). This key can be **persistent**, **transient**, or the full Format URI from the Subject and NameID elements used in your SAML assertion. A value of **persistent** indicates that the value in **saml:sub** is the same for a user across all sessions. If the value is **transient**, the user has a different **saml:sub** value for each session. For information about the NameID element's Format attribute, see [Configuring SAML assertions for the authentication response \(p. 199\)](#).

The following example shows a permission policy that uses the preceding keys to grant permissions to a user-specific folder in Amazon S3. The policy assumes that the Amazon S3 objects are identified using a prefix that includes both **saml:namequalifier** and **saml:sub**. Notice that the Condition element includes a test to be sure that **saml:sub_type** is set to **persistent**. If it is set to **transient**, the **saml:sub** value for the user can be different for each session, and the combination of values should not be used to identify user-specific folders.

```
>{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject",
                "s3:DeleteObject"
            ],
            "Resource": [
                "arn:aws:s3:::exampleorgBucket/backup/${saml:namequalifier}/${saml:sub}",
                "arn:aws:s3:::exampleorgBucket/backup/${saml:namequalifier}/${saml:sub}/*"
            ],
            "Condition": {"StringEquals": {"saml:sub_type": "persistent"}}
        }
    ]
}
```

For more information about mapping assertions from the IdP to policy keys, see [Configuring SAML assertions for the authentication response \(p. 199\)](#).

Creating IAM identity providers

When you want to configure federation with an external identity provider (IdP) service, you create an **IAM identity provider** to inform AWS about the IdP and its configuration. This establishes "trust" between your AWS account and the IdP. The following topics include details about how to create an IAM identity provider for each of the IdP types.

Topics

- [Creating OpenID Connect \(OIDC\) identity providers \(p. 186\)](#)
- [Creating IAM SAML identity providers \(p. 193\)](#)

Creating OpenID Connect (OIDC) identity providers

IAM OIDC identity providers are entities in IAM that describe an external identity provider (IdP) service that supports the [OpenID Connect](#) (OIDC) standard, such as Google or Salesforce. You use an IAM OIDC

identity provider when you want to establish trust between an OIDC-compatible IdP and your AWS account. This is useful when creating a mobile app or web application that requires access to AWS resources, but you don't want to create custom sign-in code or manage your own user identities. For more information about this scenario, see the section called ["About web identity federation" \(p. 177\)](#).

You can create and manage an IAM OIDC identity provider using the AWS Management Console, the AWS Command Line Interface, the Tools for Windows PowerShell, or the IAM API.

After you create an IAM OIDC identity provider, you must create one or more IAM roles. A role is an identity in AWS that doesn't have its own credentials (as a user does). But in this context, a role is dynamically assigned to a federated user that is authenticated by your organization's IdP. The role permits your organization's IdP to request temporary security credentials for access to AWS. The policies assigned to the role determine what the federated users are allowed to do in AWS. To create a role for a third-party identity provider, see [Creating a role for a third-party Identity Provider \(federation\) \(p. 234\)](#).

Topics

- [Creating and managing an OIDC provider \(console\) \(p. 187\)](#)
- [Creating and managing an IAM OIDC identity provider \(AWS CLI\) \(p. 188\)](#)
- [Creating and managing an OIDC Identity Provider \(AWS API\) \(p. 189\)](#)
- [Obtaining the root CA thumbprint for an OpenID Connect Identity Provider \(p. 190\)](#)

Creating and managing an OIDC provider (console)

Follow these instructions to create and manage an IAM OIDC identity provider in the AWS Management Console.

To create an IAM OIDC identity provider (console)

1. Before you create an IAM OIDC identity provider, you must register your application with the IdP to receive a *client ID*. The client ID (also known as *audience*) is a unique identifier for your app that is issued to you when you register your app with the IdP. For more information about obtaining a client ID, see the documentation for your IdP.
2. Open the IAM console at <https://console.aws.amazon.com/iam/>.
3. In the navigation pane, choose **Identity providers**, and then choose **Add provider**.
4. For **Configure provider**, choose **OpenID Connect**.
5. For **Provider URL**, type the URL of the IdP. The URL must comply with these restrictions:
 - The URL is case-sensitive.
 - The URL must begin with **https://**.
 - Within your AWS account, each IAM OIDC identity provider must use a unique URL.
6. Choose **Get thumbprint** to verify the server certificate of your IdP. To learn how, see [Obtaining the root CA thumbprint for an OpenID Connect Identity Provider \(p. 190\)](#).
7. For **Audience**, type the client ID of the application that you registered with the IdP and received in Step 1, and that will make requests to AWS. If you have additional client IDs (also known as *audiences*) for this IdP, you can add them later on the provider detail page.
8. Verify the information that you have provided. When you are done choose **Add provider**.
9. Assign an IAM role to your identity provider to give external user identities managed by your identity provider permissions to access AWS resources in your account. To learn more about creating roles for identity federation, see [Creating a role for a third-party Identity Provider \(federation\) \(p. 234\)](#).

To add or remove a thumbprint for an IAM OIDC identity provider (console)

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.

2. In the navigation pane, choose **Identity providers**. Then choose the name of the IAM identity provider that you want to update.
3. In the **Thumbprints** section, choose **Manage**. To enter a new thumbprint value, choose **Add thumbprint**. To remove a thumbprint, choose **Remove** next to the thumbprint that you want to remove.

Note

An IAM OIDC identity provider must have at least one and can have a maximum of five thumbprints.

When you are done, choose **Save changes**.

To add an audience for an IAM OIDC identity provider (console)

1. In the navigation pane, choose **Identity providers**, then choose the name of the IAM identity provider that you want to update.
2. In the **Audiences** section, choose **Actions** and select **Add audience**.
3. Type the client ID of the application that you registered with the IdP and received in [Step 1](#), and that will make requests to AWS. Then choose **Add audiences**.

Note

An IAM OIDC identity provider must have at least one and can have a maximum of 100 audiences.

To remove an audience for an IAM OIDC identity provider (console)

1. In the navigation pane, choose **Identity providers**, then choose the name of the IAM identity provider that you want to update.
2. In the **Audiences** section, select the radio button next to the audience that you want to remove, then select **Actions**.
3. Choose **Remove audience**. A new window opens.
4. If you remove an audience, identities federating with the audience cannot assume roles associated with the audience. In the window, read the warning and confirm that you want to remove the audience by typing the word `remove` in the field.
5. Choose **Remove** to remove the audience.

To delete an IAM OIDC identity provider (console)

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Identity providers**.
3. Select the check box next to the IAM identity provider that you want to delete. A new window opens.
4. Confirm that you want to delete the provider by typing the word `delete` in the field. Then, choose **Delete**.

Creating and managing an IAM OIDC identity provider (AWS CLI)

You can use the following AWS CLI commands to create and manage IAM OIDC identity providers.

To create an IAM OIDC identity provider (AWS CLI)

1. (Optional) To get a list of all the IAM OIDC identity providers in your AWS account, run the following command:
 - `aws iam list-open-id-connect-providers`

2. To create a new IAM OIDC identity provider, run the following command:

- `aws iam create-open-id-connect-provider`

To update the list of server certificate thumbprints for an existing IAM OIDC identity provider (AWS CLI)

- To update the list of server certificate thumbprints for an IAM OIDC identity provider, run the following command:

- `aws iam update-open-id-connect-provider-thumbprint`

To add or remove a client ID from an existing IAM OIDC identity provider (AWS CLI)

1. (Optional) To get a list of all the IAM OIDC identity provider in your AWS account, run the following command:

- `aws iam list-open-id-connect-providers`

2. (Optional) To get detailed information about an IAM OIDC identity provider, run the following command:

- `aws iam get-open-id-connect-provider`

3. To add a new client ID to an existing IAM OIDC identity provider, run the following command:

- `aws iam add-client-id-to-open-id-connect-provider`

4. To remove a client from an existing IAM OIDC identity provider, run the following command:

- `aws iam remove-client-id-from-open-id-connect-provider`

To delete an IAM OIDC identity provider (AWS CLI)

1. (Optional) To get a list of all the IAM OIDC identity provider in your AWS account, run the following command:

- `aws iam list-open-id-connect-providers`

2. (Optional) To get detailed information about an IAM OIDC identity provider, run the following command:

- `aws iam get-open-id-connect-provider`

3. To delete an IAM OIDC identity provider, run the following command:

- `aws iam delete-open-id-connect-provider`

Creating and managing an OIDC Identity Provider (AWS API)

You can use the following IAM API commands to create and manage OIDC providers.

To create an IAM OIDC identity provider (AWS API)

1. (Optional) To get a list of all the IAM OIDC identity provider in your AWS account, call the following operation:

- `ListOpenIDConnectProviders`

2. To create a new IAM OIDC identity provider, call the following operation:

- [CreateOpenIDConnectProvider](#)

To update the list of server certificate thumbprints for an existing IAM OIDC identity provider (AWS API)

- To update the list of server certificate thumbprints for an IAM OIDC identity provider, call the following operation:
 - [UpdateOpenIDConnectProviderThumbprint](#)

To add or remove a client ID from an existing IAM OIDC identity provider (AWS API)

1. (Optional) To get a list of all the IAM OIDC identity provider in your AWS account, call the following operation:
 - [ListOpenIDConnectProviders](#)
2. (Optional) To get detailed information about an IAM OIDC identity provider, call the following operation:
 - [GetOpenIDConnectProvider](#)
3. To add a new client ID to an existing IAM OIDC identity provider, call the following operation:
 - [AddClientIDToOpenIDConnectProvider](#)
4. To remove a client ID from an existing IAM OIDC identity provider, call the following operation:
 - [RemoveClientIDFromOpenIDConnectProvider](#)

To delete an IAM OIDC identity provider (AWS API)

1. (Optional) To get a list of all the IAM OIDC identity provider in your AWS account, call the following operation:
 - [ListOpenIDConnectProviders](#)
2. (Optional) To get detailed information about an IAM OIDC identity provider, call the following operation:
 - [GetOpenIDConnectProvider](#)
3. To delete an IAM OIDC identity provider, call the following operation:
 - [DeleteOpenIDConnectProvider](#)

Obtaining the root CA thumbprint for an OpenID Connect Identity Provider

When you [create an OpenID Connect \(OIDC\) identity provider \(p. 186\)](#) in IAM, you must supply a thumbprint. IAM requires the thumbprint for the root or intermediate certificate authority (CA) that signed the certificate used by the external identity provider (IdP). The thumbprint is a signature for the CA's certificate that was used to issue the certificate for the OIDC-compatible IdP. When you create an IAM OIDC identity provider, you are trusting identities authenticated by that IdP to have access to your AWS account. By supplying the CA's certificate thumbprint, you trust any certificate issued by that CA with the same DNS name as the one registered. This eliminates the need to update trusts in each account when you renew the IdP's signing certificate.

Important

In most cases, the federation server uses two different certificates. The first establishes an HTTPS connection between the clients and the federation endpoint. This can be safely issued

by a public root CA, such as AWS Certificate Manager. The second is used to sign tokens. We recommend that you issue this using a private CA.

You can create an IAM OIDC identity provider with [the AWS Command Line Interface, the Tools for Windows PowerShell, or the IAM API \(p. 188\)](#). When you use these methods, you must obtain the thumbprint manually and supply it to AWS. When you create an OIDC identity provider with [the IAM console \(p. 186\)](#), the console attempts to fetch the thumbprint for you. We recommend that you also obtain the thumbprint for your OIDC IdP manually and verify that the console fetched the correct thumbprint.

You use a web browser and the OpenSSL command line tool to obtain the thumbprint for an OIDC provider. For more information, see the following sections.

To obtain the thumbprint for an OIDC IdP

1. Before you can obtain the thumbprint for an OIDC IdP, you need to obtain the OpenSSL command-line tool. You use this tool to download the OIDC IdP's certificate chain and produce a thumbprint of the final certificate in the certificate chain. If you need to install and configure OpenSSL, follow the instructions at [Install OpenSSL \(p. 192\)](#) and [Configure OpenSSL \(p. 193\)](#).
2. Start with the OIDC IdP's URL (for example, `https://server.example.com`), and then add `/.well-known/openid-configuration` to form the URL for the IdP's configuration document, such as the following:

```
https://server.example.com/.well-known/openid-configuration
```

Open this URL in a web browser, replacing `server.example.com` with your IdP's server name.

3. In the document displayed in your web browser, find "jwks_uri". (Use your web browser's **Find** feature to locate this text on the page.) Immediately following the text "jwks_uri" you will see a colon (:) followed by a URL. Copy the fully qualified domain name of the URL. Do not include the `https://` or any path that comes after the top-level domain.
4. Use the OpenSSL command line tool to execute the following command. Replace `keys.example.com` with the domain name you obtained in [Step 3](#).

```
openssl s_client -servername keys.example.com -showcerts -connect keys.example.com:443
```

5. In your command window, scroll up until you see a certificate similar to the following example. If you see more than one certificate, find the last certificate that is displayed (at the bottom of the command output). This will be the certificate of the root CA in the certificate authority chain.

```
-----BEGIN CERTIFICATE-----
MIICiTCCAFICCQD6m7oRw0uXOjANBgkqhkiG9w0BAQUFADCBiDELMAkGA1UEBhMC
VVMxszAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGx1M0Q8wDQYDVQQKEwZBbWF6
b24xFDASBgNVBAstC01BTSDb25zb2x1MRIwEAYDVQQDEwlUZXN0Q21sYWMrHzAd
BgkqhkiG9w0BCQEWEVG5vb25lQGFTYXpvbi5jb20wHhcNMTEwNDI1MjaONTIxWhcN
MTIwNDI0MjaONTIxWjCBiDELMAkGA1UEBhMCVVMxszAJBgNVBAgTAldBMRAwDgYD
VQQHEwdTZWF0dGx1M0Q8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAstC01BTSDb25z
b2x1MRIwEAYDVQQDEwlUZXN0Q21sYWMrHzAdBgkqhkiG9w0BCQEWEVG5vb25lQGFT
YXpvbi5jb20wgZ8wDQYJKoZIhvcNAQEBQADgY0AMIGJAoGBAMak0dn+a4GmWIJ
21uUSfwfEvySWtC2XADZ4nB+BLYgVIk60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9T
rDHudUZg3qX4walG5M43q7Wgc/MbQITxOUSQv7c7ugFFDzQGBzzswY6786m86gpE
Ibb3OhjZnzcvQAArHhd1QWIIm2nrAgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4
nUhVVxYUntneD9+h8Mg9q6q+auNKyExzyLwax1Aoo7TJHidbtS4J5iNmZgXL0Fkb
FFBjvsfpJlJ00zbhNYS5f6GuoEDmFJ10ZxBHjJnyp3780D8uTs7fLvjx79LjSTb
NYiytvBZPQUQ5Yaxu2jXnimvw3rrszlaEXAMPLE=
-----END CERTIFICATE-----
```

Copy the certificate (including the -----BEGIN CERTIFICATE----- and -----END CERTIFICATE----- lines) and paste it into a text file. Then save the file with the file name `certificate.crt`.

6. Use the OpenSSL command-line tool to execute the following command.

```
openssl x509 -in certificate.crt -fingerprint -noout
```

Your command window displays the certificate thumbprint, which looks similar to the following example:

```
SHA1 Fingerprint=99:0F:41:93:97:2F:2B:EC:F1:2D:DE:DA:52:37:F9:C9:52:F2:0D:9E
```

Remove the colon characters (:) from this string to produce the final thumbprint, like this:

```
990F4193972F2BECF12DDEDA5237F9C952F20D9E
```

7. If you are creating the IAM OIDC identity provider with the AWS CLI, Tools for Windows PowerShell, or the IAM API, supply this thumbprint when creating the provider.

If you are creating the IAM OIDC identity provider in the IAM console, compare this thumbprint to the thumbprint shown on the console **Verify Provider Information** page when you create an OIDC provider.

Important

If the thumbprint you obtained does not match the one you see in the console, you should not create the OIDC provider in the console. Instead, you should wait a while and then try again to create the OIDC provider, ensuring that the thumbprints match before you create the provider. If the thumbprints still do not match after a second attempt, use the [IAM Forum](#) to contact AWS.

Install OpenSSL

If you don't already have OpenSSL installed, follow the instructions in this section.

To install OpenSSL on Linux or Unix

1. Go to [OpenSSL: Source, Tarballs](#) (<https://openssl.org/source/>).
2. Download the latest source and build the package.

To install OpenSSL on Windows

1. Go to [OpenSSL: Binary Distributions](#) (<https://wiki.openssl.org/index.php/Binaries>) for a list of sites from which you can install the Windows version.
2. Follow the instructions on your selected site to start the installation.
3. If you are asked to install the **Microsoft Visual C++ 2008 Redistributables** and it is not already installed on your system, choose the download link appropriate for your environment. Follow the instructions provided by the **Microsoft Visual C++ 2008 Redistributable Setup Wizard**.

Note

If you are not sure whether the Microsoft Visual C++ 2008 Redistributables is already installed on your system, you can try installing OpenSSL first. The OpenSSL installer displays an alert if the Microsoft Visual C++ 2008 Redistributables is not yet installed. Make sure that you install the architecture (32-bit or 64-bit) that matches the version of OpenSSL that you install.

4. After you have installed the Microsoft Visual C++ 2008 Redistributables, select the appropriate version of the OpenSSL binaries for your environment and save the file locally. Start the **OpenSSL Setup Wizard**.
5. Follow the instructions described in the **OpenSSL Setup Wizard**.

Configure OpenSSL

Before you use OpenSSL commands, you must configure the operating system so that it has information about the location where OpenSSL is installed.

To configure OpenSSL on Linux or Unix

- At the command line, set the `OpenSSL_HOME` variable to the location of the OpenSSL installation:

```
$ export OpenSSL_HOME=path_to_your_OpenSSL_installation
```

- Set the path to include the OpenSSL installation:

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

Note

Any changes you make to environment variables with the `export` command are valid only for the current session. You can make persistent changes to the environment variables by setting them in your shell configuration file. For more information, see the documentation for your operating system.

To configure OpenSSL on Windows

- Open a **Command Prompt** window.
- Set the `OpenSSL_HOME` variable to the location of the OpenSSL installation:

```
C:\> set OpenSSL_HOME=path_to_your_OpenSSL_installation
```

- Set the `OpenSSL_CONF` variable to the location of the configuration file in your OpenSSL installation:

```
C:\> set OpenSSL_CONF=path_to_your_OpenSSL_installation\bin\openssl.cfg
```

- Set the path to include the OpenSSL installation:

```
C:\> set Path=%Path%;%OpenSSL_HOME%\bin
```

Note

Any changes you make to Windows environment variables in a **Command Prompt** window are valid only for the current command line session. You can make persistent changes to the environment variables by setting them as system properties. The exact procedures depend 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.

Creating IAM SAML identity providers

An IAM SAML 2.0 identity provider is an entity in IAM that describes an external identity provider (IdP) service that supports the [SAML 2.0 \(Security Assertion Markup Language 2.0\)](#) standard. You use an IAM identity provider when you want to establish trust between a SAML-compatible IdP such as Shibboleth or Active Directory Federation Services and AWS, so that users in your organization can access AWS resources. IAM SAML identity providers are used as principals in an IAM trust policy.

For more information about this scenario, see [About SAML 2.0-based federation \(p. 182\)](#).

You can create and manage an IAM identity provider in the AWS Management Console or with AWS CLI, Tools for Windows PowerShell, or AWS API calls.

After you create a SAML provider, you must create one or more IAM roles. A role is an identity in AWS that doesn't have its own credentials (as a user does). But in this context, a role is dynamically assigned to a federated user that is authenticated by your organization's IdP. The role permits your organization's IdP to request temporary security credentials for access to AWS. The policies assigned to the role determine what the federated users are allowed to do in AWS. To create a role for SAML federation, see [Creating a role for a third-party Identity Provider \(federation\) \(p. 234\)](#).

Finally, after you create the role, you complete the SAML trust by configuring your IdP with information about AWS and the roles that you want your federated users to use. This is referred to as configuring relying party trust between your IdP and AWS. To configure relying party trust, see [Configuring your SAML 2.0 IdP with relying party trust and adding claims \(p. 196\)](#).

Topics

- [Creating and managing an IAM identity provider \(console\) \(p. 194\)](#)
- [Creating and managing an IAM SAML Identity Provider \(AWS CLI\) \(p. 195\)](#)
- [Creating and managing an IAM SAML identity provider \(AWS API\) \(p. 195\)](#)
- [Configuring your SAML 2.0 IdP with relying party trust and adding claims \(p. 196\)](#)
- [Integrating third-party SAML solution providers with AWS \(p. 197\)](#)
- [Configuring SAML assertions for the authentication response \(p. 199\)](#)

[Creating and managing an IAM identity provider \(console\)](#)

You can use the AWS Management Console to create and delete IAM SAML identity providers.

To create an IAM SAML identity provider (console)

1. Before you can create an IAM SAML identity provider, you need the SAML metadata document that you get from the IdP. This document includes the issuer's name, expiration information, and keys that can be used to validate the SAML authentication response (assertions) that are received from the IdP. To generate the metadata document, use the identity management software your organization uses as its IdP. For instructions on how to configure many of the available IdPs to work with AWS, including how to generate the required SAML metadata document, see [Integrating third-party SAML solution providers with AWS \(p. 197\)](#).

Important

The metadata file must be encoded in UTF-8 format without a byte order mark (BOM). Also, the X.509 certificate that is included as part of the SAML metadata document must use a key size of at least 1024 bits. If the key size is smaller, the IdP creation fails with an "Unable to parse metadata" error. To remove the BOM, you can encode the file as UTF-8 using a text editing tool, such as Notepad++.

2. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
3. In the navigation pane, choose **Identity providers** and then choose **Add provider**.
4. For **Configure provider**, choose **SAML**.
5. Type a name for the identity provider.
6. For **Metadata document**, choose **Choose file**, specify the SAML metadata document that you downloaded in Step 1.
7. Verify the information that you have provided. When you are done, choose **Add provider**.
8. Assign an IAM role to your identity provider to give external user identities managed by your identity provider permissions to access AWS resources in your account. To learn more about creating roles for identity federation, see [Creating a role for a third-party Identity Provider \(federation\) \(p. 234\)](#).

To delete a SAML provider (console)

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, choose **Identity providers**.
3. Select the radio button next to the identity provider that you want to delete.
4. Choose **Delete**. A new window opens.
5. Confirm that you want to delete the provider by typing the word **delete** in the field. Then, choose **Delete**.

Creating and managing an IAM SAML Identity Provider (AWS CLI)

You can use the AWS CLI to create and manage SAML providers.

Before you can create an IAM identity provider, you need the SAML metadata document that you get from the IdP. This document includes the issuer's name, expiration information, and keys that can be used to validate the SAML authentication response (assertions) that are received from the IdP. To generate the metadata document, use the identity management software your organization uses as its IdP. For instructions on how to configure many of the available IdPs to work with AWS, including how to generate the required SAML metadata document, see [Integrating third-party SAML solution providers with AWS \(p. 197\)](#).

Important

The metadata file must be encoded in UTF-8 format without a byte order mark (BOM). Also, the X.509 certificate that is included as part of the SAML metadata document must use a key size of at least 1024 bits. If the key size is smaller, the IdP creation fails with an "Unable to parse metadata" error. To remove the BOM, you can encode the file as UTF-8 using a text editing tool, such as Notepad++.

To create an IAM identity provider and upload a metadata document (AWS CLI)

- Run this command: `aws iam create-saml-provider`

To upload a new metadata document for an IAM identity provider (AWS CLI)

- Run this command: `aws iam update-saml-provider`

To delete an IAM SAML identity provider (AWS CLI)

1. (Optional) To list information for all providers, such as the ARN, creation date, and expiration, run the following command:
 - `aws iam list-saml-providers`
2. (Optional) To get information about a specific provider, such as the ARN, creation date, and expiration, run the following command:
 - `aws iam get-saml-provider`
3. To delete an IAM identity provider, run the following command:
 - `aws iam delete-saml-provider`

Creating and managing an IAM SAML identity provider (AWS API)

You can use the AWS API to create and manage SAML providers.

Before you can create an IAM identity provider, you need the SAML metadata document that you get from the IdP. This document includes the issuer's name, expiration information, and keys that can be used to validate the SAML authentication response (assertions) that are received from the IdP. To generate the metadata document, use the identity management software your organization uses as its IdP. For instructions on how to configure many of the available IdPs to work with AWS, including how to generate the required SAML metadata document, see [Integrating third-party SAML solution providers with AWS \(p. 197\)](#).

Important

The metadata file must be encoded in UTF-8 format without a byte order mark (BOM). Also, the X.509 certificate that is included as part of the SAML metadata document must use a key size of at least 1024 bits. If the key size is smaller, the IdP creation fails with an "Unable to parse metadata" error. To remove the BOM, you can encode the file as UTF-8 using a text editing tool, such as Notepad++.

To create an IAM identity provider and upload a metadata document (AWS API)

- Call this operation: [CreateSAMLProvider](#)

To upload a new metadata document for an IAM identity provider (AWS API)

- Call this operation: [UpdateSAMLProvider](#)

To delete an IAM identity provider (AWS API)

1. (Optional) To list information for all IdPs, such as the ARN, creation date, and expiration, call the following operation:
 - [ListSAMLProviders](#)
2. (Optional) To get information about a specific provider, such as the ARN, creation date, and expiration, call the following operation:
 - [GetSAMLProvider](#)
3. To delete an IdP, call the following operation:
 - [DeleteSAMLProvider](#)

Configuring your SAML 2.0 IdP with relying party trust and adding claims

When you create an IAM identity provider and role for SAML access, you are telling AWS about the external identity provider (IdP) and what its users are allowed to do. Your next step is to then tell the IdP about AWS as a service provider. This is called adding *relying party trust* between your IdP and AWS. The exact process for adding relying party trust depends on what IdP you're using. For details, see the documentation for your identity management software.

Many IdPs allow you to specify a URL from which the IdP can read an XML document that contains relying party information and certificates. For AWS, you can use <https://signin.aws.amazon.com/static/saml-metadata.xml>

If you can't specify a URL directly, then download the XML document from the preceding URL and import it into your IdP software.

You also need to create appropriate claim rules in your IdP that specify AWS as a relying party. When the IdP sends a SAML response to the AWS endpoint, it includes a SAML *assertion* that contains one or more *claims*. A claim is information about the user and its groups. A claim rule maps that information into SAML attributes. This lets you make sure that SAML authentication responses from your IdP contain

the necessary attributes that AWS uses in IAM policies to check permissions for federated users. For more information, see the following topics:

- [Overview of the role to allow SAML-federated access to your AWS resources \(p. 184\)](#). This topic discusses using SAML-specific keys in IAM policies and how to use them to restrict permissions for SAML-federated users.
- [Configuring SAML assertions for the authentication response \(p. 199\)](#). This topic discusses how to configure SAML claims that include information about the user. The claims are bundled into a SAML assertion and included in the SAML response that is sent to AWS. You must ensure that the information needed by AWS policies is included in the SAML assertion in a form that AWS can recognize and use.
- [Integrating third-party SAML solution providers with AWS \(p. 197\)](#). This topic provides links to documentation provided by third-party organizations about how to integrate identity solutions with AWS.

[Integrating third-party SAML solution providers with AWS](#)

The following links help you configure third-party SAML 2.0 identity provider (IdP) solutions to work with AWS federation.

Note

AWS Support engineers can assist customers who have business and enterprise support plans with some integration tasks that involve third-party software. For a current list of supported platforms and applications, see [What third-party software is supported?](#) in the *AWS Support FAQs*.

Solution	More information
Auth0	Integrate with Amazon Web services – This page on the Auth0 documentation website has links to resources that describe how to set up single sign-on (SSO) with the AWS Management Console and includes a JavaScript example. You can configure Auth0 to pass session tags (p. 293) . For more information, see Auth0 Announces Partnership with AWS for IAM Session Tags .
Centrify	Configure Centrify and Use SAML for SSO to AWS – This page on the Centrify website explains how to configure Centrify to use SAML for SSO to AWS.
ForgeRock	The ForgeRock Identity Platform integrates with AWS. You can configure ForgeRock to pass session tags (p. 293) . For more information, see Attribute Based Access Control for Amazon Web Services .
Google Workspace	Amazon Web Services cloud application – This article on the Google Workspace Admin Help site describes how to configure Google Workspace as a SAML 2.0 IdP with AWS as the service provider.
IBM	You can configure IBM to pass session tags (p. 293) . For more information, see IBM Cloud Identity IDaaS one of first to support AWS session tags .
Matrix42	MyWorkspace Getting Started Guide – This guide describes how to integrate AWS identity services with Matrix42 MyWorkspace.

Solution	More information
Microsoft Active Directory Federation Services (AD FS)	<p>Enabling Federation to AWS Using Windows Active Directory, AD FS, and SAML 2.0 – This post on the AWS Security Blog shows how to set up AD FS on an EC2 instance and enable SAML federation with AWS. You can configure AD FS to pass session tags (p. 293). For more information, see Use attribute-based access control with AD FS to simplify IAM permissions management.</p> <p>PowerShell Automation to Give AWS Console Access – This post on Sivaprasad Padisetty's blog describes how to use Windows PowerShell to automate the process of setting up Active Directory and AD FS. It also covers enabling SAML federation with AWS.</p>
miniOrange	<p>SSO for AWS – This page on the miniOrange website describes how to establish secure access to AWS for enterprises and full control over access of AWS applications.</p>
Okta	<p>Integrating the Amazon Web Services Command Line Interface Using Okta – From this page on the Okta support site you can learn how to configure Okta for use with AWS. You can configure Okta to pass session tags (p. 293). For more information, see Okta and AWS Partner to Simplify Access Via Session Tags.</p>
Okta	<p>How to Configure SAML 2.0 for AWS Single Sign-On – This article on the Okta website describes how to set up and enable SSO for AWS.</p>
OneLogin	<p>From the OneLogin Knowledgebase, search for SAML AWS for a list of articles that explain how to set up AWS SSO functionality between OneLogin and AWS for a single-role and multi-role scenarios. You can configure OneLogin to pass session tags (p. 293). For more information, see OneLogin and Session Tags: Attribute-Based Access Control for AWS Resources.</p>
Ping Identity	<p>PingFederate AWS Connector – View details about the PingFederate AWS Connector, a quick connection template to easily set up a single sign-on (SSO) and provisioning connection. Read documentation and download the latest PingFederate AWS Connector for integrations with AWS. You can configure Ping Identity to pass session tags (p. 293). For more information, see Announcing Ping Identity Support for Attribute-Based Access Control in AWS.</p>
RadiantLogic	<p>Radiant Logic Technology Partners – Radiant Logic's RadiantOne Federated Identity Service integrates with AWS to provide an identity hub for SAML-based SSO.</p>
RSA	<p>RSA Link is an online community that facilitates information sharing and discussion. You can configure RSA to pass session tags (p. 293). For more information, see Simplify Identity Access and Assurance Decisions on AWS with RSA SecurID and Session Tags.</p>

Solution	More information
Salesforce.com	How to configure SSO from Salesforce to AWS – This how-to article on the Salesforce.com developer site describes how to set up an identity provider (IdP) in Salesforce and configure AWS as a service provider.
SecureAuth	AWS - SecureAuth SAML SSO – This article on the SecureAuth website describes how to set up SAML integration with AWS for a SecureAuth appliance.
Shibboleth	How to Use Shibboleth for SSO to the AWS Management Console – This entry on the AWS Security Blog provides a step-by-step tutorial on how to set up Shibboleth and configure it as an identity provider for AWS. You can configure Shibboleth to pass session tags (p. 293) .

For more details, see the [IAM Partners](#) page on the AWS website.

Configuring SAML assertions for the authentication response

In your organization, after a user's identity has been verified, the external identity provider (IdP) sends an authentication response to the AWS SAML endpoint at <https://signin.aws.amazon.com/saml>. This response is a POST request that includes a SAML token that adheres to the [HTTP POST Binding for SAML 2.0](#) standard and that contains the following elements, or *claims*. You configure these claims in your SAML-compatible IdP. Refer to the documentation for your IdP for instructions on how to enter these claims.

When the IdP sends the response containing the claims to AWS, many of the incoming claims map to AWS context keys. These context keys can be checked in IAM policies using the `Condition` element. A listing of the available mappings follows in the section [Mapping SAML attributes to AWS trust policy context keys \(p. 202\)](#).

Subject And NameID

The following excerpt shows an example. Substitute your own values for the marked ones. There must be exactly one `SubjectConfirmation` element with a `SubjectConfirmationData` element that includes both the `NotOnOrAfter` attribute and a `Recipient` attribute. These attributes include a value that must match the AWS endpoint (<https://signin.aws.amazon.com/saml>), as shown in the following example. For information about the name identifier formats supported for single sign-on interactions, see [Oracle Sun OpenSSO Enterprise Administration Reference](#).

```
<Subject>
  <NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-
format:persistent">\_cbb88bf52c2510eabe00c1642d4643f41430fe25e3</NameID>
  <SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
    <SubjectConfirmationData NotOnOrAfter="2013-11-05T02:06:42.876Z" Recipient="https://
    signin.aws.amazon.com/saml"/>
  </SubjectConfirmation>
</Subject>
```

AudienceRestriction And Audience

For security reasons, AWS should be included as an audience in the SAML assertion your IdP sends to AWS. For the value of the `Audience` element, specify either <https://signin.aws.amazon.com/saml> or `urn:amazon:webservices`. The following sample XML snippets from SAML assertions show how this key can be specified by the IdP. Include whichever sample applies to your use case.

```
<Conditions>
<AudienceRestriction>
    <Audience>https://signin.aws.amazon.com/saml</Audience>
</AudienceRestriction>
</Conditions>
```

```
<Conditions>
<AudienceRestriction>
    <Audience>urn:amazon:webservices</Audience>
</AudienceRestriction>
</Conditions>
```

Important

The SAML AudienceRestriction value in the SAML assertion from the IdP does *not* map to the saml:aud context key that you can test in an IAM policy. Instead, the saml:aud context key comes from the SAML recipient attribute because it is the SAML equivalent to the OIDC audience field, for example, by accounts.google.com:aud.

SAML role Attribute

You can use an Attribute element with the Name attribute set to `https://aws.amazon.com/SAML/Attributes/Role`. This element contains one or more AttributeValue elements that list the IAM identity provider and role to which the user is mapped by your IdP. The IAM role and IAM identity provider are specified as a comma-delimited pair of ARNs in the same format as the RoleArn and PrincipalArn parameters that are passed to [AssumeRoleWithSAML](#). This element must contain at least one role-provider pair (AttributeValue element), and can contain multiple pairs. If the element contains multiple pairs, then the user is asked to select which role to assume when they use WebSSO to sign into the AWS Management Console.

Important

The value of the Name attribute in the Attribute tag is case-sensitive. It must be set to `https://aws.amazon.com/SAML/Attributes/Role` exactly.

```
<Attribute Name="https://aws.amazon.com/SAML/Attributes/Role">
    <AttributeValue>arn:aws:iam::account-number:role/role-name1,arn:aws:iam::account-
    number:saml-provider/provider-name</AttributeValue>
    <AttributeValue>arn:aws:iam::account-number:role/role-name2,arn:aws:iam::account-
    number:saml-provider/provider-name</AttributeValue>
    <AttributeValue>arn:aws:iam::account-number:role/role-name3,arn:aws:iam::account-
    number:saml-provider/provider-name</AttributeValue>
</Attribute>
```

SAML RoleSessionName Attribute

You can use an Attribute element with the Name attribute set to `https://aws.amazon.com/SAML/Attributes/RoleSessionName`. This element contains one AttributeValue element that provides an identifier for the AWS temporary credentials that are issued for SSO. This element is used to display user information in the AWS Management Console. The value in the AttributeValue element must be between 2 and 64 characters long, can contain only alphanumeric characters, underscores, and the following characters: + (plus sign), = (equals sign), , (comma), . (period), @ (at symbol), and - (hyphen). It cannot contain spaces. The value is typically a user ID (johndoe) or an email address (johndoe@example.com). It should not be a value that includes a space, like a user's display name (John Doe).

Important

The value of the Name attribute in the Attribute tag is case-sensitive. It must be set to `https://aws.amazon.com/SAML/Attributes/RoleSessionName` exactly.

```
<Attribute Name="https://aws.amazon.com/SAML/Attributes/RoleSessionName">
```

```
<AttributeValue>user-id-name</AttributeValue>
</Attribute>
```

SAML SessionDuration Attribute

(Optional) You can use an `Attribute` element with the `Name` attribute set to `https://aws.amazon.com/SAML/Attributes/SessionDuration`. This element contains one `AttributeValue` element that specifies how long the user can access the AWS Management Console before having to request new temporary credentials. The value is an integer representing the number of seconds for the session. The value can range from 900 seconds (15 minutes) to 43200 seconds (12 hours). If this attribute is not present, then the credential last for one hour (the default value of the `DurationSeconds` parameter of the `AssumeRoleWithSAML` API).

To use this attribute, you must configure the SAML provider to provide single sign-on access to the AWS Management Console through the console sign-in web endpoint at `https://signin.aws.amazon.com/saml`. Note that this attribute extends sessions only to the AWS Management Console. It cannot extend the lifetime of other credentials. However, if it is present in an `AssumeRoleWithSAML` API call, it can be used to *shorten* the duration of the session. The default lifetime of the credentials returned by the call is 60 minutes.

Note, too, that if a `SessionNotOnOrAfter` attribute is also defined, then the *lesser* value of the two attributes, `SessionDuration` or `SessionNotOnOrAfter`, establishes the maximum duration of the console session.

When you enable console sessions with an extended duration the risk of compromise of the credentials rises. To help you mitigate this risk, you can immediately disable the active console sessions for any role by choosing **Revoke Sessions** on the **Role Summary** page in the IAM console. For more information, see [Revoking IAM role temporary security credentials \(p. 271\)](#).

Important

The value of the `Name` attribute in the `Attribute` tag is case-sensitive. It must be set to `https://aws.amazon.com/SAML/Attributes/SessionDuration` exactly.

```
<Attribute Name="https://aws.amazon.com/SAML/Attributes/SessionDuration">
  <AttributeValue>1800</AttributeValue>
</Attribute>
```

SAML PrincipalTag Attribute

(Optional) You can use an `Attribute` element with the `Name` attribute set to `https://aws.amazon.com/SAML/Attributes/PrincipalTag:{TagKey}`. This element allows you to pass attributes as session tags in the SAML assertion. For more information about session tags, see [Passing session tags in AWS STS \(p. 293\)](#).

To pass attributes as session tags, include the `AttributeValue` element that specifies the value of the tag. For example, to pass the tag key-value pairs `Project = Marketing` and `CostCenter = 12345`, use the following attribute. Include a separate `Attribute` element for each tag.

```
<Attribute Name="https://aws.amazon.com/SAML/Attributes/PrincipalTag:Project">
  <AttributeValue>Marketing</AttributeValue>
</Attribute>
<Attribute Name="https://aws.amazon.com/SAML/Attributes/PrincipalTag:CostCenter">
  <AttributeValue>12345</AttributeValue>
</Attribute>
```

To set the tags above as transitive, include another `Attribute` element with the `Name` attribute set to `https://aws.amazon.com/SAML/Attributes/TransitiveTagKeys`. This is an optional multivalued attribute that sets your session tags as transitive. Transitive tags persist when you use the SAML session to assume another role in AWS. This is known as [role chaining \(p. 169\)](#). For example, to

set both the `Principal` and `CostCenter` tags as transitive, use the following attribute to specify the keys.

```
<Attribute Name="https://aws.amazon.com/SAML/Attributes/TransitiveTagKeys">
  <AttributeValue>Project</AttributeValue>
  <AttributeValue>CostCenter</AttributeValue>
</Attribute>
```

Mapping SAML attributes to AWS trust policy context keys

The tables in this section list commonly used SAML attributes and how they map to trust policy condition context keys in AWS. You can use these keys to control access to a role. To do that, compare the keys to the values that are included in the assertions that accompany a SAML access request.

Important

These keys are available only in IAM trust policies (policies that determine who can assume a role) and are not applicable to permissions policies.

In the `eduPerson` and `eduOrg` attributes table, values are typed either as strings or as lists of strings. For string values, you can test these values in IAM trust policies using `StringEquals` or `StringLike` conditions. For values that contain a list of strings, you can use the `ForAnyValue` and `ForAllValues` policy set operators (p. 652) to test the values in trust policies.

Note

You should include only one claim per AWS context key. If you include more than one, only one claim will be mapped.

eduPerson and eduOrg attributes

eduPerson or eduOrg attribute (<code>Name</code> key)	Maps to this AWS context key (<code>FriendlyName</code> key)	Type
<code>urn:oid:1.3.6.1.4.1.5923.1.1.1.1</code>	<code>eduPersonAffiliation</code>	List of strings
<code>urn:oid:1.3.6.1.4.1.5923.1.1.1.2</code>	<code>eduPersonNickname</code>	List of strings
<code>urn:oid:1.3.6.1.4.1.5923.1.1.1.3</code>	<code>eduPersonOrgDN</code>	String
<code>urn:oid:1.3.6.1.4.1.5923.1.1.1.4</code>	<code>eduPersonOrgUnitDN</code>	List of strings
<code>urn:oid:1.3.6.1.4.1.5923.1.1.1.5</code>	<code>eduPersonPrimaryAffiliation</code>	String
<code>urn:oid:1.3.6.1.4.1.5923.1.1.1.6</code>	<code>eduPersonPrincipalName</code>	String
<code>urn:oid:1.3.6.1.4.1.5923.1.1.1.7</code>	<code>eduPersonEntitlement</code>	List of strings
<code>urn:oid:1.3.6.1.4.1.5923.1.1.1.8</code>	<code>eduPersonPrimaryOrgUnitDN</code>	String
<code>urn:oid:1.3.6.1.4.1.5923.1.1.1.9</code>	<code>eduPersonScopedAffiliation</code>	List of strings
<code>urn:oid:1.3.6.1.4.1.5923.1.1.1.10</code>	<code>eduPersonTargetedID</code>	List of strings
<code>urn:oid:1.3.6.1.4.1.5923.1.1.1.11</code>	<code>eduPersonAssurance</code>	List of strings
<code>urn:oid:1.3.6.1.4.1.5923.1.2.1.2</code>	<code>eduOrgHomePageURI</code>	List of strings
<code>urn:oid:1.3.6.1.4.1.5923.1.2.1.3</code>	<code>eduOrgIdentityAuthNPolicy</code>	List of strings
<code>urn:oid:1.3.6.1.4.1.5923.1.2.1.4</code>	<code>eduOrgLegalName</code>	List of strings
<code>urn:oid:1.3.6.1.4.1.5923.1.2.1.5</code>	<code>eduOrgSuperiorURI</code>	List of strings
<code>urn:oid:1.3.6.1.4.1.5923.1.2.1.6</code>	<code>eduOrgWhitePagesURI</code>	List of strings

eduPerson or eduOrg attribute (Name key)	Maps to this AWS context key (FriendlyName key)	Type
urn:oid:2.5.4.3	cn	List of strings

Active Directory attributes

AD attribute	Maps to this AWS context key	Type
http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name	name	String
http://schemas.xmlsoap.org/claims/CommonName	commonName	String
http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname	givenName	String
http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname	surname	String
http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress	mail	String
http://schemas.microsoft.com/ws/2008/06/identity/claims/primarygroupsid	uid	String

X.500 attributes

X.500 attribute	Maps to this AWS context key	Type
2.5.4.3	commonName	String
2.5.4.4	surname	String
2.4.5.42	givenName	String
2.5.4.45	x500UniqueIdentifier	String
0.9.2342.19200300100.1.1	uid	String
0.9.2342.19200300100.1.3	mail	String
0.9.2342.19200300.100.1.45	organizationStatus	String

Enabling SAML 2.0 federated users to access the AWS Management Console

You can use a role to configure your SAML 2.0-compliant identity provider (IdP) and AWS to permit your federated users to access the AWS Management Console. The role grants the user permissions to carry out tasks in the console. If you want to give SAML federated users other ways to access AWS, see one of these topics:

- AWS CLI: [Switching to an IAM role \(AWS CLI\) \(p. 256\)](#)
- Tools for Windows PowerShell: [Switching to an IAM role \(Tools for Windows PowerShell\) \(p. 260\)](#)

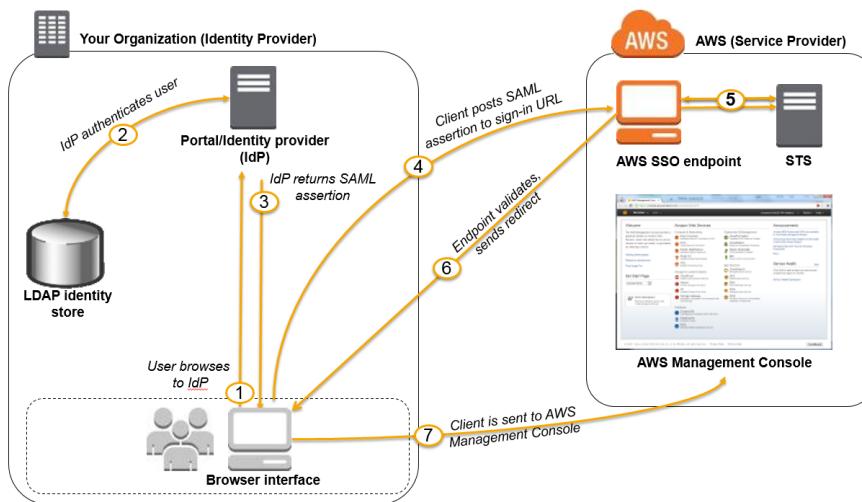
- AWS API: Switching to an IAM role (AWS API) (p. 262)

Overview

The following diagram illustrates the flow for SAML-enabled single sign-on.

Note

This specific use of SAML differs from the more general one illustrated at [About SAML 2.0-based federation \(p. 182\)](#) because this workflow opens the AWS Management Console on behalf of the user. This requires the use of the AWS SSO endpoint instead of directly calling the `AssumeRoleWithSAML` API. The endpoint calls the API for the user and returns a URL that automatically redirects the user's browser to the AWS Management Console.



The diagram illustrates the following steps:

1. The user browses to your organization's portal and selects the option to go to the AWS Management Console. In your organization, the portal is typically a function of your IdP that handles the exchange of trust between your organization and AWS. For example, in Active Directory Federation Services, the portal URL is: `https://ADFSServiceName/adfs/ls/IdpInitiatedSignOn.aspx`
2. The portal verifies the user's identity in your organization.
3. The portal generates a SAML authentication response that includes assertions that identify the user and include attributes about the user. You can also configure your IdP to include a SAML assertion attribute called `SessionDuration` that specifies how long the console session is valid. You can also configure the IdP to pass attributes as [session tags \(p. 293\)](#). The portal sends this response to the client browser.
4. The client browser is redirected to the AWS single sign-on endpoint and posts the SAML assertion.
5. The endpoint requests temporary security credentials on behalf of the user and creates a console sign-in URL that uses those credentials.
6. AWS sends the sign-in URL back to the client as a redirect.
7. The client browser is redirected to the AWS Management Console. If the SAML authentication response includes attributes that map to multiple IAM roles, the user is first prompted to select the role for accessing the console.

From the user's perspective, the process happens transparently: The user starts at your organization's internal portal and ends up at the AWS Management Console, without ever having to supply any AWS credentials.

Consult the following sections for an overview of how to configure this behavior along with links to detailed steps.

Configure your network as a SAML provider for AWS

Inside your organization's network, you configure your identity store (such as Windows Active Directory) to work with a SAML-based IdP like Windows Active Directory Federation Services, Shibboleth, etc. Using your IdP, you generate a metadata document that describes your organization as an IdP and includes authentication keys. You also configure your organization's portal to route user requests for the AWS Management Console to the AWS SAML endpoint for authentication using SAML assertions. How you configure your IdP to produce the metadata.xml file depends on your IdP. Refer to your IdP's documentation for instructions, or see [Integrating third-party SAML solution providers with AWS \(p. 197\)](#) for links to the web documentation for many of the SAML providers supported.

Create a SAML provider in IAM

Next, you sign in to the AWS Management Console and go to the IAM console. There you create a new SAML provider, which is an entity in IAM that holds information about your organization's IdP. As part of this process, you upload the metadata document produced by the IdP software in your organization in the previous section. For details, see [Creating IAM SAML identity providers \(p. 193\)](#).

Configure permissions in AWS for your federated users

The next step is to create an IAM role that establishes a trust relationship between IAM and your organization's IdP. This role must identify your IdP as a principal (trusted entity) for purposes of federation. The role also defines what users authenticated by your organization's IdP are allowed to do in AWS. You can use the IAM console to create this role. When you create the trust policy that indicates who can assume the role, you specify the SAML provider that you created earlier in IAM. You also specify one or more SAML attributes that a user must match to be allowed to assume the role. For example, you can specify that only users whose SAML eduPersonOrgDN value is ExampleOrg are allowed to sign in. The role wizard automatically adds a condition to test the saml:aud attribute to make sure that the role is assumed only for sign-in to the AWS Management Console. The trust policy for the role might look like this:

```
{  
    "Version": "2012-10-17",  
    "Statement": [{  
        "Effect": "Allow",  
        "Principal": {"Federated": "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:saml-provider/ExampleOrgSSOPProvider"},  
        "Action": "sts:AssumeRoleWithSAML",  
        "Condition": {"StringEquals": {  
            "saml:edupersonorgdn": "ExampleOrg",  
            "saml:aud": "https://signin.aws.amazon.com/saml"  
        }}  
    }]  
}
```

For the [permission policy \(p. 351\)](#) in the role, you specify permissions as you would for any role, user, or group. For example, if users from your organization are allowed to administer Amazon EC2 instances, you explicitly allow Amazon EC2 actions in the permission policy. You can do this by assigning a [managed policy \(p. 454\)](#), such as the [Amazon EC2 Full Access](#) managed policy.

For details about creating a role for a SAML IdP, see [Creating a role for SAML 2.0 federation \(console\) \(p. 241\)](#).

Finish configuration and create SAML assertions

After you create the role, inform your SAML IdP about AWS as a service provider by installing the saml-metadata.xml file found at <https://signin.aws.amazon.com/static/saml-metadata.xml>. How you install that file depends on your IdP. Some providers give you the option to type the URL, whereupon the IdP

gets and installs the file for you. Others require you to download the file from the URL and then provide it as a local file. Refer to your IdP documentation for details, or see [Integrating third-party SAML solution providers with AWS \(p. 197\)](#) for links to the web documentation for many of the supported SAML providers.

You also configure the information that you want the IdP to pass as SAML attributes to AWS as part of the authentication response. Most of this information appears in AWS as condition context keys that you can evaluate in your policies. These condition keys ensure that only authorized users in the right contexts are granted permissions to access your AWS resources. You can specify time windows that restrict when the console may be used. You can also specify the maximum time (up to 12 hours) that users can access the console before having to refresh their credentials. For details, see [Configuring SAML assertions for the authentication response \(p. 199\)](#).

Enabling custom identity broker access to the AWS console

You can write and run code to create a URL that lets users who sign in to your organization's network securely access the AWS Management Console. The URL includes a sign-in token that you get from AWS and that authenticates the user to AWS.

Note

If your organization uses an identity provider (IdP) that is compatible with SAML, you can set up access to the console without writing code. This works with providers like Microsoft's Active Directory Federation Services or open-source Shibboleth. For details, see [Enabling SAML 2.0 federated users to access the AWS Management Console \(p. 203\)](#).

To enable your organization's users to access the AWS Management Console, you can create a custom *identity broker* that performs the following steps:

1. Verify that the user is authenticated by your local identity system.
2. Call the AWS Security Token Service (AWS STS) [AssumeRole](#) (recommended) or [GetFederationToken](#) API operations to obtain temporary security credentials for the user. To learn about the different methods that you can use to assume a role, see [Using IAM roles \(p. 246\)](#). To learn how to pass optional session tags when you obtain your security credentials, see [Passing session tags in AWS STS \(p. 293\)](#).
 - If you use one of the `AssumeRole*` API operations to get the temporary security credentials for a role, you can include the `DurationSeconds` parameter in your call. This parameter specifies the duration of your role session, from 900 seconds (15 minutes) up to the maximum session duration setting for the role. To learn how to view or change the maximum value for a role, see [View the maximum session duration setting for a role \(p. 248\)](#). Additionally, if you use the `AssumeRole*` API operations, you must call them as an IAM user with long-term credentials. Otherwise, the call to the federation endpoint in step 3 fails.
 - If you use the `GetFederationToken` API operation to get the credentials, you can include the `DurationSeconds` parameter in your call. This parameter specifies the duration of your role session. The value can range from 900 seconds (15 minutes) to 129,600 seconds (36 hours). You can make this API call only by using the long-term AWS security credentials of an IAM user. You can also make these calls using AWS account root user credentials, but we do not recommend it. If you make this call as the root user, the default session lasts for one hour. Or you can specify a session from 900 seconds (15 minutes) up to 3,600 seconds (one hour).
3. Call the AWS federation endpoint and supply the temporary security credentials to request a sign-in token.
4. Construct a URL for the console that includes the token:
 - If you use one of the `AssumeRole*` API operations in your URL, you can include the `SessionDuration` HTTP parameter. This parameter specifies the duration of the console session, from 900 seconds (15 minutes) to 43200 seconds (12 hours).
 - If you use the `GetFederationToken` API operation in your URL, you can include the `DurationSeconds` parameter. This parameter specifies the duration of the federated console session. The value can range from 900 seconds (15 minutes) to 129,600 seconds (36 hours).

Note

Do not use the `SessionDuration` HTTP parameter if you got the temporary credentials with `GetFederationToken`. Doing so will cause the operation to fail.

5. Give the URL to the user or invoke the URL on the user's behalf.

The URL that the federation endpoint provides is valid for 15 minutes after it is created. This differs from the duration (in seconds) of the temporary security credential session that is associated with the URL. Those credentials are valid for the duration you specified when you created them, starting from the time they were created.

Important

The URL grants access to your AWS resources through the AWS Management Console if you have enabled permissions in the associated temporary security credentials. For this reason, you should treat the URL as a secret. We recommend returning the URL through a secure redirect, for example, by using a 302 HTTP response status code over an SSL connection. For more information about the 302 HTTP response status code, go to [RFC 2616, section 10.3.3](#).

To view a sample application that shows you how you can implement a single sign-on solution, go to [AWS Management Console federation proxy sample use case](#) in the *AWS Sample Code & Libraries*.

To complete these tasks, you can use the [HTTPS Query API for AWS Identity and Access Management \(IAM\)](#) and the [AWS Security Token Service \(AWS STS\)](#). Or, you can use programming languages, such as Java, Ruby, or C#, along with the appropriate [AWS SDK](#). Each of these methods is described in the following sections.

Topics

- [Example code using IAM query API operations \(p. 207\)](#)
- [Example code using Python \(p. 209\)](#)
- [Example code using Java \(p. 210\)](#)
- [Example showing how to construct the URL \(Ruby\) \(p. 212\)](#)

Example code using IAM query API operations

You can construct a URL that gives your federated users direct access to the AWS Management Console. This task uses the IAM and AWS STS HTTPS Query API. For more information about making query requests, see [Making Query Requests](#).

Note

The following procedure contains examples of text strings. To enhance readability, line breaks have been added to some of the longer examples. When you create these strings for your own use, you should omit any line breaks.

To give a federated user access to your resources from the AWS Management Console

1. Authenticate the user in your identity and authorization system.
2. Obtain temporary security credentials for the user. The temporary credentials consist of an access key ID, a secret access key, and a security token. For more information about creating temporary credentials, see [Temporary security credentials in IAM \(p. 301\)](#).

To get temporary credentials, you call either the AWS STS `AssumeRole` API (recommended) or the `GetFederationToken` API. For more information about the differences between these API operations, see [Understanding the API Options for Securely Delegating Access to Your AWS Account](#) in the AWS Security Blog.

Important

When you use the `GetFederationToken` API to create temporary security credentials, you must specify the permissions that the credentials grant to the user who assumes the

role. For any of the API operations that begin with `AssumeRole*`, you use an IAM role to assign permissions. For the other API operations, the mechanism varies with the API. For more details, see [Controlling permissions for temporary security credentials \(p. 316\)](#).

Additionally, if you use the `AssumeRole*` API operations, you must call them as an IAM user with long-term credentials. Otherwise, the call to the federation endpoint in step 3 fails.

3. After you obtain the temporary security credentials, build them into a JSON session string to exchange them for a sign-in token. The following example shows how to encode the credentials. You replace the placeholder text with the appropriate values from the credentials that you receive in the previous step.

```
{"sessionId": "*** temporary access key ID ***",
"sessionKey": "*** temporary secret access key ***",
"sessionToken": "*** security token ***"}
```

4. [URL encode](#) the session string from the previous step. Because the information that you are encoding is sensitive, we recommend that you avoid using a web service for this encoding. Instead, use a locally installed function or feature in your development toolkit to securely encode this information. You can use the `urllib.quote_plus` function in Python, the `URLEncoder.encode` function in Java, or the `CGI.escape` function in Ruby. See the examples later in this topic.
5. Send your request to the AWS federation endpoint at the following address:

<https://signin.aws.amazon.com/federation>

The request must include the `Action` and `Session` parameters, and (optionally) if you used an `AssumeRole*` API operation, a `SessionDuration` HTTP parameter as shown in the following example.

```
Action = getSigninToken
SessionDuration = time in seconds
Session = *** the URL encoded JSON string created in steps 3 & 4 ***
```

The `SessionDuration` HTTP parameter specifies the duration of the console session. This is separate from the duration of the temporary credentials that you specify using the `DurationSeconds` parameter. You can specify a `SessionDuration` maximum value of 43,200 (12 hours). If the `SessionDuration` parameter is missing, then the session defaults to the duration of the credentials that you retrieved from AWS STS in step 2 (which defaults to one hour). See the [documentation for the AssumeRole API](#) for details about how to specify a duration using the `DurationSeconds` parameter. The ability to create a console session that is longer than one hour is intrinsic to the `getSigninToken` operation of the federation endpoint.

Note

Do not use the `SessionDuration` HTTP parameter if you got the temporary credentials with `GetFederationToken`. Doing so will cause the operation to fail.

When you enable console sessions with an extended duration, you increase the risk of credential exposure. To help you mitigate this risk, you can immediately disable the active console sessions for any role by choosing **Revoke Sessions** on the **Role Summary** IAM console page. For more information, see [Revoking IAM role temporary security credentials \(p. 271\)](#).

The following is an example of what your request might look like. The lines are wrapped here for readability, but you should submit it as a one-line string.

```
https://signin.aws.amazon.com/federation
?Action=getSigninToken
&SessionDuration=1800
&Session=%7B%22sessionId%22%3A+%22ASIAJUMHIZPTOKTBMK5A%22%2C+%22sessionKey%22
%3A+%22LSD7LWI%2FI%2FN%2BgYpan5Qfz0XUpc8s7HYjRsgcsrsm%22%2C+%22sessionToken%22
```

```
2%3A+%22FQoDYXdzEBQaDLbj3VWv2u50NN%2F3yyLSASwYtWhPnGPMNmzZFFzSL0Qd3vtYHw5A5dW
AjOsrkdpkghomIe3mJip5%2FdjDBbo7SmO%2FENDEiCcdpsQKodTpleKA8xQq0CwfG6a69xdEBQT8
FipATnLbKoyS4b%2FehbnstUjZZQWp0wXXqFF7gSm%2FMe2tXe0jzsdp0012obeZ9lijPsdf1k2b5
PfGhiuyAR9aD5%2BubM0pY86fKex1qsytjvyTbZ9nXe6DvxVDcnCohOGETJ7XFkSFdH0v%2FYR25C
UAhJ3nXIkIbG7Ucv9cOEpcf%2Fg23ijRgILIBQ%3D%3D%22%7D
```

The response from the federation endpoint is a JSON document with a `SigninToken` value. It will look similar to the following example.

```
{"SigninToken": "*** the SigninToken string ***"}
```

- Finally, create the URL that your federated users can use to access the AWS Management Console. The URL is the same federation URL endpoint that you used in [Step 5 \(p. 208\)](#), plus the following parameters:

```
?Action = login
&Issuer = *** the form-urlencoded URL for your internal sign-in page ***
&Destination = *** the form-urlencoded URL to the desired AWS console page ***
&SigninToken = *** the value of SigninToken received in the previous step ***
```

The following example shows what the final URL might look like. The URL is valid for 15 minutes from the time it is created. The temporary security credentials and console session embedded within the URL are valid for the duration you specify in the `SessionDuration` HTTP parameter when you initially request them.

```
https://signin.aws.amazon.com/federation
?Action=login
&Issuer=https%3A%2F%2Fexample.com
&Destination=https%3A%2F%2Fconsole.aws.amazon.com%2F
&SigninToken=VCQgs5qZzt3Q6fn8Tr5EXAMPLELnwB7JjUc-SHwnUUWabcRdnWsi4DBn-dvC
CZ85wrD0nmlUcZEXAMPLE-vXYH4Q__mleuF_W2BE5HYexbe9y4Of-kje53SsjNNecATfjIzpW1
WibbnH6YcRiB0ffZBGExbEXAMPLE5aiKX4THWjQKC6gg6alHu6JFrnOJoK3dtP6I9a6hi6yPgm
iOkPZMmNGmhsvVxetKzr8mx3pxhHbMEXAMPLETv1pij0rok3IyCR2YVcIjqwfWv32HU2Xl471u
3fu6uOfUComeKiqTGx974xzJ0Zbdm_t_llrhEXAMPLEDDIissnyHGw2xaZzqudm4mo2uTDk9Pv
915K0ZCqIgEXAMPLEca6tgLPykEWGUyH6BdSC6166n4M4JkXIQgac7_7821YqixsNxZ6rsrpzwf
nQoS1407R0eJCCJ684EXAMPLERdBNnuLbUYpz2Iw3vIN0tQgOujwnwydPscm9F7foaEK3jwMkg
Apeb1-6L_OB12Mzhfxx55555EXAMPLEhyETEd4ZulKpdXHkg1619ZkI1Hz2Uy1RUTUhUxNtSQ
nWc5xkbBoEcXqposIEk7yhje9Vzh61AEXAMPLElbWeouACEMG6-Vd3dAgFYd6i5FYoyFrZLWvm
0LSG7RyYKeYN5VIZuk3YWQpyjPORiT5KUrsUi-NEXAMPLExMOMdoODEBgKQsk-iu2ozh6r8bxwC
RNhuJg
```

Example code using Python

The following example shows how to use Python to programmatically construct a URL that gives federated users direct access to the AWS Management Console. The example uses the [AWS SDK for Python \(Boto3\)](#).

The code uses the `AssumeRole` API to obtain temporary security credentials.

```
import urllib, json, sys
import requests # 'pip install requests'
import boto3 # AWS SDK for Python (Boto3) 'pip install boto3'

# Step 1: Authenticate user in your own identity system.

# Step 2: Using the access keys for an IAM user in your AWS account,
# call "AssumeRole" to get temporary access keys for the federated user

# Note: Calls to AWS STS AssumeRole must be signed using the access key ID
```

```
# and secret access key of an IAM user or using existing temporary credentials.
# The credentials can be in EC2 instance metadata, in environment variables,
# or in a configuration file, and will be discovered automatically by the
# client('sts') function. For more information, see the Python SDK docs:
# http://boto3.readthedocs.io/en/latest/reference/services/sts.html
# http://boto3.readthedocs.io/en/latest/reference/services/sts.html#STS.Client.assume_role
sts_connection = boto3.client('sts')

assumed_role_object = sts_connection.assume_role(
    RoleArn="arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:role/ROLE-NAME",
    RoleSessionName="AssumeRoleSession",
)

# Step 3: Format resulting temporary credentials into JSON
url_credentials = {}
url_credentials['sessionId'] = assumed_role_object.get('Credentials').get('AccessKeyId')
url_credentials['sessionKey'] =
    assumed_role_object.get('Credentials').get('SecretAccessKey')
url_credentials['sessionToken'] =
    assumed_role_object.get('Credentials').get('SessionToken')
json_string_with_temp_credentials = json.dumps(url_credentials)

# Step 4. Make request to AWS federation endpoint to get sign-in token. Construct the
# parameter string with
# the sign-in action request, a 12-hour session duration, and the JSON document with
# temporary credentials
# as parameters.
request_parameters = "?Action=getSigninToken"
request_parameters += "&SessionDuration=43200"
if sys.version_info[0] < 3:
    def quote_plus_function(s):
        return urllib.quote_plus(s)
else:
    def quote_plus_function(s):
        return urllib.parse.quote_plus(s)
request_parameters += "&Session=" + quote_plus_function(json_string_with_temp_credentials)
request_url = "https://signin.aws.amazon.com/federation" + request_parameters
r = requests.get(request_url)
# Returns a JSON document with a single element named SigninToken.
signin_token = json.loads(r.text)

# Step 5: Create URL where users can use the sign-in token to sign in to
# the console. This URL must be used within 15 minutes after the
# sign-in token was issued.
request_parameters = "?Action=login"
request_parameters += "&Issuer=Example.org"
request_parameters += "&Destination=" + quote_plus_function("https://
console.aws.amazon.com/")
request_parameters += "&SigninToken=" + signin_token["SigninToken"]
request_url = "https://signin.aws.amazon.com/federation" + request_parameters

# Send final URL to stdout
print (request_url)
```

Example code using Java

The following example shows how to use Java to programmatically construct a URL that gives federated users direct access to the AWS Management Console. The following code snippet uses the [AWS SDK for Java](#).

```
import java.net.URLEncoder;
import java.net.URL;
import java.netURLConnection;
import java.io.BufferedReader;
```

```

import java.io.InputStreamReader;
// Available at http://www.json.org/java/index.html
import org.json.JSONObject;
import com.amazonaws.auth.AWS Credentials;
import com.amazonaws.auth.BasicAWS Credentials;
import com.amazonaws.services.securitytoken.AWSSecurityTokenServiceClient;
import com.amazonaws.services.securitytoken.model.Credentials;
import com.amazonaws.services.securitytoken.model.GetFederationTokenRequest;
import com.amazonaws.services.securitytoken.model.GetFederationTokenResult;

/* Calls to AWS STS API operations must be signed using the access key ID
   and secret access key of an IAM user or using existing temporary
   credentials. The credentials should not be embedded in code. For
   this example, the code looks for the credentials in a
   standard configuration file.
*/
AWS Credentials credentials =
    new PropertiesCredentials(
        AwsConsoleApp.class.getResourceAsStream("AwsCredentials.properties"));

AWSSecurityTokenServiceClient stsClient =
    new AWSSecurityTokenServiceClient(credentials);

GetFederationTokenRequest getFederationTokenRequest =
    new GetFederationTokenRequest();
getFederationTokenRequest.setDurationSeconds(1800);
getFederationTokenRequest.setName("UserName");

// A sample policy for accessing Amazon Simple Notification Service (Amazon SNS) in the
// console.

String policy = "{\"Version\":\"2012-10-17\", \"Statement\":[{\"Action\":\"sns:*\", \" +
    "\"Effect\":\"Allow\", \"Resource\":\"*\"]}]";

getFederationTokenRequest.setPolicy(policy);

GetFederationTokenResult federationTokenResult =
    stsClient.getFederationToken(getFederationTokenRequest);

Credentials federatedCredentials = federationTokenResult.getCredentials();

// The issuer parameter specifies your internal sign-in
// page, for example https://mysignin.internal.mycompany.com/.
// The console parameter specifies the URL to the destination console of the
// AWS Management Console. This example goes to Amazon SNS.
// The signin parameter is the URL to send the request to.

String issuerURL = "https://mysignin.internal.mycompany.com/";
String consoleURL = "https://console.aws.amazon.com/sns";
String signInURL = "https://signin.aws.amazon.com/federation";

// Create the sign-in token using temporary credentials,
// including the access key ID, secret access key, and security token.
String sessionJson = String.format(
    "{\"%1$s\":\"%2$s\", \"%3$s\": \"%4$s\", \"%5$s\":\"%6$s\"}",
    "sessionId", federatedCredentials.getAccessKeyId(),
    "sessionKey", federatedCredentials.getSecretAccessKey(),
    "sessionToken", federatedCredentials.getSessionToken());

// Construct the sign-in request with the request sign-in token action, a
// 12-hour console session duration, and the JSON document with temporary
// credentials as parameters.

String getSignInTokenURL = signInURL +
    "?Action=getSignInToken" +

```

```

        "&DurationSeconds=43200" +
        "&SessionType=json&Session=" +
        URLEncoder.encode(sessionJson, "UTF-8");

URL url = new URL(getSigninTokenURL);

// Send the request to the AWS federation endpoint to get the sign-in token
URLConnection conn = url.openConnection();

BufferedReader bufferReader = new BufferedReader(new
    InputStreamReader(conn.getInputStream()));
String returnContent = bufferReader.readLine();

String signinToken = new JSONObject(returnContent).getString("SigninToken");

String signinTokenParameter = "&SigninToken=" + URLEncoder.encode(signinToken, "UTF-8");

// The issuer parameter is optional, but recommended. Use it to direct users
// to your sign-in page when their session expires.

String issuerParameter = "&Issuer=" + URLEncoder.encode(issuerURL, "UTF-8");

// Finally, present the completed URL for the AWS console session to the user

String destinationParameter = "&Destination=" + URLEncoder.encode(consoleURL, "UTF-8");
String loginURL = signInURL + "?Action=login" +
    signinTokenParameter + issuerParameter + destinationParameter;

```

Example showing how to construct the URL (Ruby)

The following example shows how to use Ruby to programmatically construct a URL that gives federated users direct access to the AWS Management Console. This code snippet uses the [AWS SDK for Ruby](#).

```

require 'rubygems'
require 'json'
require 'open-uri'
require 'cgi'
require 'aws-sdk'

# Create a new STS instance
#
# Note: Calls to AWS STS API operations must be signed using an access key ID
# and secret access key. The credentials can be in EC2 instance metadata
# or in environment variables and will be automatically discovered by
# the default credentials provider in the AWS Ruby SDK.
sts = Aws::STS::Client.new()

# The following call creates a temporary session that returns
# temporary security credentials and a session token.
# The policy grants permissions to work
# in the AWS SNS console.

session = sts.get_federation_token({
    duration_seconds: 1800,
    name: "UserName",
    policy: "{\"Version\":\"2012-10-17\", \"Statement\":{\"Effect\":\"Allow\", \"Action\":
    \"sns:*\", \"Resource\":\"*\"}}",
})

# The issuer value is the URL where users are directed (such as
# to your internal sign-in page) when their session expires.
#
# The console value specifies the URL to the destination console.
# This example goes to the Amazon SNS console.

```

```

#
# The sign-in value is the URL of the AWS STS federation endpoint.
issuer_url = "https://mysignin.internal.mycompany.com/"
console_url = "https://console.aws.amazon.com/sns"
signin_url = "https://signin.aws.amazon.com/federation"

# Create a block of JSON that contains the temporary credentials
# (including the access key ID, secret access key, and session token).
session_json = {
    :sessionId => session.credentials[:access_key_id],
    :sessionKey => session.credentials[:secret_access_key],
    :sessionToken => session.credentials[:session_token]
}.to_json

# Call the federation endpoint, passing the parameters
# created earlier and the session information as a JSON block.
# The request returns a sign-in token that's valid for 15 minutes.
# Signing in to the console with the token creates a session
# that is valid for 12 hours.
get_signin_token_url = signin_url +
    "?Action=getSigninToken" +
    "&SessionType=json&Session=" +
    CGI.escape(session_json)

returned_content = URI.parse(get_signin_token_url).read

# Extract the sign-in token from the information returned
# by the federation endpoint.
signin_token = JSON.parse(returned_content)['SigninToken']
signin_token_param = "&SigninToken=" + CGI.escape(signin_token)

# Create the URL to give to the user, which includes the
# sign-in token and the URL of the console to open.
# The "Issuer" parameter is optional but recommended.
issuer_param = "&Issuer=" + CGI.escape(issuer_url)
destination_param = "&Destination=" + CGI.escape(console_url)
login_url = signin_url + "?Action=login" + signin_token_param +
    issuer_param + destination_param
  
```

Using service-linked roles

A service-linked role is a unique type of IAM role that is linked directly to an AWS service. Service-linked roles are predefined by the service and include all the permissions that the service requires to call other AWS services on your behalf. The linked service also defines how you create, modify, and delete a service-linked role. A service might automatically create or delete the role. It might allow you to create, modify, or delete the role as part of a wizard or process in the service. Or it might require that you use IAM to create or delete the role. Regardless of the method, service-linked roles make setting up a service easier because you don't have to manually add the necessary permissions for the service to complete actions on your behalf.

The linked service defines the permissions of its service-linked roles, and unless defined otherwise, only that service can assume the roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete the roles only after first deleting their related resources. This protects your resources because you can't inadvertently remove permission to access the resources.

Tip

For information about which services support using service-linked roles, see [AWS services that work with IAM \(p. 611\)](#) and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions

You must configure permissions for an IAM entity (user or role) to allow the user or role to create or edit the service-linked role.

Note

The ARN for a service-linked role includes a service principal, which is indicated in the policies below as `SERVICE-NAME.amazonaws.com`. Do not try to guess the service principal, because it is case sensitive and the format can vary across AWS services. To view the service principal for a service, see its service-linked role documentation.

To allow an IAM entity to create a specific service-linked role

Add the following policy to the IAM entity that needs to create the service-linked role.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/SERVICE-
NAME.amazonaws.com/SERVICE-LINKED-ROLE-NAME-PREFIX*",
            "Condition": {"StringLike": {"iam:AWSServiceName": "SERVICE-
NAME.amazonaws.com"}}
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:AttachRolePolicy",
                "iam:PutRolePolicy"
            ],
            "Resource": "arn:aws:iam::*:role/aws-service-role/SERVICE-
NAME.amazonaws.com/SERVICE-LINKED-ROLE-NAME-PREFIX*"
        }
    ]
}
```

To allow an IAM entity to create any service-linked role

Add the following statement to the permissions policy for the IAM entity that needs to create a service-linked role, or any service role that includes the needed policies. This policy statement does not allow the IAM entity to attach a policy to the role.

```
{
    "Effect": "Allow",
    "Action": "iam:CreateServiceLinkedRole",
    "Resource": "arn:aws:iam::*:role/aws-service-role/*"
}
```

To allow an IAM entity to edit the description of any service roles

Add the following statement to the permissions policy for the IAM entity that needs to edit the description of a service-linked role, or any service role.

```
{
    "Effect": "Allow",
    "Action": "iam:UpdateRoleDescription",
    "Resource": "arn:aws:iam::*:role/aws-service-role/*"
}
```

To allow an IAM entity to delete a specific service-linked role

Add the following statement to the permissions policy for the IAM entity that needs to delete the service-linked role.

```
{  
    "Effect": "Allow",  
    "Action": [  
        "iam:DeleteServiceLinkedRole",  
        "iam:GetServiceLinkedRoleDeletionStatus"  
    ],  
    "Resource": "arn:aws:iam::*:role/aws-service-role/SERVICE-NAME.amazonaws.com/SERVICE-LINKED-ROLE-NAME-PREFIX*"  
}
```

To allow an IAM entity to delete any service-linked role

Add the following statement to the permissions policy for the IAM entity that needs to delete a service-linked role, but not service role.

```
{  
    "Effect": "Allow",  
    "Action": [  
        "iam:DeleteServiceLinkedRole",  
        "iam:GetServiceLinkedRoleDeletionStatus"  
    ],  
    "Resource": "arn:aws:iam::*:role/aws-service-role/*"  
}
```

To allow an IAM entity to pass an existing role to the service

Some AWS services allow you to pass an existing role to the service, instead of creating a new service-linked role. To do this, a user must have permissions to *pass the role* to the service. Add the following statement to the permissions policy for the IAM entity that needs to pass a role. This policy statement also allows the entity to view a list of roles from which they can choose the role to pass. For more information, see [Granting a user permissions to pass a role to an AWS service \(p. 251\)](#).

```
{  
    "Effect": "Allow",  
    "Action": [  
        "iam>ListRoles",  
        "iam:PassRole"  
    ],  
    "Resource": "arn:aws:iam::123456789012:role/my-role-for-XYZ"  
}
```

Transferring service-linked role permissions

The permissions granted by a service-linked role are indirectly transferable to other users and roles. When you allow a service to perform operations in other services, the service can use those permissions in the future. If another user or role has permission to perform actions in the service, the service can then assume the role and access resources in other services. This means that the other user or role can indirectly access the other services.

For example, when you create an Amazon RDS DB instance, [RDS creates the service-linked role](#) for you. This role allows RDS to call Amazon EC2, Amazon SNS, Amazon CloudWatch Logs, and Amazon Kinesis on your behalf whenever you edit the DB instance. If you create a policy to allow users and roles in your account or another account to access that Amazon RDS instance, then RDS can still use that role make

changes to EC2, SNS, CloudWatch Logs, and Kinesis on their behalf. The new user or role can indirectly edit resources in those other services.

Creating a service-linked role

The method that you use to create a service-linked role depends on the service. In some cases, you don't need to manually create a service-linked role. For example, when you complete a specific action (such as creating a resource) in the service, the service might create the service-linked role for you. Or if you were using a service before it began supporting service-linked roles, then the service might have automatically created the role in your account. To learn more, see [A new role appeared in my AWS account \(p. 587\)](#).

In other cases, the service might support creating a service-linked role manually using the service console, API, or CLI. For information about which services support using service-linked roles, see [AWS services that work with IAM \(p. 611\)](#) and look for the services that have Yes in the **Service-Linked Role** column. To learn whether the service supports creating the service-linked role, choose the Yes link to view the service-linked role documentation for that service.

If the service does not support creating the role, then you can use IAM to create the service-linked role.

Important

Service-linked roles count toward your [IAM roles in an AWS account](#) limit, but if you have reached your limit, you can still create service-linked roles in your account. Only service-linked roles can exceed the limit.

Creating a service-linked role (console)

Before you create a service-linked role in IAM, find out whether the linked service automatically creates service-linked roles. In addition, learn whether you can create the role from the service's console, API, or CLI.

To create a service-linked role (console)

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 IAM console, choose **Roles**. Then choose **Create role**.
3. Choose the **AWS Service** role type, and then choose the service that you want to allow to assume this role.
4. Choose the use case for your service. If the specified service has only one use case, it is selected for you. Use cases are defined by the service to include the trust policy required by the service. Then choose **Next: Permissions**.
5. Choose one or more permissions policies to attach to the role. Depending on the use case that you selected, the service might do any of the following:
 - Define the permissions used by the role
 - Allow you to choose from a limited set of permissions
 - Allow you to choose from any permissions
 - Allow you to select no policies at this time, create the policies later, and then attach them to the role.

Select the box next to the policy that assigns the permissions that you want the role to have, and then choose **Next: Tags**.

Note

The permissions that you specify are available to any entity that uses the role. By default, a role has no permissions.

6. Choose **Next: Review**. You cannot attach tags to service-linked roles during creation. For more information about using tags in IAM, see [Tagging IAM users and roles \(p. 289\)](#).
7. For **Role name**, the degree of role name customization is defined by the service. If the service defines the role's name, then this option is not editable. In other cases, the service might define a prefix for the role and allow you to type an optional suffix.

If possible, type a role name suffix to add to the default name. This suffix helps you identify the purpose of this role. Role names must be unique within your AWS account. They are not distinguished by case. For example, you cannot create roles named both `<service-linked-role-name>_SAMPLE` and `<service-linked-role-name>_sample`. Because various entities might reference the role, you cannot edit the name of the role after it has been created.

8. (Optional) For **Role description**, edit the description for the new service-linked role.
9. Review the role and then choose **Create role**.

Creating a service-linked role (AWS CLI)

Before creating a service-linked role in IAM, find out whether the linked service automatically creates service-linked roles and whether you can create the role from the service's CLI. If the service CLI is not supported, you can use IAM commands to create a service-linked role with the trust policy and inline policies that the service needs to assume the role.

To create a service-linked role (AWS CLI)

Run the following command:

```
aws iam create-service-linked-role --aws-service-name SERVICE-NAME.amazonaws.com
```

Creating a service-linked role (AWS API)

Before creating a service-linked role in IAM, find out whether the linked service automatically creates service-linked roles and whether you can create the role from the service's API. If the service API is not supported, you can use the AWS API to create a service-linked role with the trust policy and inline policies that the service needs to assume the role.

To create a service-linked role (AWS API)

Use the [CreateServiceLinkedRole](#) API call. In the request, specify a service name of `SERVICE_NAME_URL`.amazonaws.com.

For example, to create the **Lex Bots** service-linked role, use `lex.amazonaws.com`.

Editing a service-linked role

The method that you use to edit a service-linked role depends on the service. Some services might allow you to edit the permissions for a service-linked role from the service console, API, or CLI. However, after you create a service-linked role, you cannot change the name of the role because various entities might reference the role. You can edit the description of any role from the IAM console, API, or CLI.

For information about which services support using service-linked roles, see [AWS services that work with IAM \(p. 611\)](#) and look for the services that have **Yes** in the **Service-Linked Role** column. To learn whether the service supports editing the service-linked role, choose the **Yes** link to view the service-linked role documentation for that service.

Editing a service-linked role description (console)

You can use the IAM console to edit the description of a service-linked role.

To edit the description of a service-linked role (console)

1. In the navigation pane of the IAM console, choose **Roles**.
2. Choose the name of the role to modify.
3. To the far right of **Role description**, choose **Edit**.
4. Type a new description in the box and choose **Save**.

Editing a service-linked role description (AWS CLI)

You can use IAM commands from the AWS CLI to edit the description of a service-linked role.

To change the description of a service-linked role (AWS CLI)

1. (Optional) To view the current description for a role, run the following command:

```
aws iam get-role --role-name ROLE-NAME
```

Use the role name, not the ARN, to refer to roles with the CLI commands. For example, if a role has the following ARN: `arn:aws:iam::123456789012:role/myrole`, you refer to the role as **myrole**.

2. To update a service-linked role's description, run the following command:

```
aws iam update-role --role-name ROLE-NAME --description OPTIONAL-DESCRIPTION
```

Editing a service-linked role description (AWS API)

You can use the AWS API to edit the description of a service-linked role.

To change the description of a service-linked role (AWS API)

1. (Optional) To view the current description for a role, call the following operation, and specify the name of the role:

AWS API: [GetRole](#)

2. To update a role's description, call the following operation, and specify the name (and optional description) of the role:

AWS API: [UpdateRole](#)

Deleting a service-linked role

The method that you use to create a service-linked role depends on the service. In some cases, you don't need to manually delete a service-linked role. For example, when you complete a specific action (such as removing a resource) in the service, the service might delete the service-linked role for you.

In other cases, the service might support deleting a service-linked role manually from the service console, API, or CLI.

For information about which services support using service-linked roles, see [AWS services that work with IAM \(p. 611\)](#) and look for the services that have **Yes** in the **Service-Linked Role** column. To learn whether the service supports deleting the service-linked role, choose the **Yes** link to view the service-linked role documentation for that service.

If the service does not support deleting the role, then you can delete the service-linked role from the IAM console, API, or CLI. If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that is not actively monitored or maintained. However, you must clean up your service-linked role before you can delete it.

Cleaning up a service-linked role

Before you can use IAM to delete a service-linked role, you must first confirm that the role has no active sessions and remove any resources used by the role.

To check whether the service-linked role has an active session in the IAM console

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 IAM console, choose **Roles**. Then choose the name (not the check box) of the service-linked role.
3. On the **Summary** page for the selected role, choose the **Access Advisor** tab.
4. On the **Access Advisor** tab, review recent activity for the service-linked role.

Note

If you are unsure whether the service is using the service-linked role, you can try to delete the role. If the service is using the role, then the deletion fails and you can view the regions where the role is being used. If the role is being used, then you must wait for the session to end before you can delete the role. You cannot revoke the session for a service-linked role.

To remove resources used by a service-linked role

For information about which services support using service-linked roles, see [AWS services that work with IAM \(p. 611\)](#) and look for the services that have **Yes** in the **Service-Linked Role** column. To learn whether the service supports deleting the service-linked role, choose the **Yes** link to view the service-linked role documentation for that service. See the documentation for that service to learn how to remove resources used by your service-linked role.

Deleting a service-linked role (console)

You can use the IAM console to delete a service-linked role.

To delete a service-linked role (console)

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 IAM console, choose **Roles**. Then select the check box next to the role name that you want to delete, not the name or row itself.
3. For **Role actions** at the top of the page, choose **Delete role**.
4. In the confirmation dialog box, review the last accessed information, which shows when each of the selected roles last accessed an AWS service. This helps you to confirm whether the role is currently active. If you want to proceed, choose **Yes, Delete** to submit the service-linked role for deletion.
5. Watch the IAM console notifications to monitor the progress of the service-linked role deletion. Because the IAM service-linked role deletion is asynchronous, after you submit the role for deletion, the deletion task can succeed or fail.
 - If the task succeeds, then the role is removed from the list and a notification of success appears at the top of the page.
 - If the task fails, you can choose **View details** or **View Resources** from the notifications to learn why the deletion failed. If the deletion fails because the role is using the service's resources, then

the notification includes a list of resources, if the service returns that information. You can then [clean up the resources \(p. 219\)](#) and submit the deletion again.

Note

You might have to repeat this process several times, depending on the information that the service returns. For example, your service-linked role might use six resources and your service might return information about five of them. If you clean up the five resources and submit the role for deletion again, the deletion fails and the service reports the one remaining resource. A service might return all of the resources, a few of them, or it might not report any resources.

- If the task fails and the notification does not include a list of resources, then the service might not return that information. To learn how to clean up the resources for that service, see [AWS services that work with IAM \(p. 611\)](#). Find your service in the table, and choose the **Yes** link to view the service-linked role documentation for that service.

Deleting a service-linked role (AWS CLI)

You can use IAM commands from the AWS CLI to delete a service-linked role.

To delete a service-linked role (AWS CLI)

1. If you don't know the name of the service-linked role that you want to delete, type the following command to list the roles and their Amazon Resource Names (ARNs) in your account:

```
aws iam get-role --role-name role-name
```

Use the role name, not the ARN, to refer to roles with the CLI commands. For example, if a role has the following ARN: `arn:aws:iam::123456789012:role/myrole`, you refer to the role as **myrole**.

2. Because a service-linked role cannot be deleted if it is being used or has associated resources, you must submit a deletion request. That request can be denied if these conditions are not met. You must capture the `deletion-task-id` from the response to check the status of the deletion task. Type the following command to submit a service-linked role deletion request:

```
aws iam delete-service-linked-role --role-name role-name
```

3. Type the following command to check the status of the deletion task:

```
aws iam get-service-linked-role-deletion-status --deletion-task-id deletion-task-id
```

The status of the deletion task can be NOT_STARTED, IN_PROGRESS, SUCCEEDED, or FAILED. If the deletion fails, the call returns the reason that it failed so that you can troubleshoot. If the deletion fails because the role is using the service's resources, then the notification includes a list of resources, if the service returns that information. You can then [clean up the resources \(p. 219\)](#) and submit the deletion again.

Note

You might have to repeat this process several times, depending on the information that the service returns. For example, your service-linked role might use six resources and your service might return information about five of them. If you clean up the five resources and submit the role for deletion again, the deletion fails and the service reports the one remaining resource. A service might return all of the resources, a few of them, or it might not report any resources. To learn how to clean up the resources for a service that does not report any resources, see [AWS services that work with IAM \(p. 611\)](#). Find your service in the table, and choose the **Yes** link to view the service-linked role documentation for that service.

Deleting a service-linked role (AWS API)

You can use the AWS API to delete a service-linked role.

To delete a service-linked role (AWS API)

1. To submit a deletion request for a service-linked role, call [DeleteServiceLinkedRole](#). In the request, specify a role name.

Because a service-linked role cannot be deleted if it is being used or has associated resources, you must submit a deletion request. That request can be denied if these conditions are not met. You must capture the `DeletionTaskId` from the response to check the status of the deletion task.

2. To check the status of the deletion, call [GetServiceLinkedRoleDeletionStatus](#). In the request, specify the `DeletionTaskId`.

The status of the deletion task can be `NOT_STARTED`, `IN_PROGRESS`, `SUCCEEDED`, or `FAILED`. If the deletion fails, the call returns the reason that it failed so that you can troubleshoot. If the deletion fails because the role is using the service's resources, then the notification includes a list of resources, if the service returns that information. You can then [clean up the resources \(p. 219\)](#) and submit the deletion again.

Note

You might have to repeat this process several times, depending on the information that the service returns. For example, your service-linked role might use six resources and your service might return information about five of them. If you clean up the five resources and submit the role for deletion again, the deletion fails and the service reports the one remaining resource. A service might return all of the resources, a few of them, or it might not report any resources. To learn how to clean up the resources for a service that does not report any resources, see [AWS services that work with IAM \(p. 611\)](#). Find your service in the table, and choose the **Yes** link to view the service-linked role documentation for that service.

Creating IAM roles

To create a role, you can use the AWS Management Console, the AWS CLI, the Tools for Windows PowerShell, or the IAM API.

If you use the AWS Management Console, a wizard guides you through the steps for creating a role. The wizard has slightly different steps depending on whether you're creating a role for an AWS service, for an AWS account, or for a federated user.

Topics

- [Creating a role to delegate permissions to an IAM user \(p. 221\)](#)
- [Creating a role to delegate permissions to an AWS service \(p. 229\)](#)
- [Creating a role for a third-party Identity Provider \(federation\) \(p. 234\)](#)
- [Examples of policies for delegating access \(p. 243\)](#)

Creating a role to delegate permissions to an IAM user

You can use IAM roles to delegate access to your AWS resources. With IAM roles, you can establish trust relationships between your *trusting* account and other AWS *trusted* accounts. The trusting account owns the resource to be accessed and the trusted account contains the users who need access to the resource. However, it is possible for another account to own a resource in your account. For example, the trusting account might allow the trusted account to create new resources, such as creating new objects in an

Amazon S3 bucket. In that case, the account that creates the resource owns the resource and controls who can access that resource.

After you create the trust relationship, an IAM user or an application from the trusted account can use the AWS Security Token Service (AWS STS) [AssumeRole](#) API operation. This operation provides temporary security credentials that enable access to AWS resources in your account.

The accounts can both be controlled by you, or the account with the users can be controlled by a third party. If the other account with the users is an AWS account that you do not control, then you can use the `externalId` attribute. The external ID can be any word or number that is agreed upon between you and the administrator of the third-party account. This option automatically adds a condition to the trust policy that allows the user to assume the role only if the request includes the correct `sts:ExternalID`. For more information, see [How to use an external ID when granting access to your AWS resources to a third party \(p. 225\)](#).

For information about how to use roles to delegate permissions, see [Roles terms and concepts \(p. 168\)](#). For information about using a service role to allow services to access resources in your account, see [Creating a role to delegate permissions to an AWS service \(p. 229\)](#).

Creating an IAM role (console)

You can use the AWS Management Console to create a role that an IAM user can assume. For example, assume that your organization has multiple AWS accounts to isolate a development environment from a production environment. For a high-level description of the steps to set up and use a role that allows users in the development account to access resources in the production account, see [Example scenario using separate development and production accounts \(p. 172\)](#).

To create a role (console)

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, choose **Roles** and then choose **Create role**.
3. Choose the **Another AWS account** role type.
4. For **Account ID**, type the AWS account ID to which you want to grant access to your resources.

The administrator of the specified account can grant permission to assume this role to any IAM user in that account. To do this, the administrator attaches a policy to the user or a group that grants permission for the `sts:AssumeRole` action. That policy must specify the role's ARN as the Resource.

5. If you are granting permissions to users from an account that you do not control, and the users will assume this role programmatically, then select **Require external ID**. The external ID can be any word or number that is agreed upon between you and the administrator of the third-party account. This option automatically adds a condition to the trust policy that allows the user to assume the role only if the request includes the correct `sts:ExternalID`. For more information, see [How to use an external ID when granting access to your AWS resources to a third party \(p. 225\)](#).

Important

Choosing this option restricts access to the role only through the AWS CLI, Tools for Windows PowerShell, or the AWS API. This is because you cannot use the AWS console to switch to a role that has an `externalId` condition in its trust policy. However, you can create this kind of access programmatically by writing a script or an application using the relevant SDK. For more information and a sample script, see [How to Enable Cross-Account Access to the AWS Management Console](#) in the [AWS Security Blog](#).

6. If you want to restrict the role to users who sign in with multi-factor authentication (MFA), select **Require MFA**. This adds a condition to the role's trust policy that checks for an MFA sign-in. A user who wants to assume the role must sign in with a temporary one-time password from a configured

MFA device. Users without MFA authentication cannot assume the role. For more information about MFA, see [Using multi-factor authentication \(MFA\) in AWS \(p. 111\)](#)

7. Choose **Next: Permissions**.
8. IAM includes a list of the AWS managed and customer managed policies in your account. Select the policy to use for the permissions policy or choose **Create policy** to open a new browser tab and create a new policy from scratch. For more information, see step 4 in the procedure [Creating IAM policies \(console\) \(p. 439\)](#). After you create the policy, close that tab and return to your original tab. Select the check box next to the permissions policies that you want anyone who assumes the role to have. If you prefer, you can select no policies at this time, and then attach policies to the role later. By default, a role has no permissions.
9. (Optional) Set a [permissions boundary \(p. 365\)](#). This is an advanced feature.
Open the **Set permissions boundary** section and choose **Use a permissions boundary to control the maximum role permissions**. Select the policy to use for the permissions boundary.
10. Choose **Next: Tags**.
11. (Optional) Add metadata to the role by attaching tags as key–value pairs. For more information about using tags in IAM, see [Tagging IAM users and roles \(p. 289\)](#).
12. Choose **Next: Review**.
13. For **Role name**, type a name for your role. Role names must be unique within your AWS account. They are not distinguished by case. For example, you cannot create roles named both **PRODROLE** and **prodrole**. Because other AWS resources might reference the role, you cannot edit the name of the role after it has been created.
14. (Optional) For **Role description**, type a description for the new role.
15. Review the role and then choose **Create role**.

Important

Remember that this is only the first half of the configuration required. You must also give individual users in the trusted account permissions to switch to the role in the console, or assume the role programmatically. For more information about this step, see [Granting a user permissions to switch roles \(p. 248\)](#).

Creating an IAM role (AWS CLI)

Creating a role from the AWS CLI involves multiple steps. When you use the console to create a role, many of the steps are done for you, but with the AWS CLI you must explicitly perform each step yourself. You must create the role and then assign a permissions policy to the role. Optionally, you can also set the [permissions boundary \(p. 365\)](#) for your role.

To create a role for cross-account access (AWS CLI)

1. Create a role: [aws iam create-role](#)
2. Attach a managed permissions policy to the role: [aws iam attach-role-policy](#)
or
Create an inline permissions policy for the role: [aws iam put-role-policy](#)
3. (Optional) Add custom attributes to the role by attaching tags: [aws iam tag-role](#)
For more information, see [Managing tags on IAM entities \(AWS CLI or AWS API\) \(p. 292\)](#).
4. (Optional) Set the [permissions boundary \(p. 365\)](#) for the role: [aws iam put-role-permissions-boundary](#)

A permissions boundary controls the maximum permissions that a role can have. Permissions boundaries are an advanced AWS feature.

The following example shows the first two, and most common steps for creating a cross-account role in a simple environment. This example allows any user in the 123456789012 account to assume the role and view the example_bucket Amazon S3 bucket. This example also assumes that you are using a client computer running Windows, and have already configured your command line interface with your account credentials and Region. For more information, see [Configuring the AWS Command Line Interface](#).

In this example, include the following trust policy in the first command when you create the role. This trust policy allows users in the 123456789012 account to assume the role using the `AssumeRole` operation, but only if the user provides MFA authentication using the `SerialNumber` and `TokenCode` parameters. For more information about MFA, see [Using multi-factor authentication \(MFA\) in AWS \(p. 111\)](#).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": { "AWS": "arn:aws:iam::123456789012:root" },
      "Action": "sts:AssumeRole",
      "Condition": { "Bool": { "aws:MultiFactorAuthPresent": "true" } }
    }
  ]
}
```

Important

If your `Principal` element contains the ARN for a specific IAM role or user, then that ARN is transformed to a unique principal ID when the policy is saved. This helps mitigate the risk of someone escalating their permissions by removing and recreating the role or user. You don't normally see this ID in the console because there is also a reverse transformation back to the ARN when the trust policy is displayed. However, if you delete the role or user, then the principal ID appears in the console because AWS can no longer map it back to an ARN. Therefore, if you delete and recreate a user or role referenced in a trust policy's `Principal` element, you must edit the role to replace the ARN.

When you use the second command, you must attach an existing managed policy to the role. The following permissions policy allows anyone who assumes the role to perform only the `ListBucket` action on the example_bucket Amazon S3 bucket.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3>ListBucket",
      "Resource": "arn:aws:s3:::example_bucket"
    }
  ]
}
```

To create this `Test-UserAccess-Role` role, you must first save the previous trust policy with the name `trustpolicyforacct123456789012.json` to the `policies` folder in your local C: drive. Then save the previous permissions policy as a customer managed policy in your AWS account with the name `PolicyForRole`. You can then use the following commands to create the role and attach the managed policy.

```
# Create the role and attach the trust policy file that allows users in the specified
# account to assume the role.
$ aws iam create-role --role-name Test-UserAccess-Role --assume-role-policy-document
file://C:\policies\trustpolicyforacct123456789012.json
```

```
# Attach the permissions policy (in this example a managed policy) to the role to specify what it is allowed to do.  
$ aws iam attach-role-policy --role-name Test-UserAccess-Role --policy-arn arn:aws:iam::123456789012:role/PolicyForRole
```

Important

Remember that this is only the first half of the configuration required. You must also give individual users in the trusted account permissions to switch to the role. For more information about this step, see [Granting a user permissions to switch roles \(p. 248\)](#).

After you create the role and grant it permissions to perform AWS tasks or access AWS resources, any users in the 123456789012 account can assume the role. For more information, see [Switching to an IAM role \(AWS CLI\) \(p. 256\)](#).

Creating an IAM role (AWS API)

Creating a role from the AWS API involves multiple steps. When you use the console to create a role, many of the steps are done for you, but with the API you must explicitly perform each step yourself. You must create the role and then assign a permissions policy to the role. Optionally, you can also set the [permissions boundary \(p. 365\)](#) for your role.

To create a role in code (AWS API)

1. Create a role: [CreateRole](#)
For the role's trust policy, you can specify a file location.
2. Attach a managed permission policy to the role: [AttachRolePolicy](#)

or

Create an inline permission policy for the role: [PutRolePolicy](#)

Important

Remember that this is only the first half of the configuration required. You must also give individual users in the trusted account permissions to switch to the role. For more information about this step, see [Granting a user permissions to switch roles \(p. 248\)](#).

3. (Optional) Add custom attributes to the user by attaching tags: [TagRole](#)
For more information, see [Managing tags on IAM entities \(AWS CLI or AWS API\) \(p. 292\)](#).
4. (Optional) Set the [permissions boundary \(p. 365\)](#) for the role: [PutRolePermissionsBoundary](#)

A permissions boundary controls the maximum permissions that a role can have. Permissions boundaries are an advanced AWS feature.

After you create the role and grant it permissions to perform AWS tasks or access AWS resources, you must grant permissions to users in the account to allow them to assume the role. For more information about assuming a role, see [Switching to an IAM role \(AWS API\) \(p. 262\)](#).

How to use an external ID when granting access to your AWS resources to a third party

At times, you need to give a third party access to your AWS resources (delegate access). One important aspect of this scenario is the *External ID*, optional information that you can use in an IAM role trust policy to designate who can assume the role.

Important

AWS does not treat the external ID as a secret. After you create a secret like an access key pair or a password in AWS, you cannot view them again. The external ID for a role can be seen by anyone with permission to view the role.

In a multi-tenant environment where you support multiple customers with different AWS accounts, we recommend using one external ID per AWS account. This ID should be a random string generated by the third party.

To require that the third party provides an external ID when assuming a role, update the role's trust policy with the external ID of your choice.

To provide an external ID when you assume a role, use the AWS CLI or AWS API to assume that role. For more information, see the STS [AssumeRole](#) API operation, or the STS [assume-role](#) CLI operation.

For example, let's say that you decide to hire a third-party company called Example Corp to monitor your AWS account and help optimize costs. In order to track your daily spending, Example Corp needs to access your AWS resources. Example Corp also monitors many other AWS accounts for other customers.

Do not give Example Corp access to an IAM user and its long-term credentials in your AWS account. Instead, use an IAM role and its temporary security credentials. An IAM role provides a mechanism to allow a third party to access your AWS resources without needing to share long-term credentials (for example, an IAM user's access key).

You can use an IAM role to establish a trusted relationship between your AWS account and the Example Corp account. After this relationship is established, a member of the Example Corp account can call the AWS STS [AssumeRole](#) API to obtain temporary security credentials. The Example Corp members can then use the credentials to access AWS resources in your account.

Note

For more information about the [AssumeRole](#) and other AWS API operations that you can call to obtain temporary security credentials, see [Requesting temporary security credentials \(p. 303\)](#).

Here's a more detailed breakdown of this scenario:

1. You hire Example Corp, so they create a unique customer identifier for you. They provide you with this unique customer ID and their AWS account number. You need this information to create an IAM role in the next step.

Note

Example Corp can use any string value they want for the ExternalId, as long as it is unique for each customer. It can be a customer account number or even a random string of characters, as long as no two customers have the same value. It is not intended to be a 'secret'. Example Corp must provide the ExternalId value to each customer. What is crucial is that it must be generated by Example Corp and **not** their customers.

2. You sign in to AWS and create an IAM role that gives Example Corp access to your resources. Like any IAM role, the role has two policies, a permission policy and a trust policy. The role's trust policy specifies who can assume the role. In our sample scenario, the policy specifies the AWS account number of Example Corp as the Principal. This allows identities from that account to assume the role. In addition, you add a Condition element to the trust policy. This Condition tests the ExternalId context key to ensure that it matches the unique customer ID from Example Corp. For example:

```
"Principal": {"AWS": "Example Corp's AWS Account ID"},  
"Condition": {"StringEquals": {"sts:ExternalId": "Unique ID Assigned by Example Corp"}}
```

3. The permission policy for the role specifies what the role allows someone to do. For example, you could specify that the role allows someone to manage only your Amazon EC2 and Amazon RDS resources but not your IAM users or groups. In our sample scenario, you use the permission policy to give Example Corp read-only access to all of the resources in your account.
4. After you create the role, you provide the Amazon Resource Name (ARN) of the role to Example Corp.
5. When Example Corp needs to access your AWS resources, someone from the company calls the AWS sts:AssumeRole API. The call includes the ARN of the role to assume and the ExternalId parameter that corresponds to their customer ID.

If the request comes from someone using Example Corp's AWS account, and if the role ARN and the external ID are correct, the request succeeds. It then provides temporary security credentials that Example Corp can use to access the AWS resources that your role allows.

In other words, when a role policy includes an external ID, anyone who wants to assume the role must be a principal in the role and must include the correct external ID.

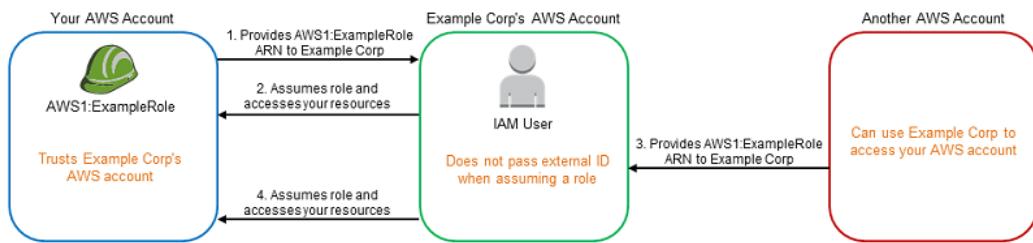
Why use an external ID?

In abstract terms, the external ID allows the user that is assuming the role to assert the circumstances in which they are operating. It also provides a way for the account owner to permit the role to be assumed only under specific circumstances. The primary function of the external ID is to address and prevent the "confused deputy" problem.

The confused deputy problem

To continue the previous example, Example Corp requires access to certain resources in your AWS account. But in addition to you, Example Corp has other customers and needs a way to access each customer's AWS resources. Instead of asking its customers for their AWS account access keys, which are secrets that should never be shared, Example Corp requests a role ARN from each customer. But another Example Corp customer might be able to guess or obtain your role ARN. That customer could then use your role ARN to gain access to your AWS resources by way of Example Corp. This form of permission escalation is known as the confused deputy problem.

The following diagram illustrates the confused deputy problem.



This diagram assumes the following:

- **AWS1** is your AWS account.
- **AWS1:ExampleRole** is a role in your account. This role's trust policy trusts Example Corp by specifying Example Corp's AWS account as the one that can assume the role.

Here's what happens:

1. When you start using Example Corp's service, you provide the ARN of **AWS1:ExampleRole** to Example Corp.
2. Example Corp uses that role ARN to obtain temporary security credentials to access resources in your AWS account. In this way, you are trusting Example Corp as a "deputy" that can act on your behalf.
3. Another AWS customer also starts using Example Corp's service, and this customer also provides the ARN of **AWS1:ExampleRole** for Example Corp to use. Presumably the other customer learned or guessed the **AWS1:ExampleRole**, which isn't a secret.
4. When the other customer asks Example Corp to access AWS resources in (what it claims to be) its account, Example Corp uses **AWS1:ExampleRole** to access resources in your account.

This is how the other customer could gain unauthorized access to your resources. Because this other customer was able to trick Example Corp into unwittingly acting on your resources, Example Corp is now a "confused deputy."

How does an external ID prevent the confused deputy problem?

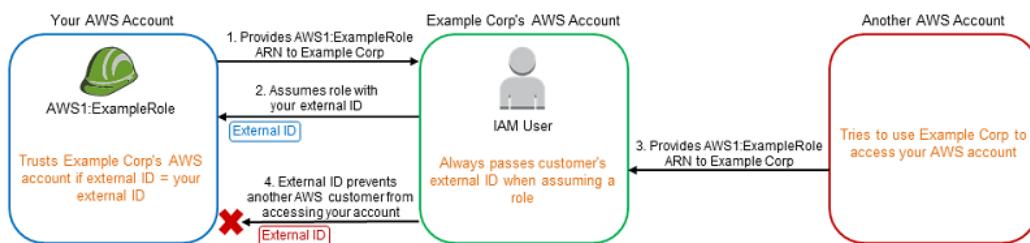
You address the confused deputy problem by including the `ExternalId` condition check in the role's trust policy. The "deputy" company inserts a unique external ID value for each customer into the request for AWS credentials. The external ID is a customer ID value that must be unique among Example Corp's customers and is out of the control of Example Corp's customers. This is why you get it from Example Corp and you don't come up with it on your own. This helps prevent one customer from successfully impersonating another customer. Example Corp always inserts the customer's assigned external ID, so you should never see a request coming from Example Corp with any external ID except your own.

In our scenario, imagine Example Corp's unique identifier for you is "12345," and its identifier for the other customer is "67890." These identifiers are simplified for this scenario. Generally, these identifiers are GUIDs. Assuming that these identifiers are unique among Example Corp's customers, they are sensible values to use for the external ID.

Example Corp gives the external ID value of "12345" to you. You must then add a `Condition` element to the role's trust policy that requires the `sts:ExternalId` value to be 12345, like this:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Principal": {"AWS": "Example Corp's AWS Account ID"},
            "Condition": {"StringEquals": {"sts:ExternalId": "12345"}}
        }
    ]
}
```

The `Condition` element in this policy allows Example Corp to assume the role only when the `AssumeRole` API call includes the external ID value of "12345". Example Corp makes sure that whenever it assumes a role on behalf of a customer, it always includes that customer's external ID value in the `AssumeRole` call. Even if another customer supplies Example Corp with your ARN, it cannot control the external ID that Example Corp includes in its request to AWS. This helps prevent an unauthorized customer from gaining access to your resources, as shown in the following diagram.



1. As before, when you start using Example Corp's service, you provide the ARN of **AWS1:ExampleRole** to Example Corp.
2. When Example Corp uses that role ARN to assume the role **AWS1:ExampleRole**, Example Corp includes your external ID ("12345") in the `AssumeRole` API call. The external ID matches the role's trust policy, so the `AssumeRole` API call succeeds and Example Corp obtains temporary security credentials to access resources in your AWS account.
3. Another AWS customer also starts using Example Corp's service, and as before, this customer also provides the ARN of **AWS1:ExampleRole** for Example Corp to use.
4. But this time, when Example Corp attempts to assume the role **AWS1:ExampleRole**, it provides the external ID associated with the other customer ("67890"). The other customer has no way to change this. Example Corp does this because the request to use the role came from the other customer, so "67890" indicates the circumstance in which Example Corp is acting. Because you added a condition with your own external ID ("12345") to the trust policy of **AWS1:ExampleRole**, the `AssumeRole` API

call fails. The other customer is prevented from gaining unauthorized access to resources in your account (indicated by the red "X" in the diagram).

The external ID helps prevent any other customer from tricking Example Corp into unwittingly accessing your resources—it mitigates the confused deputy problem.

When should I use an external ID?

Use an external ID in the following situations:

- You are an AWS account owner and you have configured a role for a third party that accesses other AWS accounts in addition to yours. You should ask the third party for an external ID that it includes when it assumes your role. Then you check for that external ID in your role's trust policy. Doing so ensures that the external party can assume your role only when it is acting on your behalf.
- You are in the position of assuming roles on behalf of different customers like Example Corp in our previous scenario. You should assign a unique external ID to each customer and instruct them to add the external ID to their role's trust policy. You must then ensure that you always include the correct external ID in your requests to assume roles.

You probably already have a unique identifier for each of your customers, and this unique ID is sufficient for use as an external ID. The external ID is not a special value that you need to create explicitly, or track separately, just for this purpose.

You should always specify the external ID in your `AssumeRole` API calls. In addition when a customer gives you a role ARN, test whether you can assume the role both with and without the correct external ID. If you can assume the role without the correct external ID, don't store the customer's role ARN in your system. Wait until your customer has updated the role trust policy to require the correct external ID. In this way you help your customers to do the right thing, which helps to keep both of you protected against the confused deputy problem.

Creating a role to delegate permissions to an AWS service

Many AWS services require that you use roles to allow the service to access resources in other services on your behalf. A role that a service assumes to perform actions on your behalf is called a [service role \(p. 168\)](#). When a role serves a specialized purpose for a service, it is categorized as a [service role for EC2 instances \(p. 168\)](#) (for example), or a [service-linked role \(p. 168\)](#). To see what services support using service-linked roles, or whether a service supports any form of temporary credentials, see [AWS services that work with IAM \(p. 611\)](#). To learn how an individual service uses roles, choose the service name in the table to view the documentation for that service.

For information about how roles help you to delegate permissions, see [Roles terms and concepts \(p. 168\)](#).

Service role permissions

You must configure permissions to allow an IAM entity (user or role) to create or edit a service role.

Note

The ARN for a service-linked role includes a service principal, which is indicated in the policies below as `SERVICE-NAME.amazonaws.com`. Do not try to guess the service principal, because it is case sensitive and the format can vary across AWS services. To view the service principal for a service, see its service-linked role documentation.

To allow an IAM entity to create a specific service role

Add the following policy to the IAM entity that needs to create the service role. This policy allows you to create a service role for the specified service and with a specific name. You can then attach managed or inline policies to that role.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:AttachRolePolicy",
                "iam:CreateRole",
                "iam:PutRolePolicy"
            ],
            "Resource": "arn:aws:iam::*:role/SERVICE-ROLE-NAME"
        }
    ]
}
```

To allow an IAM entity to create any service role

Add the following statement to the permissions policy for the IAM entity that needs to create a service role. This statement allows you to create any service role for any service, and then attach managed or inline policies to that role.

```
{
    "Effect": "Allow",
    "Action": [
        "iam:AttachRolePolicy",
        "iam:CreateRole",
        "iam:PutRolePolicy"
    ],
    "Resource": "*"
}
```

To allow an IAM entity to edit a service role

Add the following policy to the IAM entity that needs to edit the service role.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "EditSpecificServiceRole",
            "Effect": "Allow",
            "Action": [
                "iam:AttachRolePolicy",
                "iam:DeleteRolePolicy",
                "iam:DetachRolePolicy",
                "iam:GetRole",
                "iam:GetRolePolicy",
                "iam>ListAttachedRolePolicies",
                "iam>ListRolePolicies",
                "iam:PutRolePolicy",
                "iam:UpdateRole",
                "iam:UpdateRoleDescription"
            ],
            "Resource": "arn:aws:iam::*:role/SERVICE-ROLE-NAME"
        },
        {
            "Sid": "ViewRolesAndPolicies",
            "Effect": "Allow",
            "Action": [
                "iam:GetPolicy",
                "iam>ListRoles"
            ],
            "Resource": "*"
        }
    ]
}
```

```
        "Resource": ""  
    }  
}  
]
```

To allow an IAM entity to delete a specific service role

Add the following statement to the permissions policy for the IAM entity that needs to delete the specified service role.

```
{  
    "Effect": "Allow",  
    "Action": "iam:DeleteRole",  
    "Resource": "arn:aws:iam::*:role/SERVICE-ROLE-NAME"  
}
```

To allow an IAM entity to delete any service role

Add the following statement to the permissions policy for the IAM entity that needs to delete a service role.

```
{  
    "Effect": "Allow",  
    "Action": "iam:DeleteRole",  
    "Resource": "*"  
}
```

Creating a role for an AWS service (console)

You can use the AWS Management Console to create a role for a service. Because some services support more than one service role, see the [AWS documentation](#) for your service to see which use case to choose. You can learn how to assign the necessary trust and permissions policies to the role so that the service can assume the role on your behalf. The steps that you can use to control the permissions for your role can vary, depending on how the service defines the use cases, and whether or not you create a service-linked role.

To create a role for an AWS service (console)

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 IAM console, choose **Roles**, and then choose **Create role**.
3. For **Select type of trusted entity**, choose **AWS service**.
4. Choose the service that you want to allow to assume this role.
5. Choose the use case for your service. If the specified service has only one use case, it is selected for you. Use cases are defined by the service to include the trust policy that the service requires. Then choose **Next: Permissions**.
6. If possible, select the policy to use for the permissions policy or choose **Create policy** to open a new browser tab and create a new policy from scratch. For more information, see step 4 in the procedure [Creating IAM policies \(console\) \(p. 439\)](#). After you create the policy, close that tab and return to your original tab. Select the check box next to the permissions policies that you want the service to have.

Depending on the use case that you selected, the service might allow you to do any of the following:

- Nothing, because the service defines the permissions for the role
- Allow you to choose from a limited set of permissions
- Allow you to choose from any permissions

- Allow you to select no policies at this time, create the policies later, and then attach them to the role
7. (Optional) Set a [permissions boundary \(p. 365\)](#). This is an advanced feature that is available for service roles, but not service-linked roles.

Open the **Set permissions boundary** section and choose **Use a permissions boundary to control the maximum role permissions**. IAM includes a list of the AWS managed and customer managed policies in your account. Select the policy to use for the permissions boundary or choose **Create policy** to open a new browser tab and create a new policy from scratch. For more information, see step 4 in the procedure [Creating IAM policies \(console\) \(p. 439\)](#). After you create the policy, close that tab and return to your original tab to select the policy to use for the permissions boundary.

8. Choose **Next: Tags**.
9. (Optional) Add metadata to the role by attaching tags as key–value pairs. For more information about using tags in IAM, see [Tagging IAM users and roles \(p. 289\)](#).
10. Choose **Next: Review**.
11. For **Role name**, the degree of role name customization is defined by the service. If the service defines the role's name, this option is not editable. In other cases, the service might define a prefix for the role and allow you to type an optional suffix. Some services allow you to specify the entire name of your role.

If possible, type a role name or role name suffix. Role names must be unique within your AWS account. They are not distinguished by case. For example, you cannot create roles named both **PRODROLE** and **prodrole**. Because other AWS resources might reference the role, you cannot edit the name of the role after it has been created.

12. (Optional) For **Role description**, type a description for the new role.
13. Review the role and then choose **Create role**.

Creating a role for a service (AWS CLI)

Creating a role from the AWS CLI involves multiple steps. When you use the console to create a role, many of the steps are done for you, but with the AWS CLI you must explicitly perform each step yourself. You must create the role and then assign a permissions policy to the role. If the service you are working with is Amazon EC2, then you must also create an instance profile and add the role to it. Optionally, you can also set the [permissions boundary \(p. 365\)](#) for your role.

To create a role for an AWS service from the AWS CLI

1. Create a role: [aws iam create-role](#)
2. Attach a managed permissions policy to the role: [aws iam attach-role-policy](#)

or

Create an inline permissions policy for the role: [aws iam put-role-policy](#)
3. (Optional) Add custom attributes to the role by attaching tags: [aws iam tag-role](#)

For more information, see [Managing tags on IAM entities \(AWS CLI or AWS API\) \(p. 292\)](#).
4. (Optional) Set the [permissions boundary \(p. 365\)](#) for the role: [aws iam put-role-permissions-boundary](#)

A permissions boundary controls the maximum permissions that a role can have. Permissions boundaries are an advanced AWS feature.

If you are going to use the role with Amazon EC2 or another AWS service that uses Amazon EC2, you must store the role in an instance profile. An instance profile is a container for a role that can be attached

to an Amazon EC2 instance when launched. An instance profile can contain only one role, and that limit cannot be increased. If you create the role using the AWS Management Console, the instance profile is created for you with the same name as the role. For more information about instance profiles, see [Using instance profiles \(p. 270\)](#). For information about how to launch an EC2 instance with a role, see [Controlling Access to Amazon EC2 Resources](#) in the *Amazon EC2 User Guide for Linux Instances*.

To create an instance profile and store the role in it (AWS CLI)

1. Create an instance profile: [aws iam create-instance-profile](#)
2. Add the role to the instance profile: [aws iam add-role-to-instance-profile](#)

The AWS CLI example command set below demonstrates the first two steps for creating a role and attaching permissions. It also shows the two steps for creating an instance profile and adding the role to the profile. This example trust policy allows the Amazon EC2 service to assume the role and view the `example_bucket` Amazon S3 bucket. The example also assumes that you are running on a client computer running Windows and have already configured your command line interface with your account credentials and Region. For more information, see [Configuring the AWS Command Line Interface](#).

In this example, include the following trust policy in the first command when you create the role. This trust policy allows the Amazon EC2 service to assume the role.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {"Service": "ec2.amazonaws.com"},
      "Action": "sts:AssumeRole"
    }
}
```

When you use the second command, you must attach a permissions policy to the role. The following example permissions policy allows the role to perform only the `ListBucket` action on the `example_bucket` Amazon S3 bucket.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3>ListBucket",
      "Resource": "arn:aws:s3:::example_bucket"
    }
}
```

To create this `Test-Role-for-EC2` role, you must first save the previous trust policy with the name `trustpolicyforec2.json` and the previous permissions policy with the name `permissionspolicyforec2.json` to the `policies` directory in your local C: drive. You can then use the following commands to create the role, attach the policy, create the instance profile, and add the role to the instance profile.

```
# Create the role and attach the trust policy that allows EC2 to assume this role.
$ aws iam create-role --role-name Test-Role-for-EC2 --assume-role-policy-document file://C:\\policies\\trustpolicyforec2.json

# Embed the permissions policy (in this example an inline policy) to the role to specify
# what it is allowed to do.
$ aws iam put-role-policy --role-name Test-Role-for-EC2 --policy-name Permissions-Policy-
For-Ec2 --policy-document file://C:\\policies\\permissionspolicyforec2.json
```

```
# Create the instance profile required by EC2 to contain the role
$ aws iam create-instance-profile --instance-profile-name EC2-ListBucket-S3

# Finally, add the role to the instance profile
$ aws iam add-role-to-instance-profile --instance-profile-name EC2-ListBucket-S3 --role-name Test-Role-for-EC2
```

When you launch the EC2 instance, specify the instance profile name in the **Configure Instance Details** page if you use the AWS console. If you use the `aws ec2 run-instances` CLI command, specify the `--iam-instance-profile` parameter.

Creating a role for a service (AWS API)

Creating a role from the AWS API involves multiple steps. When you use the console to create a role, many of the steps are done for you, but with the API you must explicitly perform each step yourself. You must create the role and then assign a permissions policy to the role. If the service you are working with is Amazon EC2, then you must also create an instance profile and add the role to it. Optionally, you can also set the [permissions boundary \(p. 365\)](#) for your role.

To create a role for an AWS service (AWS API)

1. Create a role: [CreateRole](#)

For the role's trust policy, you can specify a file location.

2. Attach a managed permissions policy to the role: [AttachRolePolicy](#)

or

Create an inline permissions policy for the role: [PutRolePolicy](#)

3. (Optional) Add custom attributes to the user by attaching tags: [TagRole](#)

For more information, see [Managing tags on IAM entities \(AWS CLI or AWS API\) \(p. 292\)](#).

4. (Optional) Set the [permissions boundary \(p. 365\)](#) for the role: [PutRolePermissionsBoundary](#)

A permissions boundary controls the maximum permissions that a role can have. Permissions boundaries are an advanced AWS feature.

If you are going to use the role with Amazon EC2 or another AWS service that uses Amazon EC2, you must store the role in an instance profile. An instance profile is a container for a role. Each instance profile can contain only one role, and that limit cannot be increased. If you create the role in the AWS Management Console, the instance profile is created for you with the same name as the role. For more information about instance profiles, see [Using instance profiles \(p. 270\)](#). For information about how to launch an Amazon EC2 instance with a role, see [Controlling Access to Amazon EC2 Resources](#) in the *Amazon EC2 User Guide for Linux Instances*.

To create an instance profile and store the role in it (AWS API)

1. Create an instance profile: [CreateInstanceProfile](#)
2. Add the role to the instance profile: [AddRoleToInstanceProfile](#)

Creating a role for a third-party Identity Provider (federation)

You can use identity providers instead of creating IAM users in your AWS account. With an identity provider (IdP), you can manage your user identities outside of AWS and give these external user identities permissions to access AWS resources in your account. For more information about federation and identity providers, see [Identity providers and federation \(p. 176\)](#).

Creating a role for federated users (console)

The procedures for creating a role for federated users depend on your choice of third-party providers:

- For Web Identity or OpenID Connect (OIDC), see [Creating a role for web identity or OpenID connect federation \(console\) \(p. 237\)](#).
- For SAML 2.0, see [Creating a role for SAML 2.0 federation \(console\) \(p. 241\)](#).

Creating a role for federated access (AWS CLI)

The steps to create a role for the supported identity providers (OIDC or SAML) from the AWS CLI are identical. The difference is in the contents of the trust policy that you create in the prerequisite steps. Begin by following the steps in the **Prerequisites** section for the type of provider you are using:

- For an OIDC provider, see [Prerequisites for creating a role for web identity or OIDC \(p. 237\)](#).
- For a SAML provider, see [Prerequisites for creating a role for SAML \(p. 241\)](#).

Creating a role from the AWS CLI involves multiple steps. When you use the console to create a role, many of the steps are done for you, but with the AWS CLI you must explicitly perform each step yourself. You must create the role and then assign a permissions policy to the role. Optionally, you can also set the [permissions boundary \(p. 365\)](#) for your role.

To create a role for identity federation (AWS CLI)

1. Create a role: `aws iam create-role`
2. Attach a permissions policy to the role: `aws iam attach-role-policy`

or

Create an inline permissions policy for the role: `aws iam put-role-policy`
3. (Optional) Add custom attributes to the role by attaching tags: `aws iam tag-role`

For more information, see [Managing tags on IAM entities \(AWS CLI or AWS API\) \(p. 292\)](#).
4. (Optional) Set the [permissions boundary \(p. 365\)](#) for the role: `aws iam put-role-permissions-boundary`

A permissions boundary controls the maximum permissions that a role can have. Permissions boundaries are an advanced AWS feature.

The following example shows the first two, and most common, steps for creating an identity provider role in a simple environment. This example allows any user in the 123456789012 account to assume the role and view the `example_bucket` Amazon S3 bucket. This example also assumes that you are running the AWS CLI on a computer running Windows, and have already configured the AWS CLI with your credentials. For more information, see [Configuring the AWS Command Line Interface](#).

In this example, include the following trust policy in the first command when you create the role. This trust policy allows users in the 123456789012 account to assume the role using the `AssumeRole` operation, but only if the user provides MFA authentication using the `SerialNumber` and `TokenCode` parameters. For more information about MFA, see [Using multi-factor authentication \(MFA\) in AWS \(p. 111\)](#).

The following example trust policy is designed for a mobile app if the user signs in using Amazon Cognito. In this example, `us-east-12345678-ffff-ffff-ffff-123456` represents the identity pool ID assigned by Amazon Cognito.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "RoleForCognito",
            "Effect": "Allow",
            "Principal": {"Federated": "cognito-identity.amazonaws.com"},
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {"StringEquals": {"cognito-identity.amazonaws.com:aud": "us-east-1:12345678-ffff-ffff-123456"}}
        }
    ]
}
```

The following permissions policy allows anyone who assumes the role to perform only the `ListBucket` action on the `example_bucket` Amazon S3 bucket.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3>ListBucket",
            "Resource": "arn:aws:s3:::example_bucket"
        }
    ]
}
```

To create this `Test-Cognito-Role` role, you must first save the previous trust policy with the name `trustpolicyforcognitofederation.json` and the previous permissions policy with the name `permsspolicyforcognitofederation.json` to the `policies` folder in your local C: drive. You can then use the following commands to create the role and attach the inline policy.

```
# Create the role and attach the trust policy that enables users in an account to assume
the role.
$ aws iam create-role --role-name Test-Cognito-Role --assume-role-policy-document file://C:
\policies\trustpolicyforcognitofederation.json

# Attach the permissions policy to the role to specify what it is allowed to do.
aws iam put-role-policy --role-name Test-Cognito-Role --policy-name Perms-Policy-For-
CognitoFederation --policy-document file://C:\policies\permsspolicyforcognitofederation.json
```

Creating a role for federated access (AWS API)

The steps to create a role for the supported identity providers (OIDC or SAML) from the AWS CLI are identical. The difference is in the contents of the trust policy that you create in the prerequisite steps. Begin by following the steps in the **Prerequisites** section for the type of provider you are using:

- For an OIDC provider, see [Prerequisites for creating a role for web identity or OIDC \(p. 237\)](#).
- For a SAML provider, see [Prerequisites for creating a role for SAML \(p. 241\)](#).

To create a role for identity federation (AWS API)

1. Create a role: [CreateRole](#)
2. Attach a permissions policy to the role: [AttachRolePolicy](#)

or

Create an inline permissions policy for the role: [PutRolePolicy](#)

3. (Optional) Add custom attributes to the user by attaching tags: [TagRole](#)

For more information, see [Managing tags on IAM entities \(AWS CLI or AWS API\) \(p. 292\)](#).

4. (Optional) Set the [permissions boundary \(p. 365\)](#) for the role: [PutRolePermissionsBoundary](#)

A permissions boundary controls the maximum permissions that a role can have. Permissions boundaries are an advanced AWS feature.

Creating a role for web identity or OpenID connect federation (console)

You can use Web Identity or OpenID Connect Federation (OIDC) identity providers instead of creating IAM users in your AWS account. With an identity provider (IdP), you can manage your user identities outside of AWS and give these external user identities permissions to access AWS resources in your account. For more information about federation and identity providers, see [Identity providers and federation \(p. 176\)](#).

Prerequisites for creating a role for web identity or OIDC

Before you can create a role for web identity federation, you must first complete the following prerequisite steps.

To prepare to create a role for web identity federation

1. Sign up as a developer with one or more IdPs. If you are creating an app that needs access to your AWS resources, you also configure your app with the provider information. When you do, the provider gives you an application or audience ID that's unique to your app. (Different providers use different terminology for this process. This guide uses the term *configure* for the process of identifying your app with the provider.) You can configure multiple apps with each provider, or multiple providers with a single app. View information about using the identity providers:
 - [Login with Amazon Developer Center](#)
 - [Add Facebook Login to Your App or Website](#) on the Facebook developers site.
 - [Using OAuth 2.0 for Login \(OpenID Connect\)](#) on the Google developers site.
2. After getting the required information from the identity provider, create an identity provider in IAM. For more information, see [Creating OpenID Connect \(OIDC\) identity providers \(p. 186\)](#).
3. Prepare the policies for the role that the IdP-authenticated users will assume. As with any role, a role for a mobile app includes two policies. One is the trust policy that specifies who can assume the role. The other is the permissions policy that specifies the AWS actions and resources that the mobile app is allowed or denied access to.

For web identity providers, we recommend that you use [Amazon Cognito](#) to manage identities. In this case, use a trust policy similar to this example.

```
{  
    "Version": "2012-10-17",  
    "Statement": {  
        "Effect": "Allow",  
        "Principal": {"Federated": "cognito-identity.amazonaws.com"},  
        "Action": "sts:AssumeRoleWithWebIdentity",  
        "Condition": {  
            "StringEquals": {"cognito-identity.amazonaws.com:aud": "us-east-2:12345678-  
abcd-abcd-abcd-123456"},  
            "ForAnyValue:StringLike": {"cognito-identity.amazonaws.com:amr":  
                "unauthenticated"}  
        }  
    }  
}
```

Replace `us-east-2:12345678-abcd-abcd-abcd-123456` with the identity pool ID that Amazon Cognito assigned to you.

If you manually configure a web identity IdP, when you create the trust policy, you must use three values that ensure that only your app can assume the role:

- For the `Action` element, use the `sts:AssumeRoleWithWebIdentity` action.
- For the `Principal` element, use the string `{"Federated":providerUrl/providerArn"}`.
 - For some common OpenID Connect (OIDC) IdPs, the `providerUrl` is a URL. The following examples include methods to specify the principal for some common IdPs:

```
"Principal": {"Federated": "cognito-identity.amazonaws.com"}
```

```
"Principal": {"Federated": "www.amazon.com"}
```

```
"Principal": {"Federated": "graph.facebook.com"}
```

```
"Principal": {"Federated": "accounts.google.com"}
```

- For other OIDC providers, use the ARN of the OIDC identity provider that you created in [Step 2](#), such as the following example:

```
"Principal": {"Federated": "arn:aws:iam::123456789012:oidc-provider/server.example.com"}
```

- For the `Condition` element, use a `StringEquals` condition to limit permissions. Test the identity pool ID for Amazon Cognito) or the app ID for other providers. It should match the app ID that you received when you configured the app with the IdP. This ensures that the request is coming from your app. Create a condition element similar to the following examples, depending on the IdP that you are using:

```
"Condition": {"StringEquals": {"cognito-identity.amazonaws.com:aud": "us-east:12345678-ffff-ffff-ffff-123456"}}
```

```
"Condition": {"StringEquals": {"www.amazon.com:app_id": "amzn1.application-oa2-123456"}}
```

```
"Condition": {"StringEquals": {"graph.facebook.com:app_id": "111222333444555"}}
```

```
"Condition": {"StringEquals": {"accounts.google.com:aud": "66677788899900pro0"}}
```

For OIDC providers, use the fully qualified URL of the OIDC IdP with the `aud` context key, such as the following example:

```
"Condition": {"StringEquals": {"server.example.com:aud": "appid_from_oidc_idp"}}
```

Notice that the values for the principal in the trust policy for the role are specific to an IdP. A role can specify only one principal. Therefore, if the mobile app allows users to sign in from more than one IdP, you must create a separate role for each IdP that you want to support. Therefore, you should create separate trust policies for each IdP.

The following example trust policy is designed for a mobile app if the user signs in from Login with Amazon. In the example, `amzn1.application-oa2-123456` represents the app ID that Amazon assigned when you configured the app using Login with Amazon.

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

```

        "Effect": "Allow",
        "Principal": {"Federated": "www.amazon.com"},
        "Action": "sts:AssumeRoleWithWebIdentity",
        "Condition": {"StringEquals": {"www.amazon.com:app_id": "amzn1.application-
oa2-123456"}}
    }]
}

```

The following example trust policy is designed for a mobile app if the user signs in from Facebook. In this example, **111222333444555** represents the app ID assigned by Facebook.

```

{
    "Version": "2012-10-17",
    "Statement": [{
        "Sid": "RoleForFacebook",
        "Effect": "Allow",
        "Principal": {"Federated": "graph.facebook.com"},
        "Action": "sts:AssumeRoleWithWebIdentity",
        "Condition": {"StringEquals": {"graph.facebook.com:app_id":
"111222333444555"}}
    }]
}

```

The following example trust policy is designed for a mobile app if the user signs in from Google. In this example, **666777888999000** represents the app ID assigned by Google.

```

{
    "Version": "2012-10-17",
    "Statement": [{
        "Sid": "RoleForGoogle",
        "Effect": "Allow",
        "Principal": {"Federated": "accounts.google.com"},
        "Action": "sts:AssumeRoleWithWebIdentity",
        "Condition": {"StringEquals": {"accounts.google.com:aud": "666777888999000"}}
    }]
}

```

The following example trust policy is designed for a mobile app if the user signs in using Amazon Cognito. In this example, **us-east:12345678-ffff-ffff-ffff-123456** represents the identity pool ID assigned by Amazon Cognito.

```

{
    "Version": "2012-10-17",
    "Statement": [{
        "Sid": "RoleForCognito",
        "Effect": "Allow",
        "Principal": {"Federated": "cognito-identity.amazonaws.com"},
        "Action": "sts:AssumeRoleWithWebIdentity",
        "Condition": {"StringEquals": {"cognito-identity.amazonaws.com:aud": "us-
east:12345678-ffff-ffff-ffff-123456"}}
    }]
}

```

Creating a role for web identity or OIDC

After you complete the prerequisites, you can create the role in IAM. The following procedure describes how to create the role for web identity/OIDC federation in the AWS Management Console. To create a role from the AWS CLI or AWS API, see the procedures at [Creating a role for a third-party Identity Provider \(federation\) \(p. 234\)](#).

Important

If you are using Amazon Cognito, you should use the Amazon Cognito console to set up the roles. Otherwise, use the IAM console to create a role for web identity federation.

To create an IAM role for web identity federation

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, choose **Roles** and then choose **Create role**.
3. Choose the **Web identity** role type.
4. For **Identity provider**, choose the identity provider for your role:
 - If you're creating a role for an individual web identity provider, choose **Login with Amazon, Facebook, or Google**.

Note

You must create a separate role for each identity provider that you want to support.

- If you're creating an advanced scenario role for Amazon Cognito, choose **Amazon Cognito**.

Note

You need to manually create a role for use with Amazon Cognito only when you are working on an advanced scenario. Otherwise, Amazon Cognito can create roles for you. For more information about Amazon Cognito, see [Amazon Cognito Identity](#) in the [AWS Mobile SDK for iOS Developer Guide](#) and [Amazon Cognito Identity](#) in the [AWS Mobile SDK for Android Developer Guide](#).

5. Type the identifier for your application. The label of the identifier changes depending on which provider you choose:
 - If you're creating a role for Login with Amazon, type the app ID into the **Application ID** box.
 - If you're creating a role for Facebook, type the app ID into the **Application ID** box.
 - If you're creating a role for Google, type the audience name into the **Audience** box.
 - If you're creating a role for Amazon Cognito, type the ID of the identity pool that you have created for your Amazon Cognito applications into the **Identity Pool ID** box.
6. (Optional) Click **Add condition (optional)** to create additional conditions that must be met before users of your application can use the permissions that the role grants. For example, you can add a condition that grants access to AWS resources only for a specific IAM user ID.
7. Review your web identity information and then choose **Next: Permissions**.
8. IAM includes a list of the AWS managed and customer managed policies in your account. Select the policy to use for the permissions policy or choose **Create policy** to open a new browser tab and create a new policy from scratch. For more information, see step 4 in the procedure [Creating IAM policies \(console\) \(p. 439\)](#). After you create the policy, close that tab and return to your original tab. Select the check box next to the permissions policies that you want web identity users to have. If you prefer, you can select no policies at this time, and then attach policies to the role later. By default, a role has no permissions.
9. (Optional) Set a [permissions boundary \(p. 365\)](#). This is an advanced feature.

Open the **Set permissions boundary** section and choose **Use a permissions boundary to control the maximum role permissions**. Select the policy to use for the permissions boundary.
10. Choose **Next: Tags**.

11. (Optional) Add metadata to the role by attaching tags as key-value pairs. For more information about using tags in IAM, see [Tagging IAM users and roles \(p. 289\)](#).
12. Choose **Next: Review**.
13. For **Role name**, type a role name. Role names must be unique within your AWS account. They are not distinguished by case. For example, you cannot create roles named both **PRODROLE** and **prodrole**. Because other AWS resources might reference the role, you cannot edit the name of the role after it has been created.
14. (Optional) For **Role description**, type a description for the new role.
15. Review the role and then choose **Create role**.

Creating a role for SAML 2.0 federation (console)

You can use SAML 2.0 federation instead of creating IAM users in your AWS account. With an identity provider (IdP), you can manage your user identities outside of AWS and give these external user identities permissions to access AWS resources in your account. For more information about federation and identity providers, see [Identity providers and federation \(p. 176\)](#).

Prerequisites for creating a role for SAML

Before you can create a role for SAML 2.0 federation, you must first complete the following prerequisite steps:

To prepare to create a role for SAML 2.0 federation

1. Before you create a role for SAML-based federation, you must create a SAML provider in IAM. For more information, see [Creating IAM SAML identity providers \(p. 193\)](#).
2. Prepare the policies for the role that the SAML 2.0–authenticated users will assume. As with any role, a role for the SAML federation includes two policies. One is the role trust policy that specifies who can assume the role. The other is the IAM permissions policy that specifies the AWS actions and resources that the federated user is allowed or denied access to.

When you create the trust policy for your role, you must use three values that ensure that the role can be assumed only by your application:

- For the **Action** element, use the `sts:AssumeRoleWithSAML` action.
- For the **Principal** element, use the string `{"Federated": "ARNofIdentityProvider"}`. Replace `ARNofIdentityProvider` with the ARN of the [SAML identity provider \(p. 182\)](#) that you created in [Step 1](#).
- For the **Condition** element, use a `StringEquals` condition to test that the `saml:aud` attribute from the SAML response matches the SAML federation endpoint for AWS.

The following example trust policy is designed for a SAML federated user:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {"Effect": "Allow",  
         "Action": "sts:AssumeRoleWithSAML",  
         "Principal": {"Federated": "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:saml-provider/PROVIDER-NAME"},  
         "Condition": {"StringEquals": {"SAML:aud": "https://signin.aws.amazon.com/saml"}}  
    ]  
}
```

Replace the principal ARN with the actual ARN for the SAML provider that you created in IAM. It will have your own account ID and provider name.

Creating a role for SAML

After you complete the prerequisite steps, you can create the role for SAML-based federation.

To create a role for SAML-based federation

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 IAM console, choose **Roles** and then choose **Create role**.
3. Choose the **SAML 2.0 federation** role type.
4. For **SAML Provider**, choose the provider for your role.
5. Choose the SAML 2.0 access level method.
 - Choose **Allow programmatic access only** to create a role that can be assumed programmatically from the AWS API or AWS CLI.
 - Choose **Allow programmatic and AWS Management Console access** to create a role that can be assumed programmatically and from the console.

The roles created by both are similar, but the role that can also be assumed from the console includes a trust policy with a particular condition. That condition explicitly ensures that the SAML audience (SAML:aud attribute) is set to the AWS sign-in endpoint for SAML (<https://signin.aws.amazon.com/saml>).

6. If you're creating a role for programmatic access, choose an attribute from the **Attribute** list. Then in the **Value** box, type a value to include in the role. This restricts role access to users from the identity provider whose SAML authentication response (assertion) includes the attributes that you specify. You must specify at least one attribute to ensure that your role is limited to a subset of users at your organization.

If you're creating a role for programmatic and console access, the SAML:aud attribute is automatically added and set to the URL of the AWS SAML endpoint (<https://signin.aws.amazon.com/saml>).

7. To add more attribute-related conditions to the trust policy, choose **Add condition (optional)**, select the additional condition, and specify a value.

Note

The list includes the most commonly used SAML attributes. IAM supports additional attributes that you can use to create conditions. For a list of the supported attributes, see [Available Keys for SAML Federation](#). If you need a condition for a supported SAML attribute that's not in the list, you can manually add that condition. To do that, edit the trust policy after you create the role.

8. Review your SAML 2.0 trust information and then choose **Next: Permissions**.
9. IAM includes a list of the AWS managed and customer managed policies in your account. Select the policy to use for the permissions policy or choose **Create policy** to open a new browser tab and create a new policy from scratch. For more information, see step 4 in the procedure [Creating IAM policies \(console\) \(p. 439\)](#). After you create the policy, close that tab and return to your original tab. Select the check box next to the permissions policies that you want web identity users to have. If you prefer, you can select no policies at this time, and then attach policies to the role later. By default, a role has no permissions.
10. (Optional) Set a [permissions boundary \(p. 365\)](#). This is an advanced feature.

Open the **Set permissions boundary** section and choose **Use a permissions boundary to control the maximum role permissions**. Select the policy to use for the permissions boundary.

11. Choose **Next: Tags**.
12. (Optional) Add metadata to the role by attaching tags as key-value pairs. For more information about using tags in IAM, see [Tagging IAM users and roles \(p. 289\)](#).

13. Choose **Next: Review**.
14. For **Role name**, type a role name. Role names must be unique within your AWS account. They are not distinguished by case. For example, you cannot create roles named both **PRODROLE** and **prodrole**. Because other AWS resources might reference the role, you cannot edit the name of the role after it has been created.
15. (Optional) For **Role description**, type a description for the new role.
16. Review the role and then choose **Create role**.

After you create the role, you complete the SAML trust by configuring your identity provider software with information about AWS. This information includes the roles that you want your federated users to use. This is referred to as configuring the relying party trust between your IdP and AWS. For more information, see [Configuring your SAML 2.0 IdP with relying party trust and adding claims \(p. 196\)](#).

Examples of policies for delegating access

The following examples show how you can allow or grant an AWS account access to the resources in another AWS account. To learn how to create an IAM policy using these example JSON policy documents, see the section called “[Creating policies on the JSON tab](#)” (p. 440).

Topics

- [Using roles to delegate access to another AWS account's resources \(p. 243\)](#)
- [Using a policy to delegate access to services \(p. 243\)](#)
- [Using a resource-based policy to delegate access to an Amazon S3 bucket in another account \(p. 244\)](#)
- [Using a resource-based policy to delegate access to an Amazon SQS queue in another account \(p. 245\)](#)
- [Cannot delegate access when the account is denied access \(p. 245\)](#)

Using roles to delegate access to another AWS account's resources

For a tutorial that shows how to use IAM roles to grant users in one account access to AWS resources that are in another account, see [IAM Tutorial: Delegate access across AWS accounts using IAM roles \(p. 34\)](#).

Important

You can include the ARN for a specific role or user in the **Principal** element of a role trust policy. When you save the policy, AWS transforms the ARN to a unique principal ID. This helps mitigate the risk of someone escalating their privileges by removing and recreating the role or user. You don't normally see this ID in the console, because there is also a reverse transformation back to the ARN when the trust policy is displayed. However, if you delete the role or user, then the relationship is broken. The policy no longer applies, even if you recreate the user or role because it does not match the principal ID stored in the trust policy. When this happens, the principal ID shows up in the console because AWS can no longer map it back to an ARN. The result is that if you delete and recreate a user or role referenced in a trust policy's **Principal** element, you must edit the role to replace the ARN. It is transformed into the new principal ID when you save the policy.

Using a policy to delegate access to services

The following example shows a policy that can be attached to a role. The policy enables two services, Amazon EMR and AWS Data Pipeline, to assume the role. The services can then perform any tasks granted by the permissions policy assigned to the role (not shown). To specify multiple service principals, you do not specify two **Service** elements; you can have only one. Instead, you use an array of multiple service principals as the value of a single **Service** element.

{

```

    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "elasticmapreduce.amazonaws.com",
                    "datapipeline.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}

```

Using a resource-based policy to delegate access to an Amazon S3 bucket in another account

In this example, account A uses a resource-based policy (an Amazon S3 [bucket policy](#)) to grant account B full access to account A's S3 bucket. Then account B creates an IAM user policy to delegate that access to account A's bucket to one of the users in account B.

The S3 bucket policy in account A might look like the following policy. In this example, account A's S3 bucket is named *mybucket*, and account B's account number is 111122223333. It does not specify any individual users or groups in account B, only the account itself.

```

{
    "Version": "2012-10-17",
    "Statement": {
        "Sid": "AccountBAccess1",
        "Effect": "Allow",
        "Principal": {"AWS": "111122223333"},
        "Action": "s3:*",
        "Resource": [
            "arn:aws:s3:::mybucket",
            "arn:aws:s3:::mybucket/*"
        ]
    }
}

```

Alternatively, account A can use Amazon S3 [Access Control Lists \(ACLs\)](#) to grant account B access to an S3 bucket or a single object within a bucket. In that case, the only thing that changes is how account A grants access to account B. Account B still uses a policy to delegate access to an IAM group in account B, as described in the next part of this example. For more information about controlling access on S3 buckets and objects, go to [Access Control](#) in the *Amazon Simple Storage Service Developer Guide*.

The administrator of account B might create the following policy sample. The policy allows read access to a group or user in account B. The preceding policy grants access to account B. However, individual groups and users in account B cannot access the resource until a group or user policy explicitly grants permissions to the resource. The permissions in this policy can only be a subset of those in the preceding cross-account policy. Account B cannot grant more permissions to its groups and users than account A granted to account B in the first policy. 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.

To implement this policy account B uses IAM to attach it to the appropriate user (or group) in account B.

```

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

```

```

    "Action": "s3>List*",
    "Resource": [
        "arn:aws:s3:::mybucket",
        "arn:aws:s3:::mybucket/*"
    ]
}

```

Using a resource-based policy to delegate access to an Amazon SQS queue in another account

In the following example, account A has an Amazon SQS queue that uses a resource-based policy attached to the queue to grant queue access to account B. Then account B uses an IAM group policy to delegate access to a group in account B.

The following example queue policy gives account B permission to perform the `SendMessage` and `ReceiveMessage` actions on account A's queue named `queue1`, but only between noon and 3:00 p.m. on November 30, 2014. Account B's account number is 1111-2222-3333. Account A uses Amazon SQS to implement this policy.

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Principal": {"AWS": "111122223333"},
        "Action": [
            "sns:SendMessage",
            "sns:ReceiveMessage"
        ],
        "Resource": [ "arn:aws:sns:*:123456789012:queue1" ],
        "Condition": {
            "DateGreaterThan": {"aws:CurrentTime": "2014-11-30T12:00Z"},
            "DateLessThan": {"aws:CurrentTime": "2014-11-30T15:00Z"}
        }
    }
}
```

Account B's policy for delegating access to a group in account B might look like the following example. Account B uses IAM to attach this policy to a group (or user).

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": "sns:*",
        "Resource": "arn:aws:sns:*:123456789012:queue1"
    }
}
```

In the preceding IAM user policy example, account B uses a wildcard to grant its user access to all Amazon SQS actions on account A's queue. However account B can delegate access only to the extent that account B has been granted access. The account B group that has the second policy can access the queue only between noon and 3:00 p.m. on November 30, 2014. The user can only perform the `SendMessage` and `ReceiveMessage` actions, as defined in account A's Amazon SQS queue policy.

Cannot delegate access when the account is denied access

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. The deny propagates to the users under that account whether or not the users have existing policies granting them access.

For example, account A writes a bucket policy on account A's S3 bucket that explicitly denies account B access to account A's bucket. But account B writes an IAM user policy that grants a user in account B access to account A's bucket. The explicit deny applied to account A's S3 bucket propagates to the users in account B. It overrides the IAM user policy granting access to the user in account B. (For detailed information how permissions are evaluated, see [Policy evaluation logic \(p. 666\)](#).)

Account A's bucket policy might look like the following policy. In this example, account A's S3 bucket is named *mybucket*, and account B's account number is 1111-2222-3333. Account A uses Amazon S3 to implement this policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AccountBDeny",
      "Effect": "Deny",
      "Principal": {"AWS": "111122223333"},
      "Action": "s3:*",
      "Resource": "arn:aws:s3:::mybucket/*"
    }
  ]
}
```

This explicit deny overrides any policies in account B that provide permission to access the S3 bucket in account A.

Using IAM roles

Before an IAM user, application, or service can use a role that you created, you must grant permissions to switch to the role. You can use any policy attached to one of an IAM user's groups or to the user itself to grant the necessary permissions. This section describes how to grant users permission to use a role. It also explains how the user can switch to a role from the AWS Management Console, the Tools for Windows PowerShell, the AWS Command Line Interface (AWS CLI) and the [AssumeRole](#) API.

Important

When you create a role programmatically instead of in the IAM console, you have an option to add a Path of up to 512 characters in addition to the RoleName, which can be up to 64 characters long. However, if you intend to use a role with the **Switch Role** feature in the AWS Management Console, then the combined Path and RoleName cannot exceed 64 characters.

You can switch roles from the AWS Management Console. You can assume a role by calling an AWS CLI or API operation or by using a custom URL. The method that you use determines who can assume the role and how long the role session can last.

Comparing methods for using roles

Method of assuming the role	Who can assume the role	Method to specify credential lifetime	Credential lifetime (min max default)
AWS Management Console	IAM user (by switching roles (p. 253))	Maximum session duration on the Role Summary page	1h Maximum session duration setting ² 1hr
<code>assume-role</code> CLI or AssumeRole API operation	IAM user or role ¹	duration-seconds CLI or DurationSeconds API parameter	15m Maximum session duration setting ² 1hr

Method of assuming the role	Who can assume the role	Method to specify credential lifetime	Credential lifetime (min max default)
assume-role-with-saml CLI or AssumeRoleWithSAML API operation	Any user authenticated using SAML	duration-seconds CLI or DurationSeconds API parameter	15m Maximum session duration setting ² 1hr
assume-role-with-web-identity CLI or AssumeRoleWithWebIdentity API operation	Any user authenticated using a web identity provider	duration-seconds CLI or DurationSeconds API parameter	15m Maximum session duration setting ² 1hr
Console URL (p. 206) constructed with AssumeRole	IAM user or role	SessionDuration HTML parameter in the URL	15m 12hr 1hr
Console URL (p. 206) constructed with AssumeRoleWithSAML	Any user authenticated using SAML	SessionDuration HTML parameter in the URL	15m 12hr 1hr
Console URL (p. 206) constructed with AssumeRoleWithWebIdentity	Any user authenticated using a web identity provider	SessionDuration HTML parameter in the URL	15m 12hr 1hr

¹ Using the credentials for one role to assume a different role is called [role chaining \(p. 169\)](#). When you use role chaining, your new credentials are limited to a maximum duration of one hour. When you use roles to [grant permissions to applications that run on EC2 instances \(p. 263\)](#), those applications are not subject to this limitation.

² This setting can have a value from 1 hour to 12 hours. For details about modifying the maximum session duration setting, see [Modifying a role \(p. 273\)](#). This setting determines the maximum session duration that you can request when you get the role credentials. For example, when you use the [AssumeRole*](#) API operations to assume a role, you can specify a session length using the DurationSeconds parameter. Use this parameter to specify the length of the role session from 900 seconds (15 minutes) up to the maximum session duration setting for the role. IAM users who switch roles in the console are granted the maximum session duration, or the remaining time in the IAM user's session, whichever is less. Assume that you set a maximum duration of 5 hours on a role. An IAM user that has been signed into the console for 10 hours (out of the default maximum of 12) switches to the role. The available role session duration is 2 hours. To learn how to view the maximum value for your role, see [View the maximum session duration setting for a role \(p. 248\)](#) later in this page.

Note

The maximum session duration setting does not limit sessions that are assumed by AWS services.

Topics

- [View the maximum session duration setting for a role \(p. 248\)](#)
- [Granting a user permissions to switch roles \(p. 248\)](#)
- [Granting a user permissions to pass a role to an AWS service \(p. 251\)](#)
- [Switching to a role \(console\) \(p. 253\)](#)

- [Switching to an IAM role \(AWS CLI\) \(p. 256\)](#)
- [Switching to an IAM role \(Tools for Windows PowerShell\) \(p. 260\)](#)
- [Switching to an IAM role \(AWS API\) \(p. 262\)](#)
- [Using an IAM role to grant permissions to applications running on Amazon EC2 instances \(p. 263\)](#)
- [Revoking IAM role temporary security credentials \(p. 271\)](#)

View the maximum session duration setting for a role

You can specify the maximum session duration for a role using the AWS Management Console or by using the AWS CLI or AWS API. When you use an AWS CLI or API operation to assume a role, you can specify a value for the `DurationSeconds` parameter. You can use this parameter to specify the duration of the role session, from 900 seconds (15 minutes) up to the maximum session duration setting for the role. Before you specify the parameter, you should view this setting for your role. If you specify a value for the `DurationSeconds` parameter that is higher than the maximum setting, the operation fails.

To view a role's maximum session duration (console)

1. In the navigation pane of the IAM console, choose **Roles**.
2. Choose the name of the role that you want to view.
3. Next to **Maximum session duration**, view the maximum session length that is granted for the role. This is the maximum session duration that you can specify in your AWS CLI, or API operation.

To view a role's maximum session duration setting (AWS CLI)

1. If you don't know the name of the role that you want to assume, run the following command to list the roles in your account:
 - `aws iam list-roles`
2. To view the role's maximum session duration, run the following command. Then view the maximum session duration parameter.
 - `aws iam get-role`

To view a role's maximum session duration setting (AWS API)

1. If you don't know the name of the role that you want to assume, call the following operation to list the roles in your account:
 - `ListRoles`
2. To view the role's maximum session duration, run the following operation. Then view the maximum session duration parameter.
 - `GetRole`

Granting a user permissions to switch roles

When an administrator [creates a role for cross-account access \(p. 221\)](#) they establish trust between the account that owns the role and the resources (trusting account) and the account that contains the users (trusted account). To do this, the administrator of the trusting account specifies the trusted account number as the `Principal` in the role's trust policy. That allows *potentially* any user in the trusted account to assume the role. To complete the configuration, the administrator of the trusted account must give specific groups or users in that account permission to switch to the role.

To grant a user permission to switch to a role, the administrator of the trusted account creates a new policy for the user. Or the administrator might edit an existing policy to add the required elements. The administrator can then send the users a link that takes the user to the **Switch Role** page with all the details already filled in. Alternatively, the administrator can provide the user with the account ID number or account alias that contains the role and the role name. The user then goes to the **Switch Role** page and adds the details manually. For details on how a user switches roles, see [Switching to a role \(console\) \(p. 253\)](#).

Note that you can switch roles only when you sign in as an IAM user. You cannot switch roles when you sign in as the AWS account root user.

Important

You cannot switch roles in the AWS Management Console to a role that requires an [ExternalId \(p. 225\)](#) value. You can switch to such a role only by calling the AssumeRole API that supports the `ExternalId` parameter.

Notes

- This topic discusses policies for a *user*, because we are ultimately granting permissions to a user to accomplish a task. However, it is [best practice not to grant permissions directly to an individual user \(p. 528\)](#). For easier management, we recommend assigning policies and granting permissions to IAM groups and then making the users members of the appropriate groups.
- When you switch roles in the AWS Management Console, the console always uses your original credentials to authorize the switch. This applies whether you sign in as an IAM user, as a SAML-federated role, or as a web-identity federated role. For example, if you switch to RoleA, it uses your original user or federated role credentials to determine if you are allowed to assume RoleA. If you then try to switch to RoleB *while you are using RoleA*, your **original** user or federated role credentials are used to authorize your attempt, not the credentials for RoleA.

Topics

- [Creating or editing the policy \(p. 249\)](#)
- [Providing information to the user \(p. 250\)](#)

Creating or editing the policy

A policy that grants a user permission to assume a role must include a statement with the `Allow` effect on the following:

- The `sts:AssumeRole` action
- The Amazon Resource Name (ARN) of the role in a `Resource` element

This is as shown in the following example. Users that get the policy (either through group membership or directly attached) are allowed to switch to the specified role.

Note

If `Resource` is set to `*`, the user can assume any role in any account that trusts the user's account. (In other words, the role's trust policy specifies the user's account as `Principal`). As a best practice, we recommend that you follow the [principle of least privilege](#) and specify the complete ARN for only the roles that the user needs.

The following example shows a policy that lets the user assume roles in only one account. In addition, the policy uses a wildcard (*) to specify that the user can switch to a role only if the role name begins with the letters `Test`.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "sts:AssumeRole",  
            "Resource": "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENs:role/Test*"  
        }  
    ]  
}
```

Note

The permissions that the role grants to the user do not add to the permissions already granted to the user. When a user switches to a role, the user temporarily gives up his or her original permissions in exchange for those granted by the role. When the user exits the role, then the original user permissions are automatically restored. For example, let's say the user's permissions allow working with Amazon EC2 instances, but the role's permissions policy does not grant those permissions. In that case, while using the role, the user cannot work with Amazon EC2 instances in the console. In addition, temporary credentials obtained via AssumeRole do not work with Amazon EC2 instances programmatically.

Providing information to the user

After you create a role and grant your user permissions to switch to it, you must provide the user with the following:

- The name of the role
- The ID or alias of the account that contains the role

You can make things easier for your users by sending them a link that is preconfigured with the account ID and role name. You can see the role link on the final page of the **Create Role** wizard or in the **Role Summary** page for any cross-account enabled role.

You can also use the following format to manually construct the link. Substitute your account ID or alias and the role name for the two parameters in the following example.

`https://signin.aws.amazon.com/switchrole?
account=your_account_ID_or_alias&roleName=optional_path/role_name`

We recommend that you direct your users to [Switching to a role \(console\) \(p. 253\)](#) to step them through the process.

Considerations

- If you create the role programmatically, you can create the role with a path in addition to a name. If you do so, you must provide the complete path and role name to your users so they can enter it on the **Switch Role** page of the AWS Management Console. For example: division_abc/subdivision_efg/role_XYZ.
- If you create the role programmatically, you can add a Path of up to 512 characters in addition to a RoleName. The role name can be up to 64 characters long. However, to use a role with the **Switch Role** feature in the AWS Management Console, the combined Path and RoleName cannot exceed 64 characters.
- For security purposes, you can [review AWS CloudTrail logs \(p. 340\)](#) to learn who performed an action in AWS. You can use the `aws:RoleSessionName` condition key in the role trust policy to require users to specify a session name when they assume a role. For example, you can require that IAM users specify their own user name as their session name. For more information, see [aws:RoleSessionName \(p. 718\)](#).

Granting a user permissions to pass a role to an AWS service

To configure many AWS services, you must *pass* an IAM role to the service. This allows the service to later assume the role and perform actions on your behalf. You only have to pass the role to the service once during set-up, and not every time that the service assumes the role. For example, assume that you have an application running on an Amazon EC2 instance. That application requires temporary credentials for authentication, and permissions to authorize the application to perform actions in AWS. When you set up the application, you must pass a role to EC2 to use with the instance that provides those credentials. You define the permissions for the applications running on the instance by attaching an IAM policy to the role. The application assumes the role every time it needs to perform the actions that are allowed by the role.

To pass a role (and its permissions) to an AWS service, a user must have permissions to *pass the role* to the service. This helps administrators ensure that only approved users can configure a service with a role that grants permissions. To allow a user to pass a role to an AWS service, you must grant the `PassRole` permission to the user's IAM user, role, or group.

Note

You cannot limit permissions to pass a role based on tags attached to that role using the `ResourceTag/key-name` condition key. For more information, see [Controlling access to AWS resources \(p. 388\)](#).

When you create a service-linked role, you must also have permission to pass that role to the service. Some services automatically create a service-linked role in your account when you perform an action in that service. For example, Amazon EC2 Auto Scaling creates the `AWSServiceRoleForAutoScaling` service-linked role for you the first time that you create an Auto Scaling group. If you try to create an Auto Scaling group without the `PassRole` permission, you receive an error. To learn which services support service-linked roles, see [AWS services that work with IAM \(p. 611\)](#). To learn which services automatically create a service-linked role when you perform an action in that service, choose the **Yes** link and view the service-linked role documentation for the service.

A user can pass a role ARN as a parameter in any API operation that uses the role to assign permissions to the service. The service then checks whether that user has the `iam:PassRole` permission. To limit the user to passing only approved roles, you can filter the `iam:PassRole` permission with the `Resources` element of the IAM policy statement.

You can use the `Condition` element in a JSON policy to test the value of keys that are included in the request context of all AWS requests. To learn more about using condition keys in a policy, see [IAM JSON policy elements: Condition \(p. 641\)](#). The `iam:PassedToService` condition key can be used to specify the service principal of the service to which a role can be passed. To learn more about using the `iam:PassedToService` condition key in a policy, see [iam:PassedToService \(p. 709\)](#).

Example 1

Imagine that you want to grant a user the ability to pass any of an approved set of roles to the Amazon EC2 service upon launching an instance. You need three elements:

- An IAM *permissions policy* attached to the role that determines what the role can do. Scope permissions to only the actions that the role must perform, and to only the resources that the role needs for those actions. You can use AWS managed or customer-created IAM permissions policy.

```
{  
    "Version": "2012-10-17",  
    "Statement": {  
        "Effect": "Allow",  
        "Action": [ "A list of the permissions the role is allowed to use" ],  
        "Resource": [ "A list of the resources the role is allowed to access" ]  
    }  
}
```

- A *trust policy* for the role that allows the service to assume the role. For example, you could attach the following trust policy to the role with the `UpdateAssumeRolePolicy` action. This trust policy allows Amazon EC2 to use the role and the permissions attached to the role.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "TrustPolicyStatementThatAllowsEC2ServiceToAssumeTheAttachedRole",
            "Effect": "Allow",
            "Principal": { "Service": "ec2.amazonaws.com" },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

- An IAM *permissions policy* attached to the IAM user that allows the user to pass only those roles that are approved. `iam:PassRole` usually is accompanied by `iam:GetRole` so that the user can get the details of the role to be passed. In this example, the user can pass only roles that exist in the specified account with names that begin with `EC2-roles-for-XYZ-`:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:GetRole",
                "iam:PassRole"
            ],
            "Resource": "arn:aws:iam::<account-id>:role/EC2-roles-for-XYZ-*"
        }
    ]
}
```

Now the user can start an Amazon EC2 instance with an assigned role. Applications running on the instance can access temporary credentials for the role through the instance profile metadata. The permissions policies attached to the role determine what the instance can do.

Example 2

Amazon Relational Database Service (Amazon RDS) supports a feature called Enhanced Monitoring. This feature enables Amazon RDS to monitor a database instance using an agent. It also allows Amazon RDS to log metrics to Amazon CloudWatch Logs. To enable this feature, you must create a service role to give Amazon RDS permissions to monitor and write metrics to your logs.

To create a role for Amazon RDS enhanced monitoring

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. Choose **Roles**, and then choose **Create role**.
3. Choose the **AWS Service** role type, and then choose the **Amazon RDS Role for Enhanced Monitoring** service. Then choose **Next: Permissions**.
4. Choose the **AmazonRDSEnhancedMonitoringRole** permissions policy.
5. Choose **Next: Tags**.
6. (Optional) Add metadata to the user by attaching tags as key-value pairs. For more information about using tags in IAM, see [Tagging IAM users and roles \(p. 289\)](#).
7. Choose **Next: Review**.
8. For **Role name**, type a role name that helps you identify the purpose of this role. Role names must be unique within your AWS account. They are not distinguished by case. For example, you cannot

create roles named both **PRODROLE** and **prodrole**. Because various entities might reference the role, you cannot edit the name of the role after it has been created.

9. (Optional) For **Role description**, type a description for the new role.
10. Review the role and then choose **Create role**.

The role automatically gets a trust policy that grants the `monitoring.rds.amazonaws.com` service permissions to assume the role. After it does, Amazon RDS can perform all of the actions that the `AmazonRDSEnhancedMonitoringRole` policy allows.

The user that you want to enable Enhanced Monitoring needs a policy that includes a statement that allows the user to pass the role, like the following. Use your account number and replace the role name with the name you provided in step 3:

```
{  
    "Sid": "PolicyStatementToAllowUserToPassOneSpecificRole",  
    "Effect": "Allow",  
    "Action": [ "iam:PassRole" ],  
    "Resource": "arn:aws:iam::<account-id>:role/RDS-Monitoring-Role"  
}
```

You can combine this statement with statements in another policy or put it in its own policy. To instead specify that the user can pass any role that begins with `RDS-`, you can replace the role name in the resource ARN with a wildcard, for example:

```
"Resource": "arn:aws:iam::<account-id>:role/RDS-*"
```

Switching to a role (console)

A *role* specifies a set of permissions that you can use to access AWS resources that you need. In that sense, it is similar to a [user in AWS Identity and Access Management \(IAM\)](#). When you sign in as a user, you get a specific set of permissions. However, you don't sign in to a role, but once signed in you can switch to a role. This temporarily sets aside your original user permissions and instead gives you the permissions assigned to the role. The role can be in your own account or any other AWS account. For more information about roles, their benefits, and how to create them, see [IAM roles \(p. 167\)](#), and [Creating IAM roles \(p. 221\)](#).

Important

The permissions of your IAM user and any roles that you switch to are not cumulative. Only one set of permissions is active at a time. When you switch to a role, you temporarily give up your user permissions and work with the permissions that are assigned to the role. When you exit the role, your user permissions are automatically restored.

When you switch roles in the AWS Management Console, the console always uses your original credentials to authorize the switch. This applies whether you sign in as an IAM user, as a SAML-federated role, or as a web-identity federated role. For example, if you switch to RoleA, IAM uses your original user or federated role credentials to determine whether you are allowed to assume RoleA. If you then switch to RoleB *while you are using RoleA*, AWS still uses your **original** user or federated role credentials to authorize the switch, not the credentials for RoleA.

Things to know about switching roles in the console

This section provides additional information about using the IAM console to switch to a role.

Note

You cannot switch roles if you sign in as the AWS account root user. You can switch roles when you sign in as an IAM user.

- If your administrator gives you a link, choose the link and then skip to step [Step 5](#) in the following procedure. The link takes you to the appropriate webpage and fills in the account ID (or alias) and the role name.
- You can manually construct the link and then skip to step [Step 5](#) in the following procedure. To construct your link, use the following format:

```
https://signin.aws.amazon.com/switchrole?  
account=account\_id\_number&roleName=role\_name&displayname=text\_to\_display
```

Where you replace the following text:

- account_id_number** – The 12-digit account identifier provided to you by your administrator. Alternatively, your administrator might create an account alias so that the URL includes your account name instead of an account ID. For more information, see [Your AWS account ID and its alias \(p. 67\)](#).
- role_name** – The name of the role that you want to assume. You can get this from the end of the role's ARN. For example, provide the TestRole role name from the following role ARN: namearn:aws:iam::403299380220:role/TestRole.
- (Optional) **text_to_display** – The text that you want to appear on the navigation bar in place of your user name when this role is active.
- You can manually switch roles using the information your administrator provides by using the procedures that follow.

By default, when you switch roles, your AWS Management Console session lasts for 1 hour. IAM user sessions are 12 hours by default. IAM users who switch roles in the console are granted the role maximum session duration, or the remaining time in the IAM user's session, whichever is less. For example, assume that a maximum session duration of 10 hours is set for a role. An IAM user has been signed in to the console for 8 hours when they decide to switch to the role. There are 4 hours remaining in the IAM user session, so the allowed role session duration is 4 hours. The following table shows how to determine the session duration for an IAM user when switching roles in the console.

IAM users console role session duration

IAM user session time remaining is...	Role session duration is...		
Less than role maximum session duration	Time remaining in IAM user session		
Greater than role maximum session duration	Maximum session duration value		
Equal to role maximum session duration	Maximum session duration value (approximate)		

Note

Some AWS service consoles can autorenew your role session when it expires without you taking any action. Some might prompt you to reload your browser page to reauthenticate your session.

To troubleshoot common issues that you might encounter when you assume a role, see [I can't assume a role \(p. 586\)](#).

To switch to a role (console)

1. Sign in to the AWS Management Console as an IAM user and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the IAM console, choose your user name on the navigation bar in the upper right. It typically looks like this: **username@account_ID_number_or_alias**.
3. Choose **Switch Role**. If this is the first time choosing this option, a page appears with more information. After reading it, choose **Switch Role**. If you clear your browser cookies, this page can appear again.
4. On the **Switch Role** page, type the account ID number or the account alias and the name of the role that was provided by your administrator.

Note

If your administrator created the role with a path, such as division_abc / subdivision_efg / roleToDoX, then you must type that complete path and name in the **Role** box. If you type only the role name, or if the combined Path and RoleName exceed 64 characters, the role switch fails. This is a limit of the browser cookies that store the role name. If this happens, contact your administrator and ask them to reduce the size of the path and role name.

5. (Optional) Choose a **Display name**. Type text that you want to appear on the navigation bar in place of your user name when this role is active. A name is suggested, based on the account and role information, but you can change it to whatever has meaning for you. You can also select a color to highlight the display name. The name and color can help remind you when this role is active, which changes your permissions. For example, for a role that gives you access to the test environment, you might specify a **Display name of Test** and select the green **Color**. For the role that gives you access to production, you might specify a **Display name of Production** and select red as the **Color**.
6. Choose **Switch Role**. The display name and color replace your user name on the navigation bar, and you can start using the permissions that the role grants you.

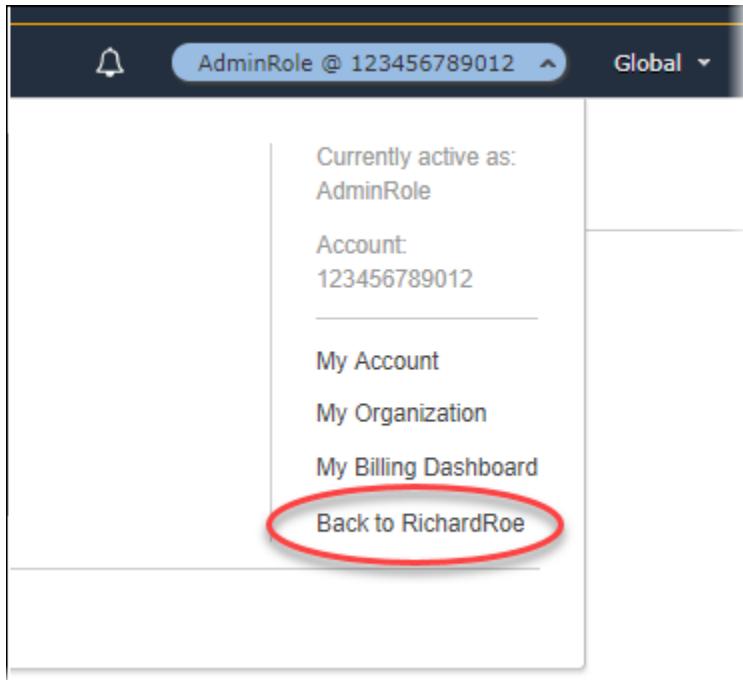
Tip

The last several roles that you used appear on the menu. The next time you need to switch to one of those roles, you can simply choose the role you want. You only need to type the account and role information manually if the role is not displayed on the menu.

To stop using a role (console)

1. In the IAM console, choose your role's **Display name** on the navigation bar in the upper right. It typically looks like this: **rolename@account_ID_number_or_alias**.
2. Choose **Back to username**. The role and its permissions are deactivated, and the permissions associated with your IAM user and groups are automatically restored.

For example, assume you are signed in to account number 123456789012 using the user name RichardRoe. After you use the AdminRole role, you want to stop using the role and return to your original permissions. To stop using a role, choose **AdminRole @ 123456789012**, and then choose **Back to RichardRoe**.



Switching to an IAM role (AWS CLI)

A *role* specifies a set of permissions that you can use to access AWS resources that you need. In that sense, it is similar to a [user in AWS Identity and Access Management \(IAM\)](#). When you sign in as a user, you get a specific set of permissions. However, you don't sign in to a role, but after signing in as a user, you can switch to a role. This temporarily sets aside your original user permissions and instead gives you the permissions assigned to the role. The role can be in your own account or any other AWS account. For more information about roles, their benefits, and how to create and configure them, see [IAM roles \(p. 167\)](#), and [Creating IAM roles \(p. 221\)](#). To learn about the different methods that you can use to assume a role, see [Using IAM roles \(p. 246\)](#).

Important

The permissions of your IAM user and any roles that you assume are not cumulative. Only one set of permissions is active at a time. When you assume a role, you temporarily give up your previous user or role permissions and work with the permissions that are assigned to the role. When you exit the role, your user permissions are automatically restored.

You can use a role to run an AWS CLI command when you are signed in as an IAM user. You can also use a role to run an AWS CLI command when you are signed in as an [externally authenticated user \(p. 176\)](#) ([SAML \(p. 182\)](#) or [OIDC \(p. 177\)](#)) that is already using a role. In addition, you can use a role to run an AWS CLI command from within an Amazon EC2 instance that is attached to a role through its instance profile. You can also use [role chaining \(p. 169\)](#), which is using a role to assume a second role. You cannot assume a role when you are signed in as the AWS account root user.

By default, your role session lasts for one hour. When you assume this role using the `assume-role*` CLI operations, you can specify a value for the `duration-seconds` parameter. This value can range from 900 seconds (15 minutes) up to the maximum session duration setting for the role. To learn how to view the maximum value for your role, see [View the maximum session duration setting for a role \(p. 248\)](#).

If you use role chaining, your session duration is limited to a maximum of one hour. If you then use the `duration-seconds` parameter to provide a value greater than one hour, the operation fails.

Example scenario: Switch to a production role

Imagine that you are an IAM user for working in the development environment. In this scenario, you occasionally need to work with the production environment at the command line with the [AWS CLI](#). You already have an access key credential set available to you. This can be the access key pair that is assigned to your standard IAM user. Or, if you signed in as a federated user, it can be the access key pair for the role that was initially assigned to you. If your current permissions grant you the ability to assume a specific IAM role, then you can identify that role in a "profile" in the AWS CLI configuration files. That command is then run with the permissions of the specified IAM role, not the original identity. Note that when you specify that profile in an AWS CLI command, you are using the new role. In this situation, you cannot make use of your original permissions in the development account at the same time. The reason is that only one set of permissions can be in effect at a time.

Note

For security purposes, administrators can [review AWS CloudTrail logs \(p. 340\)](#) to learn who performed an action in AWS. Your administrator might require that you specify your IAM user name as the session name when you assume the role. For more information, see [aws:RoleSessionName \(p. 718\)](#).

To switch to a production role (AWS CLI)

1. If you have never used the AWS CLI, then you must first configure your default CLI profile. Open a command prompt and set up your AWS CLI installation to use the access key from your IAM user or from your federated role. For more information, see [Configuring the AWS Command Line Interface](#) in the [AWS Command Line Interface User Guide](#).

Run the `aws configure` command as follows:

```
aws configure
```

When prompted, provide the following information:

```
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]: us-east-2
Default output format [None]: json
```

2. Create a new profile for the role in the `.aws/config` file in Unix or Linux, or the `C:\Users\USERNAME\.aws\config` file in Windows. The following example creates a profile called `prodaccess` that switches to the role `ProductionAccessRole` in the `123456789012` account. You get the role ARN from the account administrator who created the role. When this profile is invoked, the AWS CLI uses the credentials of the `source_profile` to request credentials for the role. Because of that, the identity referenced as the `source_profile` must have `sts:AssumeRole` permissions to the role that is specified in the `role_arn`.

```
[profile prodaccess]
  role_arn = arn:aws:iam::123456789012:role/ProductionAccessRole
  source_profile = default
```

3. After you create the new profile, any AWS CLI command that specifies the parameter `--profile prodaccess` runs under the permissions that are attached to the IAM role `ProductionAccessRole` instead of the default user.

```
aws iam list-users --profile prodaccess
```

This command works if the permissions assigned to the `ProductionAccessRole` enable listing the users in the current AWS account.

4. To return to the permissions granted by your original credentials, run commands without the `--profile` parameter. The AWS CLI reverts to using the credentials in your default profile, which you configured in [Step 1](#).

For more information, see [Assuming a Role in the AWS Command Line Interface User Guide](#).

Example scenario: Allow an instance profile role to switch to a role in another account

Imagine that you are using two AWS accounts, and you want to allow an application running on an Amazon EC2 instance to run [AWS CLI](#) commands in both accounts. Assume that the EC2 instance exists in account 111111111111. That instance includes the abcd instance profile role that allows the application to perform read-only Amazon S3 tasks on the my-bucket-1 bucket within the same 111111111111 account. However, the application must also be allowed to assume the efg cross-account role to perform tasks in account 222222222222. To do this, the abcd EC2 instance profile role must have the following permissions policy:

Account 111111111111 abcd Role Permissions Policy

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAccountLevelS3Actions",
            "Effect": "Allow",
            "Action": [
                "s3>ListAllMyBuckets",
                "s3>HeadBucket"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowListAndReadS3ActionOnMyBucket",
            "Effect": "Allow",
            "Action": [
                "s3:Get*",
                "s3>List*"
            ],
            "Resource": [
                "arn:aws:s3:::my-bucket-1/*",
                "arn:aws:s3:::my-bucket-1"
            ]
        },
        {
            "Sid": "AllowIPToAssumeCrossAccountRole",
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Resource": "arn:aws:iam:222222222222:role/efgh"
        }
    ]
}
```

Assume that the efg cross-account role allows read-only Amazon S3 tasks on the my-bucket-2 bucket within the same 222222222222 account. To do this, the efg cross-account role must have the following permissions policy:

Account 222222222222 efg Role Permissions Policy

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

```

"Statement": [
    {
        "Sid": "AllowAccountLevelS3Actions",
        "Effect": "Allow",
        "Action": [
            "s3>ListAllMyBuckets",
            "s3>HeadBucket"
        ],
        "Resource": "*"
    },
    {
        "Sid": "AllowListAndReadS3ActionOnMyBucket",
        "Effect": "Allow",
        "Action": [
            "s3:Get*",
            "s3>List*"
        ],
        "Resource": [
            "arn:aws:s3:::my-bucket-2/*",
            "arn:aws:s3:::my-bucket-2"
        ]
    }
]
}

```

The `efgh` role must allow the `abcd` instance profile role to assume it. To do this, the `efgh` role must have the following trust policy:

Account 222222222222 efgh Role Trust Policy

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "efghTrustPolicy",
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Principal": {"AWS": "arn:aws:iam::111111111111:role/abcd"}
        }
    ]
}

```

To then run AWS CLI commands in account 222222222222, you must update the CLI configuration file. Identify the `efgh` role as the "profile" and the `abcd` EC2 instance profile role as the "credential source" in the AWS CLI configuration file. Then your CLI commands are run with the permissions of the `efgh` role, not the original `abcd` role.

Note

For security purposes, you can use AWS CloudTrail to audit the use of roles in the account. To identify a role's actions in CloudTrail logs, you can use the role session name. When the AWS CLI assumes a role on a user's behalf as described in this topic, a role session name is automatically created as `AWS-CLI-session-nnnnnnnn`. Here `nnnnnnnn` is an integer that represents the time in [Unix epoch time](#) (the number of seconds since midnight UTC on January 1, 1970). For more information, see [CloudTrail Event Reference](#) in the [AWS CloudTrail User Guide](#).

To allow an EC2 instance profile role to switch to a cross-account role (AWS CLI)

1. You do not have to configure a default CLI profile. Instead, you can load credentials from the EC2 instance profile metadata. Create a new profile for the role in the `.aws/config` file. The following example creates an `instancecrossaccount` profile that switches to the role `efgh` in the 222222222222 account. When this profile is invoked, the AWS CLI uses the credentials of the

EC2 instance profile metadata to request credentials for the role. Because of that, the EC2 instance profile role must have `sts:AssumeRole` permissions to the role specified in the `role_arn`.

```
[profile instancecrossaccount]
role_arn = arn:aws:iam::222222222222:role/efgh
credential_source = Ec2InstanceMetadata
```

2. After you create the new profile, any AWS CLI command that specifies the parameter `--profile instancecrossaccount` runs under the permissions that are attached to the `efgh` role in account `222222222222`.

```
aws s3 ls my-bucket-2 --profile instancecrossaccount
```

This command works if the permissions that are assigned to the `efgh` role allow listing the users in the current AWS account.

3. To return to the original EC2 instance profile permissions in account `111111111111`, run the CLI commands without the `--profile` parameter.

For more information, see [Assuming a Role in the AWS Command Line Interface User Guide](#).

Switching to an IAM role (Tools for Windows PowerShell)

A *role* specifies a set of permissions that you can use to access AWS resources that you need. In that sense, it is similar to a [user in AWS Identity and Access Management \(IAM\)](#). When you sign in as a user, you get a specific set of permissions. However, you don't sign in to a role, but once signed in you can switch to a role. This temporarily sets aside your original user permissions and instead gives you the permissions assigned to the role. The role can be in your own account or any other AWS account. For more information about roles, their benefits, and how to create and configure them, see [IAM roles \(p. 167\)](#), and [Creating IAM roles \(p. 221\)](#).

Important

The permissions of your IAM user and any roles that you switch to are not cumulative. Only one set of permissions is active at a time. When you switch to a role, you temporarily give up your user permissions and work with the permissions that are assigned to the role. When you exit the role, your user permissions are automatically restored.

This section describes how to switch roles when you work at the command line with the AWS Tools for Windows PowerShell.

Imagine that you have an account in the development environment and you occasionally need to work with the production environment at the command line using the [Tools for Windows PowerShell](#). You already have one access key credential set available to you. These can be an access key pair assigned to your standard IAM user. Or, if you signed-in as a federated user, they can be the access key pair for the role initially assigned to you. You can use these credentials to run the `Use-STSSRole` cmdlet that passes the ARN of a new role as a parameter. The command returns temporary security credentials for the requested role. You can then use those credentials in subsequent PowerShell commands with the role's permissions to access resources in production. While you use the role, you cannot use your user permissions in the Development account because only one set of permissions are in effect at a time.

Note

For security purposes, administrators can [review AWS CloudTrail logs \(p. 340\)](#) to learn who performed an action in AWS. Your administrator might require that you specify your IAM user name as the session name when you assume the role. For more information, see [aws:RoleSessionName \(p. 718\)](#).

Note that all access keys and tokens are examples only and cannot be used as shown. Replace with the appropriate values from your live environment.

To switch to a role (Tools for Windows PowerShell)

1. Open a PowerShell command prompt and configure the default profile to use the access key from your current IAM user or from your federated role. If you have previously used the Tools for Windows PowerShell, then this is likely already done. Note that you can switch roles only if you are signed in as an IAM user, not the AWS account root user.

```
PS C:\> Set-AWSCredentials -AccessKey AKIAIOSFODNN7EXAMPLE -SecretKey wJalrXUtnFEMI/K7MDENG/bPxRficyEXAMPLEKEY -StoreAs MyMainUserProfile
PS C:\> Initialize-AWSDefaults -ProfileName MyMainUserProfile -Region us-east-2
```

For more information, see [Using AWS Credentials](#) in the *AWS Tools for Windows PowerShell User Guide*.

2. To retrieve credentials for the new role, run the following command to switch to the `RoleName` role in the 123456789012 account. You get the role ARN from the account administrator who created the role. The command requires that you provide a session name as well. You can choose any text for that. The following command requests the credentials and then captures the `Credentials` property object from the returned results object and stores it in the `$Creds` variable.

```
PS C:\> $Creds = (Use-STSSRole -RoleArn "arn:aws:iam::123456789012:role/RoleName" -RoleSessionName "MyRoleSessionName").Credentials
```

`$Creds` is an object that now contains the `AccessKeyId`, `SecretAccessKey`, and `SessionToken` elements that you need in the following steps. The following sample commands illustrate typical values:

```
PS C:\> $Creds.AccessKeyId
AKIAIOSFODNN7EXAMPLE

PS C:\> $Creds.SecretAccessKey
wJalrXUtnFEMI/K7MDENG/bPxRficyEXAMPLEKEY

PS C:\> $Creds.SessionToken
AQoDYXdzEGcaEXAMPLE2gsYULo+Im5ZEXAMPLEeYjs1M2FUIgIJx9tQqNMBEXAMPLEcvSRyh0FW7jEXAMPLEw
+vE/7s1HRp
XviG7b+qYf4nD00EXAMPLEmj4wxS04L/uZEXAMPLEcihzFB5lTYLto9dyBgSDyEXAMPLE9/
g7QRUhZp4bqbEXAMPLENwGPY
Oj59pFA41NKCikVgkREXAMPLEjlzxQ7y52gekeVEXAMPLEDiB9ST3UuysgsKdEXAMPLE1TVastU1AOSKFEXAMPLEiywCC/
C
s8EXAMPLEpZgOs+6hz4AP4KEXAMPLERbASP+4eZScEXAMPLEsnf87eNhyDHq6ikBQ==

PS C:\> $Creds.Expiration
Thursday, June 18, 2018 2:28:31 PM
```

3. To use these credentials for any subsequent command, include them with the `-Credentials` parameter. For example, the following command uses the credentials from the role and works only if the role is granted the `iam>ListRoles` permission and can therefore run the `Get-IAMRoles` cmdlet:

```
PS C:\> Get-IAMRoles -Credential $Creds
```

4. To return to your original credentials, simply stop using the `-Credentials` `$Creds` parameter and allow PowerShell to revert to the credentials that are stored in the default profile.

Switching to an IAM role (AWS API)

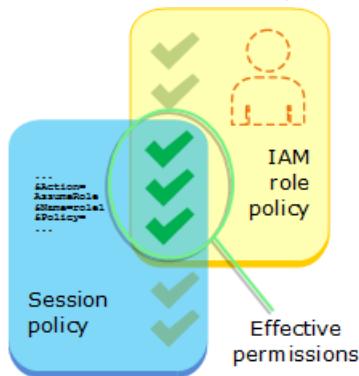
A *role* specifies a set of permissions that you can use to access AWS resources. In that sense, it is similar to an [IAM user](#). A principal (person or application) assumes a role to receive temporary permissions to carry out required tasks and interact with AWS resources. The role can be in your own account or any other AWS account. For more information about roles, their benefits, and how to create and configure them, see [IAM roles \(p. 167\)](#), and [Creating IAM roles \(p. 221\)](#). To learn about the different methods that you can use to assume a role, see [Using IAM roles \(p. 246\)](#).

Important

The permissions of your IAM user and any roles that you assume are not cumulative. Only one set of permissions is active at a time. When you assume a role, you temporarily give up your previous user or role permissions and work with the permissions that are assigned to the role. When you exit the role, your original permissions are automatically restored.

To assume a role, an application calls the AWS STS [AssumeRole](#) API operation and passes the ARN of the role to use. The operation creates a new session with temporary credentials. This session has the same permissions as the identity-based policies for that role.

When you call [AssumeRole](#), you can optionally pass inline or managed [session policies \(p. 353\)](#). Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary credential session for a role or federated user. You can pass a single JSON inline session policy document using the `Policy` parameter. You can use the `PolicyArns` parameter to specify up to 10 managed session policies. The resulting session's permissions are the intersection of the entity's identity-based policies and the session policies. Session policies are useful when you need to give the role's temporary credentials to someone else. They can use the role's temporary credentials in subsequent AWS API calls to access resources in the account that owns the role. You cannot use session policies to grant more permissions than those allowed by the identity-based policy. To learn more about how AWS determines the effective permissions of a role, see [Policy evaluation logic \(p. 666\)](#).



You can call [AssumeRole](#) when you are signed in as an IAM user, or as an [externally authenticated user \(p. 176\)](#) ([SAML \(p. 182\)](#) or [OIDC \(p. 177\)](#)) already using a role. You can also use [role chaining \(p. 169\)](#), which is using a role to assume a second role. You cannot assume a role when you are signed in as the AWS account root user.

By default, your role session lasts for one hour. When you assume this role using the AWS STS [AssumeRole*](#) API operations, you can specify a value for the `DurationSeconds` parameter. This value can range from 900 seconds (15 minutes) up to the maximum session duration setting for the role. To learn how to view the maximum value for your role, see [View the maximum session duration setting for a role \(p. 248\)](#).

If you use role chaining, your session is limited to a maximum of one hour. If you then use the `DurationSeconds` parameter to provide a value greater than one hour, the operation fails.

Note

For security purposes, administrators can [review AWS CloudTrail logs \(p. 340\)](#) to learn who performed an action in AWS. Your administrator might require that you specify your

IAM user name as the session name when you assume the role. For more information, see [aws:RoleSessionName \(p. 718\)](#).

The following example in Python using the Boto3 interface to AWS ([AWS SDK for Python \(Boto\) V3](#)) shows how to call AssumeRole. It also shows how to use the temporary security credentials returned by AssumeRole to list all Amazon S3 buckets in the account that owns the role.

```
import boto3

# The calls to AWS STS AssumeRole must be signed with the access key ID
# and secret access key of an existing IAM user or by using existing temporary
# credentials such as those from another role. (You cannot call AssumeRole
# with the access key for the root account.) The credentials can be in
# environment variables or in a configuration file and will be discovered
# automatically by the boto3.client() function. For more information, see the
# Python SDK documentation:
# http://boto3.readthedocs.io/en/latest/reference/services/sts.html#client

# Create an STS client object that represents a live connection to the
# STS service
sts_client = boto3.client('sts')

# Call the assume_role method of the STSConnection object and pass the role
# ARN and a role session name.
assumed_role_object=sts_client.assume_role(
    RoleArn="arn:aws:iam::account-of-role-to-assume:role/name-of-role",
    RoleSessionName="AssumeRoleSession1"
)

# From the response that contains the assumed role, get the temporary
# credentials that can be used to make subsequent API calls
credentials=assumed_role_object['Credentials']

# Use the temporary credentials that AssumeRole returns to make a
# connection to Amazon S3
s3_resource=boto3.resource(
    's3',
    aws_access_key_id=credentials['AccessKeyId'],
    aws_secret_access_key=credentials['SecretAccessKey'],
    aws_session_token=credentials['SessionToken'],
)

# Use the Amazon S3 resource object that is now configured with the
# credentials to access your S3 buckets.
for bucket in s3_resource.buckets.all():
    print(bucket.name)
```

Using an IAM role to grant permissions to applications running on Amazon EC2 instances

Applications that run on an EC2 instance must include AWS credentials in their AWS API requests. You could have your developers store AWS credentials directly within the EC2 instance and allow applications in that instance to use those credentials. But developers would then have to manage the credentials and ensure that they securely pass the credentials to each instance and update each EC2 instance when it's time to rotate the credentials. That's a lot of additional work.

Instead, you can and should use an IAM role to manage *temporary* credentials for applications that run on an EC2 instance. When you use a role, you don't have to distribute long-term credentials (such as a user name and password or access keys) to an EC2 instance. Instead, the role supplies temporary permissions that applications can use when they make calls to other AWS resources. When you launch

an EC2 instance, you specify an IAM role to associate with the instance. Applications that run on the instance can then use the role-supplied temporary credentials to sign API requests.

Using roles to grant permissions to applications that run on EC2 instances requires a bit of extra configuration. An application running on an EC2 instance is abstracted from AWS by the virtualized operating system. Because of this extra separation, an additional step is needed to assign an AWS role and its associated permissions to an EC2 instance and make them available to its applications. This extra step is the creation of an *instance profile* that is attached to the instance. The instance profile contains the role and can provide the role's temporary credentials to an application that runs on the instance. Those temporary credentials can then be used in the application's API calls to access resources and to limit access to only those resources that the role specifies. Note that only one role can be assigned to an EC2 instance at a time, and all applications on the instance share the same role and permissions.

Using roles in this way has several benefits. Because role credentials are temporary and rotated automatically, you don't have to manage credentials, and you don't have to worry about long-term security risks. In addition, if you use a single role for multiple instances, you can make a change to that one role and the change is propagated automatically to all the instances.

Note

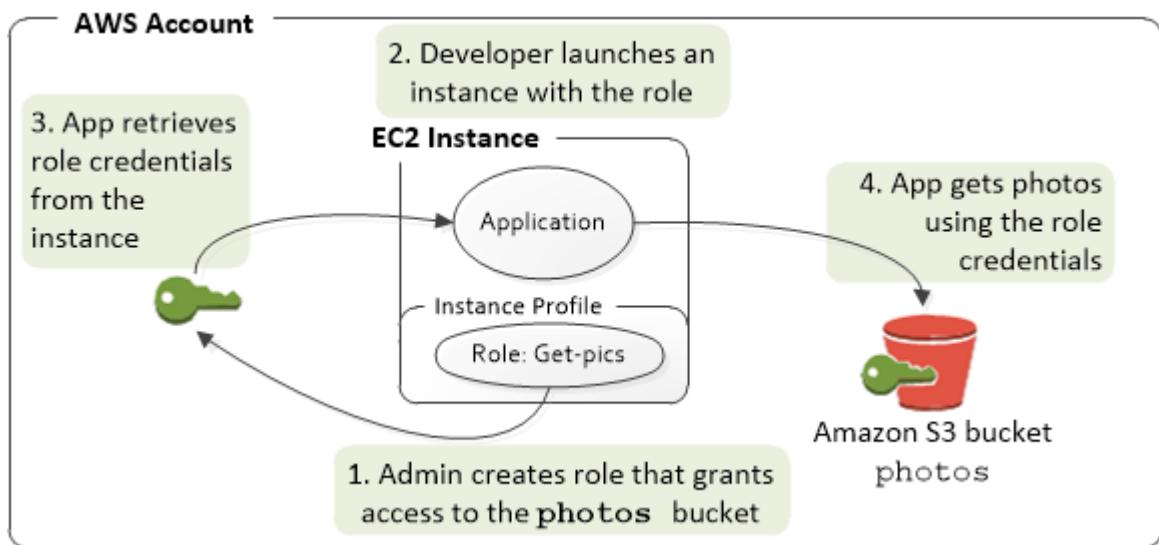
Although a role is usually assigned to an EC2 instance when you launch it, a role can also be attached to an EC2 instance that is already running. To learn how to attach a role to a running instance, see [IAM Roles for Amazon EC2](#).

Topics

- [How do roles for EC2 instances work? \(p. 264\)](#)
- [Permissions required for using roles with Amazon EC2 \(p. 266\)](#)
- [How do I get started? \(p. 269\)](#)
- [Related information \(p. 269\)](#)
- [Using instance profiles \(p. 270\)](#)

How do roles for EC2 instances work?

In the following figure, a developer runs an application on an EC2 instance that requires access to the S3 bucket named `photos`. An administrator creates the `Get-pics` service role and attaches the role to the EC2 instance. The role includes a permissions policy that grants read-only access to the specified S3 bucket. It also includes a trust policy that allows the EC2 instance to assume the role and retrieve the temporary credentials. When the application runs on the instance, it can use the role's temporary credentials to access the `photos` bucket. The administrator doesn't have to grant the developer permission to access the `photos` bucket, and the developer never has to share or manage credentials.



1. The administrator uses IAM to create the **Get-pics** role. In the role's trust policy, the administrator specifies that only EC2 instances can assume the role. In the role's permission policy, the administrator specifies read-only permissions for the photos bucket.
2. A developer launches an EC2 instance and assigns the **Get-pics** role to that instance.

Note

If you use the IAM console, the instance profile is managed for you and is mostly transparent to you. However, if you use the AWS CLI or API to create and manage the role and EC2 instance, then you must create the instance profile and assign the role to it as separate steps. Then, when you launch the instance, you must specify the instance profile name instead of the role name.

3. When the application runs, it obtains temporary security credentials from Amazon EC2 [instance metadata](#), as described in [Retrieving Security Credentials from Instance Metadata](#). These are [temporary security credentials](#) (p. 301) that represent the role and are valid for a limited period of time.

With some [AWS SDKs](#), the developer can use a provider that manages the temporary security credentials transparently. (The documentation for individual AWS SDKs describes the features supported by that SDK for managing credentials.)

Alternatively, the application can get the temporary credentials directly from the instance metadata of the EC2 instance. Credentials and related values are available from the `iam/security-credentials/role-name` category (in this case, `iam/security-credentials/Get-pics`) of the metadata. If the application gets the credentials from the instance metadata, it can cache the credentials.

4. Using the retrieved temporary credentials, the application accesses the photo bucket. Because of the policy attached to the **Get-pics** role, the application has read-only permissions.

The temporary security credentials that are available on the instance are automatically rotated before they expire so that a valid set is always available. The application just needs to make sure that it gets a new set of credentials from the instance metadata before the current ones expire. If the AWS SDK manages credentials, the application doesn't need to include additional logic to refresh the credentials. However, if the application gets temporary security credentials from the instance metadata and has cached them, it should get a refreshed set of credentials every hour, or at least 15 minutes before the current set expires. The expiration time is included in the information that is returned in the `iam/security-credentials/role-name` category.

Permissions required for using roles with Amazon EC2

To launch an instance with a role, the developer must have permission to launch 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 includes wildcards (*) to allow a user to pass any role and to perform all Amazon EC2 actions. The `ListInstanceProfiles` action allows users to view all of the roles that are available in the AWS account.

Example Example policy that grants a user permission to use the Amazon EC2 console to launch an instance with any role

```
{  
    "Version": "2012-10-17",  
    "Statement": [{  
        "Effect": "Allow",  
        "Action": [  
            "iam:PassRole",  
            "iam>ListInstanceProfiles",  
            "ec2:*"  
        ],  
        "Resource": "*"  
    }]  
}
```

Restricting which roles can be passed to EC2 instances (using PassRole)

You can use the `PassRole` permission to restrict which role a user can pass to an EC2 instance when the user launches the instance. This helps prevent the user from running applications that have more permissions than the user has been granted—that is, from being able to obtain elevated privileges. For example, imagine that user Alice has permissions only to launch EC2 instances and to work with Amazon S3 buckets, but the role she passes to an EC2 instance has permissions to work with IAM and Amazon DynamoDB. In that case, Alice might be able to launch the instance, log into it, get temporary security credentials, and then perform IAM or DynamoDB actions that she's not authorized for.

To restrict which roles a user can pass to an EC2 instance, you create a policy that allows the `PassRole` action. You then attach the policy to the user (or to an IAM group that the user belongs to) who will launch EC2 instances. In the `Resource` element of the policy, you list the role or roles that the user is allowed to pass to EC2 instances. When the user launches an instance and associates a role with it, Amazon EC2 checks whether the user is allowed to pass that role. Of course, you should also ensure that the role that the user can pass does not include more permissions than the user is supposed to have.

Note

`PassRole` is not an API action in the same way that `RunInstances` or `ListInstanceProfiles` is. Instead, it's a permission that AWS checks whenever a role ARN is passed as a parameter to an API (or the console does this on the user's behalf). It helps an administrator to control which roles can be passed by which users. In this case, it ensures that the user is allowed to attach a specific role to an Amazon EC2 instance.

Example Example policy that grants a user permission to launch an EC2 instance with a specific role

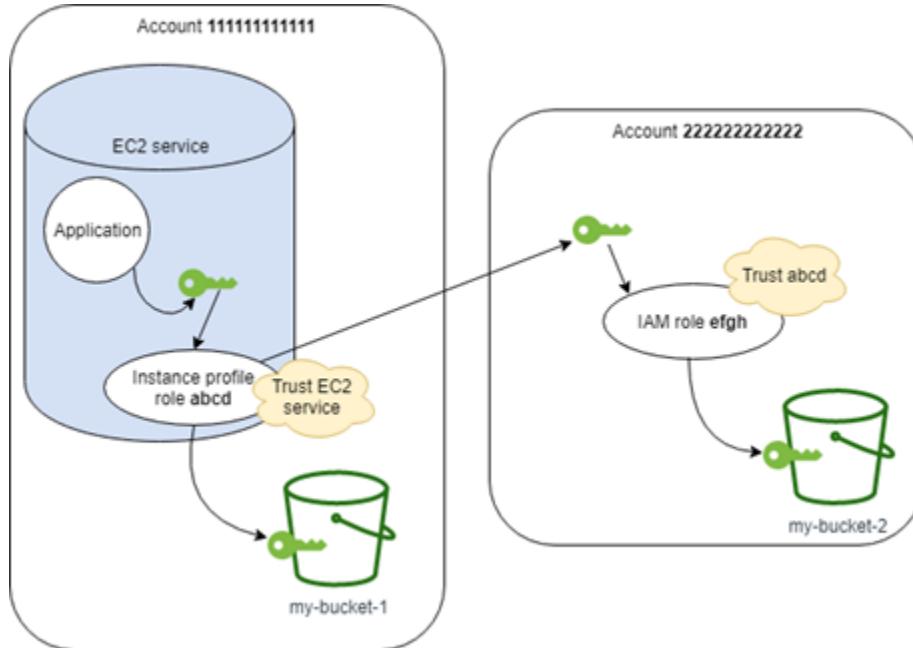
The following sample policy allows users to use the Amazon EC2 API to launch an instance with a role. The `Resource` element specifies the Amazon Resource Name (ARN) of a role. By specifying the ARN, the policy grants the user the permission to pass only the `Get-pics` role. If the user tries to specify a different role when launching an instance, the action fails. The user does have permissions to run any instance, regardless of whether they pass a role.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "ec2:RunInstances",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:role/Get-pics"
        }
    ]
}
```

Allowing an instance profile role to switch to a role in another account

You can allow an application running on an Amazon EC2 instance to run commands in another account. To do this, you must allow the EC2 instance role in the first account to switch to a role in the second account.

Imagine that you are using two AWS accounts and you want to allow an application running on an Amazon EC2 instance to run [AWS CLI](#) commands in both accounts. Assume that the EC2 instance exists in account 111111111111. That instance includes the abcd instance profile role that allows the application to perform read-only Amazon S3 tasks on the my-bucket-1 bucket within the same 111111111111 account. However, the application must also be allowed to assume the efg cross-account role to access the my-bucket-2 Amazon S3 bucket in account 222222222222.



The abcd EC2 instance profile role must have the following permissions policy to allow the application to access the my-bucket-1 Amazon S3 bucket:

Account 111111111111 abcd Role Permissions Policy

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

```

"Statement": [
    {
        "Sid": "AllowAccountLevelS3Actions",
        "Effect": "Allow",
        "Action": [
            "s3>ListAllMyBuckets",
            "s3>HeadBucket"
        ],
        "Resource": "*"
    },
    {
        "Sid": "AllowListAndReadS3ActionOnMyBucket",
        "Effect": "Allow",
        "Action": [
            "s3:Get*",
            "s3>List*"
        ],
        "Resource": [
            "arn:aws:s3:::my-bucket-1/*",
            "arn:aws:s3:::my-bucket-1"
        ]
    },
    {
        "Sid": "AllowIPToAssumeCrossAccountRole",
        "Effect": "Allow",
        "Action": "sts:AssumeRole",
        "Resource": "arn:aws:iam:222222222222:role/efgh"
    }
]
}

```

The abcd role must trust the Amazon EC2 service to assume the role. To do this, the abcd role must have the following trust policy:

Account 111111111111 abcd Role Trust Policy

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "abcdTrustPolicy",
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Principal": {"Service": "ec2.amazonaws.com"}
        }
    ]
}

```

Assume that the efgh cross-account role allows read-only Amazon S3 tasks on the my-bucket-2 bucket within the same 22222222222 account. To do this, the efgh cross-account role must have the following permissions policy:

Account 222222222222 efgh Role Permissions Policy

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAccountLevelS3Actions",
            "Effect": "Allow",
            "Action": [
                "s3>ListAllMyBuckets",
                "s3>HeadBucket"
            ],
            "Resource": "*"
        }
    ]
}

```

```
        ],
        "Resource": "*"
    },
{
    "Sid": "AllowListAndReadS3ActionOnMyBucket",
    "Effect": "Allow",
    "Action": [
        "s3:Get*",
        "s3>List*"
    ],
    "Resource": [
        "arn:aws:s3:::my-bucket-2/*",
        "arn:aws:s3:::my-bucket-2"
    ]
}
]
```

The `efgh` role must trust the `abcd` instance profile role to assume it. To do this, the `efgh` role must have the following trust policy:

Account 222222222222 efgh Role Trust Policy

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "efghTrustPolicy",
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Principal": {"AWS": "arn:aws:iam::111111111111:role/abcd"}
        }
    ]
}
```

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. You can examine the [instance metadata](#) to see how the role's temporary credentials are made available to an instance. You can also see how an application that runs on an instance can use the role. Use the following resources to learn more.

- SDK walkthroughs. The AWS SDK documentation includes walkthroughs that show an application running on an EC2 instance that uses temporary credentials for roles to read an Amazon S3 bucket. Each of the following walkthroughs presents similar steps with a different programming language:
 - [Configure IAM Roles for Amazon EC2 with the SDK for Java](#) in the *AWS SDK for Java Developer Guide*
 - [Launch an EC2 Instance using the SDK for .NET](#) in the *AWS SDK for .NET Developer Guide*
 - [Creating an Amazon EC2 Instance with the SDK for Ruby](#) in the *AWS SDK for Ruby Developer Guide*

Related information

For more information about creating roles or roles for EC2 instances, see the following information:

- For more information about [using IAM roles with Amazon EC2 instances](#), go to the *Amazon EC2 User Guide for Linux Instances*.
- To create a role, see [Creating IAM roles](#) (p. 221)

- For more information about using temporary security credentials, see [Temporary security credentials in IAM \(p. 301\)](#).
- If you work with the IAM API or CLI, you must create and manage IAM instance profiles. For more information about instance profiles, see [Using instance profiles \(p. 270\)](#).
- For more information about temporary security credentials for roles in the instance metadata, see [Retrieving Security Credentials from Instance Metadata](#) in the *Amazon EC2 User Guide for Linux Instances*.

Using instance profiles

Use an instance profile to pass an IAM role to an EC2 instance. For more information, see [IAM roles for Amazon EC2](#) in the *Amazon EC2 User Guide for Linux Instances*.

Managing instance profiles (console)

If you use the AWS Management Console to create a role for Amazon EC2, the console automatically creates an instance profile and gives it the same name as the role. When you then use the Amazon EC2 console to launch an instance with an IAM role, you can select a role to associate with the instance. In the console, the list that's displayed is actually a list of instance profile names. The console does not create an instance profile for a role that is not associated with Amazon EC2.

Managing instance profiles (AWS CLI or AWS API)

If you manage your roles from the AWS CLI or the AWS API, you create roles and instance profiles as separate actions. Because roles and instance profiles can have different names, you must know the names of your instance profiles as well as the names of roles they contain. That way you can choose the correct instance profile when you launch an EC2 instance.

Note

An instance profile can contain only one IAM role, although a role can be included in multiple instance profiles. This limit of one role per instance profile cannot be increased. You can remove the existing role and then add a different role to an instance profile. You must then wait for the change to appear across all of AWS because of [eventual consistency](#). To force the change, you must [disassociate the instance profile](#) and then [associate the instance profile](#), or you can stop your instance and then restart it.

Managing instance profiles (AWS CLI)

You can use the following AWS CLI commands to work with instance profiles in an AWS account.

- Create an instance profile: `aws iam create-instance-profile`
- Add a role to an instance profile: `aws iam add-role-to-instance-profile`
- List instance profiles: `aws iam list-instance-profiles`, `aws iam list-instance-profiles-for-role`
- Get information about an instance profile: `aws iam get-instance-profile`
- Remove a role from an instance profile: `aws iam remove-role-from-instance-profile`
- Delete an instance profile: `aws iam delete-instance-profile`

You can also attach a role to an already running EC2 instance by using the following commands. For more information, see [IAM Roles for Amazon EC2](#).

- Attach an instance profile with a role to a stopped or running EC2 instance: `aws ec2 associate-iam-instance-profile`
- Get information about an instance profile attached to an EC2 instance: `aws ec2 describe-iam-instance-profile-associations`

- Detach an instance profile with a role from a stopped or running EC2 instance: [aws ec2 disassociate-iam-instance-profile](#)

Managing instance profiles (AWS API)

You can call the following AWS API operations to work with instance profiles in an AWS account.

- Create an instance profile: [CreateInstanceProfile](#)
- Add a role to an instance profile: [AddRoleToInstanceProfile](#)
- List instance profiles: [ListInstanceProfiles](#), [ListInstanceProfilesForRole](#)
- Get information about an instance profile: [GetInstanceProfile](#)
- Remove a role from an instance profile: [RemoveRoleFromInstanceProfile](#)
- Delete an instance profile: [DeleteInstanceProfile](#)

You can also attach a role to an already running EC2 instance by calling the following operations. For more information, see [IAM Roles for Amazon EC2](#).

- Attach an instance profile with a role to a stopped or running EC2 instance: [AssociateIamInstanceProfile](#)
- Get information about an instance profile attached to an EC2 instance: [DescribeIamInstanceProfileAssociations](#)
- Detach an instance profile with a role from a stopped or running EC2 instance: [DisassociateIamInstanceProfile](#)

Revoking IAM role temporary security credentials

Warning

If you follow the steps on this page, all users with current sessions created by assuming the role are denied access to all AWS actions and resources. This can result in users losing unsaved work.

When you enable users to access the AWS Management Console with a long session duration time (such as 12 hours), their temporary credentials do not expire as quickly. If users inadvertently expose their credentials to an unauthorized third party, that party has access for the duration of the session. However, you can immediately revoke all permissions to the role's credentials issued before a certain point in time if you need to. All temporary credentials for that role issued before the specified time become invalid. This forces all users to reauthenticate and request new credentials.

Note

You cannot revoke the session for a [service-linked role \(p. 168\)](#).

When you revoke permissions for a role using the procedure in this topic, AWS attaches a new inline policy to the role that denies all permissions to all actions. It includes a condition that applies the restrictions only if the user assumed the role *before* the point in time when you revoke the permissions. If the user assumes the role *after* you revoked the permissions, then the deny policy does not apply to that user.

Important

This deny policy applies to all users of the specified role, not just those with longer duration console sessions.

Minimum permissions to revoke session permissions from a role

To successfully revoke session permissions from a role, you must have the `PutRolePolicy` permission for the role. This allows you to attach the `AWSRevokeOlderSessions` inline policy to the role.

Revoking session permissions

You can revoke the session permissions from a role.

To immediately deny all permissions to any current user of role credentials

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 **IAM Dashboard**, choose **Roles**, and then choose the name (not the check box) of the role whose permissions you want to revoke.
3. On the **Summary** page for the selected role, choose the **Revoke sessions** tab.
4. On the **Revoke sessions** tab, choose **Revoke active sessions**.
5. AWS asks you to confirm the action. Choose **Revoke active sessions** on the dialog box.

IAM immediately attaches a policy named `AWSRevokeOlderSessions` to the role. The policy denies all access to users who assumed the role before the moment you chose **Revoke active sessions**. Any user who assumes the role *after* you chose **Revoke active sessions** is not affected.

When you apply a new policy to a user or a resource, it may take a few minutes for policy updates to take effect. To learn why changes are not always immediately visible, see [Changes that I make are not always immediately visible \(p. 568\)](#).

Note

Don't worry about remembering to delete the policy. Any user who assumes the role *after* you revoked sessions is not affected by the policy. If you choose to **Revoke Sessions** again later, then the date/time stamp in the policy is refreshed and it again denies all permissions to any user who assumed the role before the new specified time.

Valid users whose sessions are revoked in this way must acquire temporary credentials for a new session to continue working. Note that the AWS CLI caches credentials until they expire. To force the CLI to delete and refresh cached credentials that are no longer valid, run one of the following commands:

Linux, macOS, or Unix

```
$ rm -r ~/.aws/cli/cache
```

Windows

```
C:\> del /s /q %UserProfile%\.aws\cli\cache
```

For more information, see [Disabling permissions for temporary security credentials \(p. 323\)](#).

Managing IAM roles

Occasionally you need to modify or delete the roles that you have created. To change a role, you can do any of the following:

- Modify the policies that are associated with the role
- Change who can access the role
- Edit the permissions that the role grants to users
- Change the maximum session duration setting for roles that are assumed using the AWS Management Console, AWS CLI or API

You can also delete roles that are no longer needed. You can manage your roles from the AWS Management Console, the AWS CLI, and the API.

Topics

- [Modifying a role \(p. 273\)](#)
- [Deleting roles or instance profiles \(p. 283\)](#)

Modifying a role

You can use the AWS Management Console, the AWS CLI, or the IAM API to make changes to a role.

Topics

- [View role access \(p. 273\)](#)
- [Modifying a role \(console\) \(p. 273\)](#)
- [Modifying a role \(AWS CLI\) \(p. 276\)](#)
- [Modifying a role \(AWS API\) \(p. 280\)](#)

View role access

Before you change the permissions for a role, you should review its recent service-level activity. This is important because you don't want to remove access from a principal (person or application) who is using it. For more information about viewing last accessed information, see [Refining permissions in AWS using last accessed information \(p. 472\)](#).

Modifying a role (console)

You can use the AWS Management Console to modify a role. To change the set of tags on a role, see [Managing tags on IAM entities \(console\) \(p. 292\)](#).

Topics

- [Modifying a role trust policy \(console\) \(p. 273\)](#)
- [Modifying a role permissions policy \(console\) \(p. 275\)](#)
- [Modifying a role description \(console\) \(p. 275\)](#)
- [Modifying a role maximum session duration \(console\) \(p. 275\)](#)
- [Modifying a role permissions boundary \(console\) \(p. 276\)](#)

Modifying a role trust policy (console)

To change who can assume a role, you must modify the role's trust policy. You cannot modify the trust policy for a [service-linked role \(p. 168\)](#).

Note

If a user is listed as the principal in a role's trust policy but cannot assume the role, check the user's [permissions boundary \(p. 365\)](#). If a permissions boundary is set for the user, then it must allow the `sts:AssumeRole` action.

To modify a role trust policy (console)

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 IAM console, choose **Roles**.
3. In the list of roles in your account, choose the name of the role that you want to modify.

4. Choose the **Trust relationships** tab, and then choose **Edit trust relationship**.
5. Edit the trust policy as needed. To add additional principals that can assume the role, specify them in the **Principal** element. For example, the following policy snippet shows how to reference two AWS accounts in the **Principal** element:

```
"Principal": {
    "AWS": [
        "arn:aws:iam::111122223333:root",
        "arn:aws:iam::444455556666:root"
    ]
},
```

If you specify a principal in another account, adding an account to the trust policy of a role is only half of establishing the cross-account trust relationship. By default, no users in the trusted accounts can assume the role. The administrator for the newly trusted account must grant the users the permission to assume the role. To do that, the administrator must create or edit a policy that is attached to the user to allow the user access to the `sts:AssumeRole` action. For more information, see the following procedure or [Granting a user permissions to switch roles \(p. 248\)](#).

The following policy snippet shows how to reference two AWS services in the **Principal** element:

```
"Principal": {
    "Service": [
        "opsworks.amazonaws.com",
        "ec2.amazonaws.com"
    ]
},
```

6. When you are finished editing your trust policy, choose **Update Trust Policy** to save your changes.

For more information about policy structure and syntax, see [Policies and permissions in IAM \(p. 351\)](#) and the [IAM JSON policy elements reference \(p. 628\)](#).

To allow users in a trusted external account to use the role (console)

For more information and detail about this procedure, see [Granting a user permissions to switch roles \(p. 248\)](#).

1. Sign in to the trusted external AWS account.
2. Decide whether to attach the permissions to a user or to a group. In the navigation pane of the IAM console, choose **Users** or **Groups** accordingly.
3. Choose the name of the user or group to which you want to grant access, and then choose the **Permissions** tab.
4. Do one of the following:
 - To edit a customer managed policy, choose the name of the policy, choose **Edit policy**, and then choose the **JSON** tab. You cannot edit an AWS managed policy. AWS managed policies appear with the AWS icon (). For more information about the difference between AWS managed policies and customer managed policies, see [Managed policies and inline policies \(p. 359\)](#).
 - To edit an inline policy, choose the arrow next to the name of the policy and choose **Edit policy**.
5. In the policy editor, add a new **Statement** element that specifies the following:

```
{
    "Effect": "Allow",
    "Action": "sts:AssumeRole",
```

```
    "Resource": "arn:aws:iam::ACCOUNT-ID:role/ROLE-NAME"  
}
```

Replace the ARN in the statement with the ARN of the role that the user can assume.

6. Follow the prompts on screen to finish editing the policy.

Modifying a role permissions policy (console)

To change the permissions allowed by the role, modify the role's permissions policy (or policies). You cannot modify the permissions policy for a [service-linked role \(p. 168\)](#) in IAM. You might be able to modify the permissions policy within the service that depends on the role. To check whether a service supports this feature, see [AWS services that work with IAM \(p. 611\)](#) and look for the services that have **Yes** in the **Service-linked roles** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

To change the permissions allowed by a role (console)

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane of the IAM console, choose **Roles**.
3. Choose the name of the role that you want to modify, and then choose the **Permissions** tab.
4. Do one of the following:
 - To edit an existing customer managed policy, choose the name of the policy and then choose **Edit policy**.
Note
You cannot edit an AWS managed policy. AWS managed policy appear with the AWS icon  (). For more information about the difference between AWS managed policies and customer managed policies, see [Managed policies and inline policies \(p. 359\)](#).
 - To attach an existing managed policy to the role, choose **Add permissions**.
 - To edit an existing inline policy, choose the arrow next to the name of the policy and choose **Edit Policy**.
 - To embed a new inline policy, choose **Add inline policy**.

Modifying a role description (console)

To change the description of the role, modify the description text.

To change the description of a role (console)

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 IAM console, choose **Roles**.
3. Choose the name of the role to modify.
4. Next to **Role description** and on the far right, choose **Edit**.
5. Type a new description in the box and choose **Save**.

Modifying a role maximum session duration (console)

To specify the maximum session duration setting for roles that are assumed using the console, the AWS CLI, or AWS API, modify the maximum session duration setting value. This setting can have a value from 1 hour to 12 hours. If you do not specify a value, the default maximum of 1 hour is applied. This setting does not limit sessions assumed by AWS services.

To change the maximum session duration setting for roles that are assumed using the console, AWS CLI, or AWS API (console)

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 IAM console, choose **Roles**.
3. Choose the name of the role to modify.
4. Next to **Maximum session duration**, choose a value. Or choose **Custom duration** and type a value (in seconds).
5. Choose **Save**.

Your changes don't take effect until the next time someone assumes this role. To learn how to revoke existing sessions for this role, see [Revoking IAM role temporary security credentials \(p. 271\)](#).

In the AWS Management Console, IAM user sessions are 12 hours by default. IAM users who switch roles in the console are granted the role maximum session duration, or the remaining time in the IAM user's session, whichever is less.

Anyone who assumes the role from the AWS CLI or AWS API can request a longer session, up to this maximum. The `MaxSessionDuration` setting determines the maximum duration of the role session that can be requested.

- To specify a session duration using the AWS CLI use the `duration-seconds` parameter. To learn more, see [Switching to an IAM role \(AWS CLI\) \(p. 256\)](#).
- To specify a session duration using the AWS API, use the `DurationSeconds` parameter. To learn more, see [Switching to an IAM role \(AWS API\) \(p. 262\)](#).

Modifying a role permissions boundary (console)

To change the maximum permissions allowed for a role, modify the role's [permissions boundary \(p. 365\)](#).

To change the policy used to set the permissions boundary for a role

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, choose **Roles**.
3. Choose the name of the role whose [permissions boundary \(p. 365\)](#) you want to change.
4. Choose the **Permissions** tab. If necessary, open the **Permissions boundary** section and then choose **Change boundary**.
5. Select the policy that you want to use for the permissions boundary.
6. Choose **Change boundary**.

Your changes don't take effect until the next time someone assumes this role.

Modifying a role (AWS CLI)

You can use the AWS Command Line Interface to modify a role. To change the set of tags on a role, see [Managing tags on IAM entities \(console\) \(p. 292\)](#).

Topics

- [Modifying a role trust policy \(AWS CLI\) \(p. 277\)](#)

- [Modifying a role permissions policy \(AWS CLI\) \(p. 278\)](#)
- [Modifying a role description \(AWS CLI\) \(p. 279\)](#)
- [Modifying a role maximum session duration \(AWS CLI\) \(p. 279\)](#)
- [Modifying a role permissions boundary \(AWS CLI\) \(p. 279\)](#)

Modifying a role trust policy (AWS CLI)

To change who can assume a role, you must modify the role's trust policy. You cannot modify the trust policy for a [service-linked role \(p. 168\)](#).

Note

If a user is listed as the principal in a role's trust policy but cannot assume the role, check the user's [permissions boundary \(p. 365\)](#). If a permissions boundary is set for the user, then it must allow the `sts:AssumeRole` action.

To modify a role trust policy (AWS CLI)

1. (Optional) If you don't know the name of the role that you want to modify, run the following command to list the roles in your account:
 - [aws iam list-roles](#)
2. (Optional) To view the current trust policy for a role, run the following command:
 - [aws iam get-role](#)
3. To modify the trusted principals that can access the role, create a text file with the updated trust policy. You can use any text editor to construct the policy.

For example, the following trust policy shows how to reference two AWS accounts in the `Principal` element. This allows users within two separate AWS accounts to assume this role.

```
{  
    "Version": "2012-10-17",  
    "Statement": {  
        "Effect": "Allow",  
        "Principal": {"AWS": [  
            "arn:aws:iam::111122223333:root",  
            "arn:aws:iam::444455556666:root"  
        ]},  
        "Action": "sts:AssumeRole"  
    }  
}
```

If you specify a principal in another account, adding an account to the trust policy of a role is only half of establishing the cross-account trust relationship. By default, no users in the trusted accounts can assume the role. The administrator for the newly trusted account must grant the users the permission to assume the role. To do that, the administrator must create or edit a policy that is attached to the user to allow the user access to the `sts:AssumeRole` action. For more information, see the following procedure or [Granting a user permissions to switch roles \(p. 248\)](#).

4. To use the file that you just created to update the trust policy, run the following command:
 - [aws iam update-assume-role-policy](#)

To allow users in a trusted external account to use the role (AWS CLI)

For more information and detail about this procedure, see [Granting a user permissions to switch roles \(p. 248\)](#).

1. Create a JSON file that contains a permissions policy that grants permissions to assume the role. For example, the following policy contains the minimum necessary permissions:

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

Replace the ARN in the statement with the ARN of the role that the user can assume.

2. Run the following command to upload the JSON file that contains the trust policy to IAM:
 - [aws iam create-policy](#)

The output of this command includes the ARN of the policy. Make a note of this ARN because you will need it in a later step.

3. Decide which user or group to attach the policy to. If you don't know the name of the intended user or group, use one of the following commands to list the users or groups in your account:
 - [aws iam list-users](#)
 - [aws iam list-groups](#)
4. Use one of the following commands to attach the policy that you created in the previous step to the user or group:
 - [aws iam attach-user-policy](#)
 - [aws iam attach-group-policy](#)

Modifying a role permissions policy (AWS CLI)

To change the permissions allowed by the role, modify the role's permissions policy (or policies). You cannot modify the permissions policy for a [service-linked role \(p. 168\)](#) in IAM. You might be able to modify the permissions policy within the service that depends on the role. To check whether a service supports this feature, see [AWS services that work with IAM \(p. 611\)](#) and look for the services that have **Yes** in the **Service-linked roles** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

To change the permissions allowed by a role (AWS CLI)

1. (Optional) To view the current permissions associated with a role, run the following commands:
 1. [aws iam list-role-policies](#) to list inline policies
 2. [aws iam list-attached-role-policies](#) to list managed policies
2. The command to update permissions for the role differs depending on whether you are updating a managed policy or an inline policy.

To update a managed policy, run the following command to create a new version of the managed policy:

- [aws iam create-policy-version](#)

To update an inline policy, run the following command:

- [aws iam put-role-policy](#)

Modifying a role description (AWS CLI)

To change the description of the role, modify the description text.

To change the description of a role (AWS CLI)

1. (Optional) To view the current description for a role, run the following command:
 - [aws iam get-role](#)
2. To update a role's description, run the following command with the `description` parameter:
 - [aws iam update-role](#)

Modifying a role maximum session duration (AWS CLI)

To specify the maximum session duration setting for roles that are assumed using the AWS CLI or API, modify the maximum session duration setting's value. This setting can have a value from 1 hour to 12 hours. If you do not specify a value, the default maximum of 1 hour is applied. This setting does not limit sessions assumed by AWS services.

Note

Anyone who assumes the role from the AWS CLI or API can use the `duration-seconds` CLI parameter or the `DurationSeconds` API parameter to request a longer session. The `MaxSessionDuration` setting determines the maximum duration of the role session that can be requested using the `DurationSeconds` parameter. If users don't specify a value for the `DurationSeconds` parameter, their security credentials are valid for one hour.

To change the maximum session duration setting for roles that are assumed using the AWS CLI (AWS CLI)

1. (Optional) To view the current maximum session duration setting for a role, run the following command:
 - [aws iam get-role](#)
2. To update a role's maximum session duration setting, run the following command with the `max-session-duration` CLI parameter or the `MaxSessionDuration` API parameter:
 - [aws iam update-role](#)

Your changes don't take effect until the next time someone assumes this role. To learn how to revoke existing sessions for this role, see [Revoking IAM role temporary security credentials \(p. 271\)](#).

Modifying a role permissions boundary (AWS CLI)

To change the maximum permissions allowed for a role, modify the role's [permissions boundary \(p. 365\)](#).

To change the managed policy used to set the permissions boundary for a role (AWS CLI)

1. (Optional) To view the current [permissions boundary \(p. 365\)](#) for a role, run the following command:
 - [aws iam get-role](#)

2. To use a different managed policy to update the permissions boundary for a role, run the following command:
 - [aws iam put-role-permissions-boundary](#)

A role can have only one managed policy set as a permissions boundary. If you change the permissions boundary, you change the maximum permissions allowed for a role.

Modifying a role (AWS API)

You can use the AWS API to modify a role. To change the set of tags on a role, see [Managing tags on IAM entities \(console\) \(p. 292\)](#).

Topics

- [Modifying a role trust policy \(AWS API\) \(p. 280\)](#)
- [Modifying a role permissions policy \(AWS API\) \(p. 281\)](#)
- [Modifying a role description \(AWS API\) \(p. 282\)](#)
- [Modifying a role maximum session duration \(AWS API\) \(p. 282\)](#)
- [Modifying a role permissions boundary \(AWS API\) \(p. 283\)](#)

Modifying a role trust policy (AWS API)

To change who can assume a role, you must modify the role's trust policy. You cannot modify the trust policy for a [service-linked role \(p. 168\)](#).

Note

If a user is listed as the principal in a role's trust policy but cannot assume the role, check the user's [permissions boundary \(p. 365\)](#). If a permissions boundary is set for the user, then it must allow the `sts:AssumeRole` action.

To modify a role trust policy (AWS API)

1. (Optional) If you don't know the name of the role that you want to modify, call the following operation to list the roles in your account:
 - [ListRoles](#)
2. (Optional) To view the current trust policy for a role, call the following operation:
 - [GetRole](#)
3. To modify the trusted principals that can access the role, create a text file with the updated trust policy. You can use any text editor to construct the policy.

For example, the following trust policy shows how to reference two AWS accounts in the `Principal` element. This allows users within two separate AWS accounts to assume this role.

```
{  
    "Version": "2012-10-17",  
    "Statement": {  
        "Effect": "Allow",  
        "Principal": {"AWS": [  
            "arn:aws:iam::111122223333:root",  
            "arn:aws:iam::444455556666:root"  
        ]},  
        "Action": "sts:AssumeRole"  
    }  
}
```

}

If you specify a principal in another account, adding an account to the trust policy of a role is only half of establishing the cross-account trust relationship. By default, no users in the trusted accounts can assume the role. The administrator for the newly trusted account must grant the users the permission to assume the role. To do that, the administrator must create or edit a policy that is attached to the user to allow the user access to the `sts:AssumeRole` action. For more information, see the following procedure or [Granting a user permissions to switch roles \(p. 248\)](#).

4. To use the file that you just created to update the trust policy, call the following operation:
 - [UpdateAssumeRolePolicy](#)

To allow users in a trusted external account to use the role (AWS API)

For more information and detail about this procedure, see [Granting a user permissions to switch roles \(p. 248\)](#).

1. Create a JSON file that contains a permissions policy that grants permissions to assume the role. For example, the following policy contains the minimum necessary permissions:

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

Replace the ARN in the statement with the ARN of the role that the user can assume.

2. Call the following operation to upload the JSON file that contains the trust policy to IAM:
 - [CreatePolicy](#)
3. Decide which user or group to attach the policy to. If you don't know the name of the intended user or group, call one of the following operations to list the users or groups in your account:
 - [ListUsers](#)
 - [ListGroups](#)
4. Call one of the following operations to attach the policy that you created in the previous step to the user or group:
 - API: [AttachUserPolicy](#)
 - [AttachGroupPolicy](#)

Modifying a role permissions policy (AWS API)

To change the permissions allowed by the role, modify the role's permissions policy (or policies). You cannot modify the permissions policy for a [service-linked role \(p. 168\)](#) in IAM. You might be able to modify the permissions policy within the service that depends on the role. To check whether a service supports this feature, see [AWS services that work with IAM \(p. 611\)](#) and look for the services that have **Yes** in the **Service-linked roles** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

To change the permissions allowed by a role (AWS API)

1. (Optional) To view the current permissions associated with a role, call the following operations:
 1. [ListRolePolicies](#) to list inline policies
 2. [ListAttachedRolePolicies](#) to list managed policies
2. The operation to update permissions for the role differs depending on whether you are updating a managed policy or an inline policy.

To update a managed policy, call the following operation to create a new version of the managed policy:

- [CreatePolicyVersion](#)

To update an inline policy, call the following operation:

- [PutRolePolicy](#)

Modifying a role description (AWS API)

To change the description of the role, modify the description text.

To change the description of a role (AWS API)

1. (Optional) To view the current description for a role, call the following operation:
 - [GetRole](#)
2. To update a role's description, call the following operation with the description parameter:
 - [UpdateRole](#)

Modifying a role maximum session duration (AWS API)

To specify the maximum session duration setting for roles that are assumed using the AWS CLI or API, modify the maximum session duration setting's value. This setting can have a value from 1 hour to 12 hours. If you do not specify a value, the default maximum of 1 hour is applied. This setting does not limit sessions assumed by AWS services.

Note

Anyone who assumes the role from the AWS CLI or API can use the `duration-seconds` CLI parameter or the `DurationSeconds` API parameter to request a longer session. The `MaxSessionDuration` setting determines the maximum duration of the role session that can be requested using the `DurationSeconds` parameter. If users don't specify a value for the `DurationSeconds` parameter, their security credentials are valid for one hour.

To change the maximum session duration setting for roles that are assumed using the API (AWS API)

1. (Optional) To view the current maximum session duration setting for a role, call the following operation:
 - [GetRole](#)
2. To update a role's maximum session duration setting, call the following operation with the `max-sessionduration` CLI parameter or the `MaxSessionDuration` API parameter:
 - [UpdateRole](#)

Your changes don't take effect until the next time someone assumes this role. To learn how to revoke existing sessions for this role, see [Revoking IAM role temporary security credentials \(p. 271\)](#).

Modifying a role permissions boundary (AWS API)

To change the maximum permissions allowed for a role, modify the role's [permissions boundary \(p. 365\)](#).

To change the managed policy used to set the permissions boundary for a role (AWS API)

1. (Optional) To view the current [permissions boundary \(p. 365\)](#) for a role, call the following operation:
 - [GetRole](#)
2. To use a different managed policy to update the permissions boundary for a role, call the following operation:
 - [PutRolePermissionsBoundary](#)

A role can have only one managed policy set as a permissions boundary. If you change the permissions boundary, you change the maximum permissions allowed for a role.

Deleting roles or instance profiles

If you no longer need a role, we recommend that you delete the role and its associated permissions. That way you don't have an unused entity that is not actively monitored or maintained.

If the role was associated with an EC2 instance, you can also remove the role from the instance profile and then delete the instance profile.

Warning

Make sure that 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 that are running on the instance.

If you prefer not to permanently delete a role, you can disable a role. To do this, change the role's policies and then revoke all current sessions. For example, you could add a policy to the role that denied access to all of AWS. You could also edit the trust policy to deny access to anyone attempting to assume the role. For more information about revoking sessions, see [Revoking IAM role temporary security credentials \(p. 271\)](#).

Topics

- [View role access \(p. 284\)](#)
- [Deleting a service-linked role \(p. 284\)](#)
- [Deleting an IAM role \(console\) \(p. 284\)](#)
- [Deleting an IAM role \(AWS CLI\) \(p. 285\)](#)
- [Deleting an IAM role \(AWS API\) \(p. 286\)](#)
- [Related information \(p. 286\)](#)

View role access

Before you delete a role, we recommend that you review when the role was last used. You can do this using the AWS Management Console, the AWS CLI, or the AWS API. It's important to view this information because you don't want to remove access from someone who is using it.

The date of the role's last activity might not match the last date reported in the **Access Advisor** tab. The [Access Advisor \(p. 477\)](#) tab reports activity only for services that are allowed by the role's permissions policies. The date of the role's last activity includes the last attempt to access any service in AWS.

Note

The tracking period for a role's last activity and Access Advisor data is for the trailing 400 days. This period can be shorter if your Region began supporting these features within the last year. The role might have been used more than 400 days ago. For more information about the tracking period, see [Where AWS tracks last accessed information \(p. 476\)](#).

To view when a role was last used (console)

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, choose **Roles**.
3. Find the row of the role whose activity you want to view. You can use the search field to narrow the results. View the **Last activity** column to see the number of days since the role was last used. If the role has not been used within the tracking period, then the table displays **None**.
4. Choose the name of the role to view more information. The role's **Summary** page also includes **Last activity**, which displays the date that the role was last used. If the role has not been used within the last 400 days, then **Last activity** displays **Not accessed in the tracking period**.

To view when a role was last used (AWS CLI)

`aws iam get-role` - Run this command to return information about a role, including the `RoleLastUsed` object. This object contains the `LastUsedDate` and the `Region` in which the role was last used. If `RoleLastUsed` is present but does not contain a value, then the role has not been used within the tracking period.

To view when a role was last used (AWS API)

`GetRole` - Call this operation to return information about a role, including the `RoleLastUsed` object. This object contains the `LastUsedDate` and the `Region` in which the role was last used. If `RoleLastUsed` is present but does not contain a value, then the role has not been used within the tracking period.

Deleting a service-linked role

If the role is a [service-linked role \(p. 168\)](#), review the documentation for the linked service to learn how to delete the role. You can view the service-linked roles in your account by going to the IAM **Roles** page in the console. Service-linked roles appear with **(Service-linked role)** in the **Trusted entities** column of the table. A banner on the role's **Summary** page also indicates that the role is a service-linked role.

If the service does not include documentation for deleting the service-linked role, you can use the IAM console, AWS CLI, or API to delete the role. For more information, see [Deleting a service-linked role \(p. 218\)](#).

Deleting an IAM role (console)

When you use the AWS Management Console to delete a role, IAM also automatically deletes the policies associated with the role. It also deletes any Amazon EC2 instance profile that contains the role.

Important

In some cases, a role might be associated with an Amazon EC2 instance profile, and the role and the instance profile might have the same name. In that case you can use the AWS Management Console to delete the role and the instance profile. This linkage happens automatically for roles and instance profiles that you create in the console. If you created the role from the AWS CLI, Tools for Windows PowerShell, or the AWS API, then the role and the instance profile might have different names. In that case you cannot use the console to delete them. Instead, you must use the AWS CLI, Tools for Windows PowerShell, or AWS API to first remove the role from the instance profile. You must then take a separate step to delete the role.

To delete a role (console)

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, choose **Roles**, and then select the check box next to the role name that you want to delete.
3. At the top of the page, choose **Delete role**.
4. In the confirmation dialog box, review the last accessed information, which shows when each of the selected roles last accessed an AWS service. This helps you to confirm whether the role is currently active. If you want to proceed, choose **Yes, Delete**. If you are sure, you can proceed with the deletion even if the last accessed information is still loading.

Note

You cannot use the console to delete an instance profile unless it has the same name as the role. In addition, you must delete the instance profile 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 AWS CLI or AWS API. For more information, see the following sections.

Deleting an IAM role (AWS CLI)

When you use the AWS CLI to delete a role, you must first delete the policies that are 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 (AWS CLI)

1. If you don't know the name of the role that you want to delete, enter the following command to list the roles in your account:

```
aws iam list-roles
```

The list includes the Amazon Resource Name (ARN) of each role. Use the role name, not the ARN, to refer to roles with the CLI commands. For example, if a role has the following ARN: arn:aws:iam::123456789012:role/myrole, you refer to the role as **myrole**.

2. Remove the role from all instance profiles that the role is in.
 - a. To list all instance profiles that the role is associated with, enter the following command:

```
aws iam list-instance-profiles-for-role --role-name role-name
```

- b. To remove the role from an instance profile, enter the following command for each instance profile:

```
aws iam remove-role-from-instance-profile --instance-profile-name instance-profile-name --role-name role-name
```

3. Delete all policies that are associated with the role.
 - a. To list all policies that are in the role, enter the following command:

```
aws iam list-role-policies --role-name role-name
```
 - b. To delete each policy from the role, enter the following command for each policy:

```
aws iam delete-role-policy --role-name role-name --policy-name policy-name
```
4. Enter the following command to delete the role:

```
aws iam delete-role --role-name role-name
```
5. If you do not plan to reuse the instance profiles that were associated with the role, you can enter the following command to delete them:

```
aws iam delete-instance-profile --instance-profile-name instance-profile-name
```

Deleting an IAM role (AWS API)

When you use the IAM API to delete a role, you must first delete the policies associated with the role. Also, if you want to delete the associated instance profile that contains the role, you must delete it separately.

To delete a role (AWS API)

1. To list all instance profiles that a role is in, call [ListInstanceProfilesForRole](#).

To remove the role from all instance profiles that the role is in, call [RemoveRoleFromInstanceProfile](#). You must pass the role name and instance profile name.

If you are not going to reuse an instance profile that was associated with the role, call [DeleteInstanceProfile](#) to delete it.

2. To list all policies for a role, call [ListRolePolicies](#).

To delete all policies that are associated with the role, call [DeleteRolePolicy](#). You must pass the role name and policy name.

3. Call [DeleteRole](#) to delete the role.

Related information

For general information about instance profiles, see [Using instance profiles \(p. 270\)](#).

For general information about service-linked roles, see [Using service-linked roles \(p. 213\)](#).

How IAM roles differ from resource-based policies

For some AWS services, you can grant cross-account access to your resources. To do this, you attach a policy directly to the resource that you want to share, instead of using a role as a proxy. The resource that you want to share must support [resource-based policies \(p. 374\)](#). Unlike an identity-based policy, a resource-based policy specifies who (which principal) can access that resource.

Note

IAM roles and resource-based policies delegate access across accounts only within a single partition. For example, assume that you have an account in US West (N. California) in the

standard aws partition. You also have an account in China (Beijing) in the aws-cn partition. You can't use an Amazon S3 resource-based policy in your account in China (Beijing) to allow access for users in your standard aws account.

Cross-account access with a resource-based policy has some advantages over cross-account access with a role. With a resource that is accessed through a resource-based policy, the principal still works in the trusted account and does not have to give up his or her permissions to receive the role permissions. In other words, the principal continues to have access to resources in the trusted account at the same time as he or she has access to the resource in the trusting account. This is useful for tasks such as copying information to or from the shared resource in the other account. To learn whether principals in accounts outside of your zone of trust (trusted organization or account) have access to assume your roles, see [What is IAM Access Analyzer?](#).

The principals that you can specify in a resource based policy include accounts, IAM users, federated users, IAM roles, assumed-role sessions, or AWS services. For more information, see [Specifying a principal \(p. 631\)](#).

The following list includes some of the AWS services that support resource-based policies. For a complete list of the growing number of AWS services that support attaching permission policies to resources instead of principals, see [AWS services that work with IAM \(p. 611\)](#) and look for the services that have Yes in the **Resource Based** column.

- **Amazon S3 buckets** – The policy is attached to the bucket, but the policy controls access to both the bucket and the objects in it. For more information, go to [Access Control](#) in the *Amazon Simple Storage Service Developer Guide*.

In some cases, it may be best to use roles for cross-account access to Amazon S3. For more information, see the [example walkthroughs](#) in the *Amazon Simple Storage Service Developer Guide*.

- **Amazon Simple Notification Service (Amazon SNS) topics** – For more information, go to [Managing Access to Your Amazon SNS Topics](#) in the *Amazon Simple Notification Service Developer Guide*.
- **Amazon Simple Queue Service (Amazon SQS) queues** – For more information, go to [Appendix: The Access Policy Language](#) in the *Amazon Simple Queue Service Developer Guide*.

About delegating AWS permissions in a resource-based policy

If a resource grants permissions to principals in your account, you can then delegate those permissions to specific IAM identities. Identities are users, groups of users, or roles in your account. You delegate permissions by attaching a policy to the identity. You can grant up to the maximum permissions that are allowed by the resource-owning account.

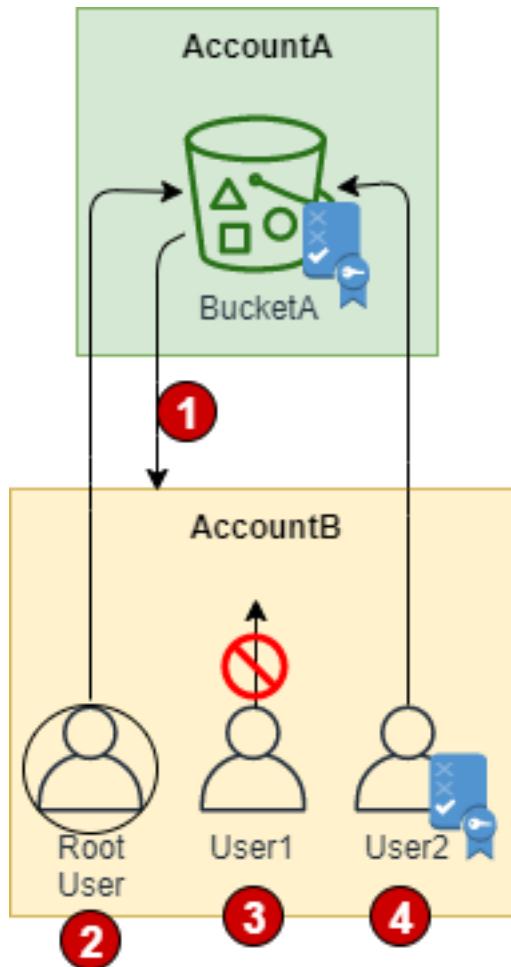
Assume that a resource-based policy allows all principals in your account full administrative access to a resource. Then you can delegate full access, read-only access, or any other partial access to principals in your AWS account. Alternatively, if the resource-based policy allows only list permissions, then you can delegate only list access. If you try to delegate more permissions than your account has, your principals will still have only list access. For information about attaching a policy to an IAM identity, see [Managing IAM policies \(p. 438\)](#).

Note

IAM roles and resource-based policies delegate access across accounts only within a single partition. For example, you can't add cross-account access between an account in the standard aws partition and an account in the aws-cn partition.

For example, assume that you manage AccountA and AccountB. In AccountA, you have the Amazon S3 bucket named BucketA. You attach a resource-based policy to BucketA that allows all principals in AccountB full access to objects in your bucket. They can create, read, or delete any objects in that bucket. In AccountB, you attach a policy to the IAM user named User2. That policy allows the user read-

only access to the objects in BucketA. That means that User2 can view the objects, but not create, edit, or delete them.



1. AccountA gives AccountB full access to BucketA by naming AccountB as a principal in the resource-based policy. As a result, AccountB is authorized to perform any action on BucketA, and the AccountB administrator can delegate access to its users in AccountB.
2. The AccountB root user has all of the permissions that are granted to the account. Therefore, the root user has full access to BucketA.
3. The AccountB administrator does not give access to User1. By default, users do not have any permissions except those that are explicitly granted. Therefore, User1 does not have access to BucketA.
4. The AccountB administrator grants User2 read-only access to BucketA. User2 can view the objects in the bucket. The maximum level of access that AccountB can delegate is the access level that is granted to the account. In this case, the resource-based policy granted full access to AccountB, but User2 is granted only read-only access.

IAM evaluates a principal's permissions at the time the principal makes a request. Therefore, if you use wildcards (*) to give users full access to your resources, principals can access any resources that your AWS account has access to. This is true even for resources you add or gain access to after creating the user's policy.

In the preceding example, if AccountB had attached a policy to User2 that allowed full access to all resources in all accounts, User2 would automatically have access to any resources that AccountB has

access to. This includes the `BucketA` access and access to any other resources granted by resource-based policies in AccountA.

Important

Give access only to entities you trust, and give the minimum level of access necessary. Whenever the trusted entity is another AWS account, that account can in turn delegate access to any of its IAM users. The trusted AWS account can delegate access only to the extent that it has been granted access; it cannot delegate more access than the account itself has been granted.

For information about permissions, policies, and the permission policy language that you use to write policies, see [Access management for AWS resources \(p. 350\)](#).

Tagging IAM users and roles

You can use IAM tags to add custom attributes to an IAM entity (user or role) using a tag key-value pair. For example, to add location information to a user, you can add the tag key `location` and the tag value `us_wa_seattle`. Or you could use three separate location tag key-value pairs: `loc-country = us`, `loc-state = wa`, and `loc-city = seattle`. You can use tags to control an entity's access to resources or to control what tags can be attached to an entity. To learn more about using tags to control access, see [Controlling access to and for IAM users and roles using IAM resource tags \(p. 384\)](#).

You can also use tags in AWS STS to add custom attributes when you assume a role or federate a user. For more information, see [Passing session tags in AWS STS \(p. 293\)](#).

Choose an AWS tag naming convention

When you begin attaching tags to your IAM users and roles, choose your tag naming convention carefully. Apply the same convention to all of your AWS tags. This is especially important if you use tags in policies to control access to AWS resources. If you already use tags in AWS, review your naming convention and adjust it accordingly. To learn more about tagging categories and strategies, see [AWS Tagging AWS Resources](#) in the *AWS General Reference Guide*. To view tagging use cases and best practices, download the [Tagging Best Practices](#) whitepaper.

Rules for tagging in IAM and AWS STS

A number of conventions govern the creation and application of tags in IAM and AWS STS.

Naming tags

Observe the following conventions when formulating a tag naming convention for IAM users, IAM roles, AWS STS assume-role sessions, and AWS STS federated user sessions:

- Tag keys and values can include any combination of letters, numbers, spaces, and `_ . : / = + - @` symbols.
- Tag key-value pairs are not case sensitive, but case is preserved. This means that you cannot have separate `Department` and `department` tag keys. If you have tagged a user with the `Department=foo` tag and you add the `department=bar` tag, it replaces the first tag. A second tag is not added.
- You cannot create a tag key or value that begins with the text `aws:`. This tag prefix is reserved for AWS internal use.
- You can create a tag with an empty value such as `phoneNumber = .` You cannot create an empty tag key.

- You cannot specify multiple values in a single tag, but you can create a custom multivalue structure in the single value. For example, assume that the user Zhang works on the engineering team and the QA team. If you attach the `team = Engineering` tag and then attach the `team = QA` tag, you change the value of the tag from `Engineering` to `QA`. Instead, you can include multiple values in a single tag with a custom separator. In this example, you could attach the `team = Engineering:QA` tag to Zhang.

Note

To control access to engineers in this example using the `team` tag, you must create a policy that allows for every configuration that might include `Engineering`, including `Engineering:QA`. To learn more about using tags in policies, see [Controlling access to and for IAM users and roles using IAM resource tags \(p. 384\)](#).

Applying and editing tags

Observe the following conventions when attaching tags to IAM entities (users or roles):

- You can tag users or roles but not groups or policies.
- You cannot use Tag Editor to tag IAM entities. Tag Editor does not support IAM tags. For information about using Tag Editor with other services, see [Working with Tag Editor in the AWS Management Console User Guide](#).
- To tag an IAM entity, you must have specific permissions. To tag or untag roles and users, you must also have permission to list tags. For more information, see [Permissions required for tagging IAM entities \(p. 290\)](#) following.
- The number and size of IAM resources in an AWS account are limited. For more information, see [IAM and STS quotas \(p. 606\)](#).
- You can apply the same tag to multiple IAM entities. For example, suppose you have a department named `AWS_Development` with 12 members. You can have 12 users and a role with the tag key of `department` and a value of `awsDevelopment` (`department = awsDevelopment`). You can also use the same tag on resources in other [services that support tagging \(p. 611\)](#).
- An IAM entity cannot have multiple instances of the same tag key. For example, if you have a user with the tag key-value pair `costCenter = 1234`, you can then attach the tag key-value pair `costCenter = 5678`. IAM updates the value of the `costCenter` tag to `5678`.
- To edit a tag that is attached to an IAM user or role, attach a tag with a new value to overwrite the existing tag. For example, assume that you have a user with the tag key-value pair `department = Engineering`. If you need to move the user to the QA department, then you can attach the `department = QA` tag key-value pair to the user. This results in the `Engineering` value of the `department` tag key being replaced with the `QA` value.

Permissions required for tagging IAM entities

You must configure permissions to allow an IAM entity (user or role) to tag other entities. You can specify one or all of the following IAM tag actions in an IAM policy:

- `iam>ListRoleTags`
- `iam>ListUserTags`
- `iam:TagRole`
- `iam:TagUser`
- `iam:UntagRole`
- `iam:UntagUser`

To allow an IAM entity to add, list, or remove a tag for a specific user

Add the following statement to the permissions policy for the IAM entity that needs to manage tags. Use your account number and replace `<username>` with the name of the user that needs to be managed. To learn how to create a policy using this example JSON policy document, see [the section called “Creating policies on the JSON tab” \(p. 440\)](#).

```
{  
    "Effect": "Allow",  
    "Action": [  
        "iam>ListUserTags",  
        "iam:TagUser",  
        "iam:UntagUser"  
    ],  
    "Resource": "arn:aws:iam:*:<account-number>:user/<username>"  
}
```

To allow an IAM user to self-manage tags

Add the following statement to the permissions policy for users to allow users to manage their own tags. To learn how to create a policy using this example JSON policy document, see [the section called “Creating policies on the JSON tab” \(p. 440\)](#).

```
{  
    "Effect": "Allow",  
    "Action": [  
        "iam>ListUserTags",  
        "iam:TagUser",  
        "iam:UntagUser"  
    ],  
    "Resource": "arn:aws:iam*:user/${aws:username}"  
}
```

To allow an IAM entity to add a tag to a specific user

Add the following statement to the permissions policy for the IAM entity that needs to add, but not remove, tags for a specific user.

Note

The `iam:AddRoleTags` and `iam:AddUserTags` actions require that you also include the `iam>ListRoleTags` and `iam>ListUserTags` actions.

To use this policy, replace `<username>` with the name of the user that needs to be managed. To learn how to create a policy using this example JSON policy document, see [the section called “Creating policies on the JSON tab” \(p. 440\)](#).

```
{  
    "Effect": "Allow",  
    "Action": [  
        "iam>ListUserTags",  
        "iam:TagUser"  
    ],  
    "Resource": "arn:aws:iam*:<account-number>:user/<username>"  
}
```

To allow an IAM entity to add, list, or remove a tag for a specific role

Add the following statement to the permissions policy for the IAM entity that needs to manage tags. Replace `<rolename>` with the name of the role that needs to be managed. To learn how to create a policy using this example JSON policy document, see [the section called “Creating policies on the JSON tab” \(p. 440\)](#).

```
{  
    "Effect": "Allow",  
    "Action": [  
        "iam>ListRoleTags",  
        "iam:TagRole",  
        "iam:UntagRole"  
    ],  
    "Resource": "arn:aws:iam:*:<account-number>:role/<rolename>"  
}
```

Alternatively, you can use an AWS managed policy such as [IAMFullAccess](#) to provide full access to IAM.

Managing tags on IAM entities (console)

You can manage tags for IAM users or roles from the AWS Management Console.

To manage tags on users or roles (console)

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, choose **Roles** or **Users** and then choose the name of the entity that you want to edit.
3. Choose the **Tags** tab and then complete one of the following actions:
 - Choose **Add tags** if the entity does not yet have tags.
 - Choose **Edit tags** to manage the existing set of tags.
4. Add or remove tags to complete the set of tags. Then choose **Save changes**.

Managing tags on IAM entities (AWS CLI or AWS API)

You can list, attach, or remove tags for IAM users and roles. You can use the AWS CLI or the AWS API to manage tags for IAM users and roles.

To list the tags currently attached to an IAM role (AWS CLI or AWS API)

- AWS CLI: [aws iam list-role-tags](#)
- AWS API: [ListRoleTags](#)

To list the tags currently attached to an IAM user (AWS CLI or AWS API)

- AWS CLI: [aws iam list-user-tags](#)
- AWS API: [ListUserTags](#)

To attach tags to an IAM role (AWS CLI or AWS API)

- AWS CLI: [aws iam tag-role](#)
- AWS API: [TagRole](#)

To attach tags to an IAM user (AWS CLI or AWS API)

- AWS CLI: [aws iam tag-user](#)

- AWS API: [TagUser](#)

To remove tags from an IAM role (AWS CLI or AWS API)

- AWS CLI: `aws iam untag-role`
- AWS API: [UntagRole](#)

To remove tags from an IAM user (AWS CLI or AWS API)

- AWS CLI: `aws iam untag-user`
- AWS API: [UntagUser](#)

For information about attaching tags to resources for other AWS services, see the documentation for those services.

For information about using tags to set more granular permissions with IAM permissions policies, see [IAM policy elements: Variables and tags \(p. 658\)](#).

Passing session tags in AWS STS

Session tags are key-value pair attributes that you pass when you assume an IAM role or federate a user in AWS STS. You do this by making an AWS CLI or AWS API request through STS or through your identity provider (IdP). When you use AWS STS to request temporary security credentials, you generate a session. Sessions expire and have [credentials](#), such as an access key pair and a session token. When you use the session credentials to make a subsequent request, the [request context \(p. 642\)](#) includes the [aws:PrincipalTag \(p. 700\)](#) context key. You can use the `aws:PrincipalTag` key in the Condition element of your policies to allow or deny access based on those tags.

When you use temporary credentials to make a request, your principal might include a set of tags. These tags come from the following sources:

1. **Session tags** – These tags were passed when you assumed the role or federated the user using the AWS CLI or AWS API. For more information about these operations, see [Session tagging operations \(p. 294\)](#) below.
2. **Incoming transitive session tags** – These tags were inherited from a previous session in a role chain. For more information, see [Chaining roles with session tags \(p. 299\)](#) later in this topic.
3. **IAM tags** – These tags were attached to the IAM role that you assumed.

Topics

- [Session tagging operations \(p. 294\)](#)
- [Things to know about session tags \(p. 294\)](#)
- [Permissions required to add session tags \(p. 295\)](#)
- [Passing session tags using AssumeRole \(p. 297\)](#)
- [Passing session tags using AssumeRoleWithSAML \(p. 297\)](#)
- [Passing session tags using AssumeRoleWithWebIdentity \(p. 298\)](#)
- [Passing session tags using GetFederationToken \(p. 299\)](#)
- [Chaining roles with session tags \(p. 299\)](#)
- [Using session tags for ABAC \(p. 300\)](#)
- [Viewing session tags in CloudTrail \(p. 301\)](#)

Session tagging operations

You can pass session tags using the following AWS CLI or AWS API operations in AWS STS. *The AWS Management Console [Switch Role](#) (p. 253) feature does not allow you to pass session tags.*

You can also set the session tags as transitive. Transitive tags persist during role chaining. For more information, see [Chaining roles with session tags](#) (p. 299).

Comparing methods for passing session tags

Method	Who can assume the role	Method to pass tags	Method to set transitive tags
assume-role CLI or AssumeRole API operation	IAM user or a session	Tags API parameter or --tags CLI option	TransitiveTagKeys API parameter or --transitive-tag-keys CLI option
assume-role-with-saml CLI or AssumeRoleWithSAML API operation	Any user authenticated using SAML identity provider	PrincipalTag SAML attribute	TransitiveTagKeys SAML Attribute
assume-role-with-web-identity CLI or AssumeRoleWithWebIdentity API operation	Any user authenticated using a web identity provider	PrincipalTag web identity token	TransitiveTagKeys web identity token
get-federation-token CLI or GetFederationToken API operation	IAM user or root user	Tags API parameter or --tags CLI option	Not supported

Operations that support session tagging can fail if any of the following conditions are true:

- You pass more than 50 session tags.
- The plain text of your session tag keys exceeds 128 characters.
- The plain text of your session tag values exceeds 256 characters.
- The total size of the plain text of session policies exceeds 2048 characters.
- The total packed size of the session policies and session tags combined is too large. If the operation fails, the error message indicates by percentage how close the policies and tags combined are to the upper size limit.

Things to know about session tags

Before you use session tags, review the following details about sessions and tags.

- When using session tags, trust policies for all roles connected to the identity provider (IdP) that is passing tags must have the [sts:TagSession](#) (p. 295) permission. For roles that don't have this permission in the trust policy, the `AssumeRole` operation will fail.
- Session tags are principal tags that you specify while requesting a session. The tags apply to requests that you make using the session's credentials.
- Session tags are key-value pairs. For example, to add contact information to a session, you can add the session tag key `email` and the tag value `johndoe@example.com`.

- Session tags must follow the [rules for naming tags in IAM and AWS STS \(p. 289\)](#). This topic includes information about case sensitivity and restricted prefixes that apply to your session tags.
- New session tags override existing assumed role or federated user tags with the same tag key, regardless of case.
- You cannot pass session tags using the AWS Management Console.
- Session tags are valid only for the current session.
- Session tags support [role chaining \(p. 169\)](#). By default, tags are not passed to subsequent role sessions. However, you can set session tags as transitive. Transitive tags persist during role chaining. For more information, see [Chaining roles with session tags \(p. 299\)](#).
- You can use session tags to control access to resources or to control what tags can be passed into a subsequent session. For more information, see [IAM Tutorial: Use SAML session tags for ABAC \(p. 57\)](#).
- You can view the principal tags for your session, including its session tags, in the AWS CloudTrail logs. For more information, see [Viewing session tags in CloudTrail \(p. 301\)](#).
- You must pass a single value for each session tag. Multivalued session tags are not supported.
- You can pass a maximum of 50 session tags. The number and size of IAM resources in an AWS account are limited. For more information, see [IAM and STS quotas \(p. 606\)](#).
- An AWS conversion compresses the passed session policies and session tags combined into a packed binary format that has a separate limit. If you exceed this limit, the AWS CLI or AWS API error message indicates by percentage how close the policies and tags combined are to the upper size limit.

Permissions required to add session tags

In addition to the action that matches the API operation, you must have the following permissions-only action in your policy:

sts:TagSession

Important

When using session tags, the role trust policies for all roles connected to an identity provider (IdP) must have the `sts:TagSession` permission. The `AssumeRole` operation will fail for any role connected to an IdP that is passing session tags without this permission. If you don't want to update the role trust policy for each role, you can use a separate IdP instance for passing session tags. Then add the `sts:TagSession` permission to only the roles that are connected to the separate IdP.

You can use the `sts:TagSession` action with the following condition keys.

- [aws:PrincipalTag \(p. 700\)](#) – Use this key to compare the tag that is attached to the principal making the request with the tag that you specify in the policy. For example, you can allow a principal to pass session tags only if the principal making the request has the specified tags.
- [aws:RequestTag \(p. 702\)](#) – Use this key to compare the tag key-value pair that was passed in the request with the tag pair that you specify in the policy. For example, you can allow the principal to pass the specified session tags, but only with the specified values.
- [aws:ResourceTag \(p. 703\)](#) – Use this key to compare the tag key-value pair that you specify in the policy with the key-value pair that is attached to the resource. For example, you can allow the principal to pass session tags only if the role they are assuming includes the specified tags.
- [aws:TagKeys \(p. 705\)](#) – Use this key to compare the tag keys in a request with the keys that you specify in the policy. For example, you can allow the principal to pass only session tags with the specified tag keys. This condition key limits the maximum set of session tags that can be passed.
- [sts:TransitiveTagKeys \(p. 719\)](#) – Use this key to compare the transitive session tag keys in the request with those specified in the policy. For example, you can write a policy to allow a principal to set only specific tags as transitive. Transitive tags persist during role chaining. For more information, see [Chaining roles with session tags \(p. 299\)](#).

For example, the following [role trust policy](#) (p. 170) allows the `test-session-tags` user to assume the role to which the policy is attached. When that user assumes the role, they must use the AWS CLI or AWS API to pass the three required session tags and the required [external ID](#) (p. 225). Additionally, the user can choose to set the `Project` and `Department` tags as transitive.

Example Example role trust policy for session tags

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowIamUserAssumeRole",
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Principal": {"AWS": "arn:aws:iam::123456789012:user/test-session-tags"},
            "Condition": {
                "StringLike": {
                    "aws:RequestTag/ProjectCostCenterDepartmentProjectCostCenterDepartmentEngineering",
                        "Marketing"
                    ]
                },
                "ForAllValues:StringEquals": {
                    "sts:TransitiveTagKeys": [
                        "Project",
                        "Department"
                    ]
                }
            }
        }
    ]
}
```

What does this policy do?

- The `AllowIamUserAssumeRole` statement allows the `test-session-tags` user to assume the role to which the policy is attached. When that user assumes the role, they must pass the required session tags and [external ID](#) (p. 225).
- The first condition block of this statement requires the user to pass the `Project`, `CostCenter`, and `Department` session tags. The tag values don't matter in this statement, so we used wildcards (*) for the tag values. This block ensures that user passes at least these three session tags or the operation will fail. The user can pass additional tags.
- The second condition block requires the user to pass an [external ID](#) (p. 225) with the value `Example987`.

- The `AllowPassSessionTagsAndTransitive` statement allows the `sts:TagSession` permissions-only action. This action must be allowed before the user can pass session tags. If your policy includes the first statement without the second statement, the user can't assume the role.
 - The first condition block of this statement allows the user to pass any value for the `CostCenter` and `Project` session tags. You do this by using wildcards (*) for the tag value in the policy, which requires that you use the [StringLike \(p. 644\)](#) condition operator.
 - The second condition block allows the user to pass only the `Engineering` or `Marketing` value for the `Department` session tag.
 - The third condition block lists the maximum set of tags that can be set as transitive. The user can choose to set a subset or no tags as transitive. But they cannot set additional tags as transitive. You can require that they set at least one of the tags as transitive by adding another condition block that includes `"Null": {"sts:TransitiveTagKeys": "false"}`.

Passing session tags using AssumeRole

The `AssumeRole` operation returns a set of temporary credentials that you can use to access AWS resources. You can use IAM user or role credentials to call `AssumeRole`. To pass session tags while assuming a role, use the `--tags` AWS CLI option or the `Tags` AWS API parameter.

To set tags as transitive, use the `--transitive-tag-keys` AWS CLI option or the `TransitiveTagKeys` AWS API parameter. Transitive tags persist during role chaining. For more information, see [Chaining roles with session tags \(p. 299\)](#).

The following example shows a sample request that uses `AssumeRole`. In this example, when you assume the `my-role-example` role, you create a session named `my-session`. You add the session tag key-value pairs `Project = Automation`, `CostCenter = 12345`, and `Department = Engineering`. You also set the `Project` and `Department` tags as transitive by specifying their keys.

Example Example AssumeRole CLI request

```
aws sts assume-role \
--role-arn arn:aws:iam::123456789012:role/my-role-example \
--role-session-name my-session \
--tags Key=Project,Value=Automation Key=CostCenter,Value=12345
Key=Department,Value=Engineering \
--transitive-tag-keys Project Department \
--external-id Example987
```

Passing session tags using AssumeRoleWithSAML

The `AssumeRoleWithSAML` operation is authenticated using SAML-based federation. This operation returns a set of temporary credentials that you can use to access AWS resources. For more information about using SAML-based federation for AWS Management Console access, see [Enabling SAML 2.0 federated users to access the AWS Management Console \(p. 203\)](#). For details about AWS CLI or AWS API access, see [About SAML 2.0-based federation \(p. 182\)](#). For a tutorial of setting up SAML federation for your Active Directory users, see [AWS Federated Authentication with Active Directory Federation Services \(ADFS\) in the AWS Security Blog](#).

As an administrator, you can allow members of your company directory to federate into AWS using the AWS STS `AssumeRoleWithSAML` operation. To do this, you must complete the following tasks:

1. [Configure your network as a SAML provider for AWS \(p. 197\)](#)
2. [Create a SAML provider in IAM \(p. 193\)](#)
3. [Configure a role and its permissions in AWS for your federated users \(p. 241\)](#)
4. [Finish configuring the SAML IdP and create assertions for the SAML authentication response \(p. 199\)](#)

AWS includes partners that have certified the end-to-end experience for session tags with their identity solutions. To learn how to use these identity providers to configure session tags, see [Integrating third-party SAML solution providers with AWS \(p. 197\)](#).

To pass SAML attributes as session tags, include the `Attribute` element with the `Name` attribute set to `https://aws.amazon.com/SAML/Attributes/PrincipalTag:{TagKey}`. Use the `AttributeValue` element to specify the value of the tag. Include a separate `Attribute` element for each session tag.

For example, assume that you want to pass the following identity attributes as session tags:

- `Project:Automation`
- `CostCenter:12345`
- `Department:Engineering`

To pass these attributes, include the following elements in your SAML assertion.

Example Example snippet of a SAML assertion

```
<Attribute Name="https://aws.amazon.com/SAML/Attributes/PrincipalTag:Project">
  <AttributeValue>Automation</AttributeValue>
</Attribute>
<Attribute Name="https://aws.amazon.com/SAML/Attributes/PrincipalTag:CostCenter">
  <AttributeValue>12345</AttributeValue>
</Attribute>
<Attribute Name="https://aws.amazon.com/SAML/Attributes/PrincipalTag:Department">
  <AttributeValue>Engineering</AttributeValue>
</Attribute>
```

To set the tags above as transitive, include another `Attribute` element with the `Name` attribute set to `https://aws.amazon.com/SAML/Attributes/TransitiveTagKeys`. Transitive tags persist during role chaining. For more information, see [Chaining roles with session tags \(p. 299\)](#).

To set the `Project` and `Department` tags as transitive, use the following multivalued attribute.

Example Example snippet of a SAML assertion

```
<Attribute Name="https://aws.amazon.com/SAML/Attributes/TransitiveTagKeys">
  <AttributeValue>Project</AttributeValue>
  <AttributeValue>Department</AttributeValue>
</Attribute>
```

Passing session tags using AssumeRoleWithWebIdentity

The `AssumeRoleWithWebIdentity` operation is authenticated using OpenID Connect (OIDC)-compliant web identity federation. This operation returns a set of temporary credentials that you can use to access AWS resources. For more information about using web identity federation for AWS Management Console access, see [About web identity federation \(p. 177\)](#).

To pass session tags from OpenID Connect (OIDC), you must include the session tags in the JSON Web Token (JWT). Include session tags in the `https://aws.amazon.com/tags` namespace in the token when you submit the `AssumeRoleWithWebIdentity` request. To learn more about OIDC tokens and claims, see [Using Tokens with User Pools](#) in the *Amazon Cognito Developer Guide*.

For example, the following decoded JWT is a token that is used to call `AssumeRoleWithWebIdentity` with the `Project`, `CostCenter`, and `Department` session tags. The token also sets the `Project` and `CostCenter` tags as transitive. Transitive tags persist during role chaining. For more information, see [Chaining roles with session tags \(p. 299\)](#).

Example Example decoded JSON web token

```
{  
    "sub": "johndoe",  
    "aud": "ac_oic_client",  
    "jti": "ZYUCeRMOVtqHypVPWAN3VB",  
    "iss": "https://xyz.com",  
    "iat": 1566583294,  
    "exp": 1566583354,  
    "auth_time": 1566583292,  
    "https://aws.amazon.com/tags": {  
        "principal_tags": {  
            "Project": ["Automation"],  
            "CostCenter": ["987654"],  
            "Department": ["Engineering"]  
        },  
        "transitive_tag_keys": [  
            "Project",  
            "CostCenter"  
        ]  
    }  
}
```

Passing session tags using GetFederationToken

The `GetFederationToken` lets you federate your user. This operation returns a set of temporary credentials that you can use to access AWS resources. To add tags to your federated user session, use the `--tags` AWS CLI option or the `Tags` AWS API parameter. You can't set session tags as transitive when you use `GetFederationToken`. This is because you can't use the temporary credentials to assume a role, which means that role chaining is not possible.

The following example shows a sample request using `GetFederationToken`. In this example, when you request the token, you create a session named `my-fed-user`. You add the session tag key-value pairs `Project = Automation` and `Department = Engineering`.

Example Example GetFederationToken CLI request

```
aws sts get-federation-token \  
--name my-fed-user \  
--tags key=Project,value=Automation key=Department,value=Engineering
```

When you use the temporary credentials that are returned by the `GetFederationToken` operation, the session's principal tags include the user's tags and the passed session tags.

Chaining roles with session tags

You can assume one role and then use the temporary credentials to assume another role. You can continue from session to session. This is called [role chaining \(p. 169\)](#). When you pass session tags while assuming a role, you can set the keys as transitive. This ensures that those session tags pass to subsequent sessions in a role chain. You cannot set role tags as transitive. To pass these tags to subsequent sessions, specify them as session tags.

The following example will help you understand how session tags, transitive tags, and role tags are passed into subsequent sessions in a role chain.

In the following example role chaining scenario, you use an IAM user's access keys in the AWS CLI to assume a role named `Role1`. You then use the resulting session credentials to assume a second role named `Role2`. You can then use the second session credentials to assume a third role named `Role3`. These requests occur as three separate operations. Each role is already tagged in IAM. And during each request, you pass additional session tags.

When you chain roles, you can ensure that tags from an earlier session persist to the later sessions. To do this using the `assume-role` CLI command, you must pass the tag as a session tag and set the tag as transitive. You pass the tag `Star = 1` as a session tag. The tag `Heart = 1` is attached to the role and will apply as a principal tag when you use the session. However, you also want the `Heart = 1` tag to automatically pass to the second or third session. To do that, you manually include it as a session tag. That way the resulting session's principal tags include these two tags, and they are set as transitive.

You perform this request using the following AWS CLI command:

Example Example AssumeRole CLI request

```
aws sts assume-role \
--role-arn arn:aws:iam::123456789012:role/Role1 \
--role-session-name Session1 \
--tags Key=Star,Value=1 Key=Heart,Value=1 \
--transitive-tag-keys Star Heart
```

You then use the credentials for that session to assume `Role2`. The tag `Sun = 2` is attached to the second role and will apply as a principal tag when you use the second session. The `Heart` and `Star` tags are inherited from the transitive session tags in the first session. The second session's resulting principal tags are `Heart = 1`, `Star = 1`, and `Sun = 2`. `Heart` and `Star` will continue to be transitive. The `Sun` tag that was attached to `Role2` is not marked as transitive because it is not a session tag. This tag will not be inherited by future sessions.

You perform this second request using the following AWS CLI command:

Example Example AssumeRole CLI request

```
aws sts assume-role \
--role-arn arn:aws:iam::123456789012:role/Role2 \
--role-session-name Session2
```

You then use the second session credentials to assume `Role3`. The principal tags for the third session come from any new session tags, the inherited transitive session tags, and the role tags. The `Heart = 1` and `Star = 1` tags on the second session were inherited from the transitive session tag in the first session. If you try to pass the `Heart = 3` session tag, the operation will fail. The inherited `Star = 1` session tag overrides the role's `Star = 3` tag. The role's `Lightning` tag also applies to the third session, and is not set as transitive.

You perform the third request using the following AWS CLI command:

Example Example AssumeRole CLI request

```
aws sts assume-role \
--role-arn arn:aws:iam::123456789012:role/Role3 \
--role-session-name Session3
```

Using session tags for ABAC

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on tag attributes.

If your company uses a SAML-based identity provider (IdP) for corporate user identities, you can configure your SAML assertion to pass session tags to AWS. When your employees federate into AWS, their attributes are applied to their resulting principal in AWS. You can then use ABAC to allow or deny permissions based on those attributes. For details, see [IAM Tutorial: Use SAML session tags for ABAC \(p. 57\)](#).

For more information about using AWS SSO with ABAC, see [Attributes for access control](#) in the [AWS Single Sign-On User Guide](#).

Viewing session tags in CloudTrail

You can use AWS CloudTrail to view the requests made to assume roles or federate users. The CloudTrail log file includes information about the principal tags for the assumed-role or federated user session. For more information, see [Logging IAM and AWS STS API calls with AWS CloudTrail \(p. 336\)](#).

For example, assume that you make an AWS STS `AssumeRoleWithSAML` request, pass session tags, and set those tags as transitive. You can find the following information in your CloudTrail log.

Example Example AssumeRoleWithSAML CloudTrail log

```
"requestParameters": {  
    "SAMLAssertionID": "_c0046cEXAMPLEb9d4b8eEXAMPLE2619aEXAMPLE",  
    "roleSessionName": "MyRoleSessionName",  
    "principalTags": {  
        "CostCenter": "987654",  
        "Project": "Unicorn"  
    },  
    "transitiveTagKeys": [  
        "CostCenter",  
        "Project"  
    ],  
    "durationSeconds": 3600,  
    "roleArn": "arn:aws:iam::123456789012:role/SAMLTestRoleShibboleth",  
    "principalArn": "arn:aws:iam::123456789012:saml-provider/Shibboleth"  
},
```

You can view the following example CloudTrail logs to view events that use session tags.

- [Example AWS STS role chaining API event in CloudTrail log file \(p. 343\)](#)
- [Example SAML AWS STS API event in CloudTrail log file \(p. 345\)](#)
- [Example web identity AWS STS API event in CloudTrail log file \(p. 347\)](#)

Temporary security credentials in IAM

You can use the AWS Security Token Service (AWS STS) to create and provide trusted users with temporary security credentials that can control access to your AWS resources. Temporary security credentials work almost identically to the long-term access key credentials that your IAM users can use, with the following differences:

- Temporary security credentials are *short-term*, as the name implies. They can be configured to last for anywhere from a few minutes to several hours. After the credentials expire, AWS no longer recognizes them or allows any kind of access from API requests made with them.
- Temporary security credentials are not stored with the user but are generated dynamically and provided to the user when requested. When (or even before) the temporary security credentials expire, the user can request new credentials, as long as the user requesting them still has permissions to do so.

These differences lead to the following advantages for using temporary credentials:

- You do not have to distribute or embed long-term AWS security credentials with an application.
- You can provide access to your AWS resources to users without having to define an AWS identity for them. Temporary credentials are the basis for [roles and identity federation \(p. 167\)](#).

- The temporary security credentials have a limited lifetime, so you do not have to rotate them or explicitly revoke them when they're no longer needed. After temporary security credentials expire, they cannot be reused. You can specify how long the credentials are valid, up to a maximum limit.

AWS STS and AWS regions

Temporary security credentials are generated by AWS STS. By default, AWS STS is a global service with a single endpoint at <https://sts.amazonaws.com>. However, you can also choose to make AWS STS API calls to endpoints in any other supported Region. This can reduce latency (server lag) by sending the requests to servers in a Region that is geographically closer to you. No matter which Region your credentials come from, they work globally. For more information, see [Managing AWS STS in an AWS Region \(p. 327\)](#).

Common scenarios for temporary credentials

Temporary credentials are useful in scenarios that involve identity federation, delegation, cross-account access, and IAM roles.

Identity federation

You can manage your user identities in an external system outside of AWS and grant users who sign in from those systems access to perform AWS tasks and access your AWS resources. IAM supports two types of identity federation. In both cases, the identities are stored outside of AWS. The distinction is where the external system resides—in your data center or an external third party on the web. For more information about external identity providers, see [Identity providers and federation \(p. 176\)](#).

- **Enterprise identity federation** – You can authenticate users in your organization's network, and then provide those users access to AWS without creating new AWS identities for them and requiring them to sign in with a separate user name and password. This is known as the *single sign-on* (SSO) approach to temporary access. AWS STS supports open standards like Security Assertion Markup Language (SAML) 2.0, with which you can use Microsoft AD FS to leverage your Microsoft Active Directory. You can also use SAML 2.0 to manage your own solution for federating user identities. For more information, see [About SAML 2.0-based federation \(p. 182\)](#).
- **Custom federation broker** – You can use your organization's authentication system to grant access to AWS resources. For an example scenario, see [Enabling custom identity broker access to the AWS console \(p. 206\)](#).
- **Federation using SAML 2.0** – You can use your organization's authentication system and SAML to grant access to AWS resources. For more information and an example scenario, see [About SAML 2.0-based federation \(p. 182\)](#).
- **Web identity federation** – You can let users sign in using a well-known third party identity provider such as Login with Amazon, Facebook, Google, or any OpenID Connect (OIDC) 2.0 compatible provider. You can exchange the credentials from that provider for temporary permissions to use resources in your AWS account. This is known as the *web identity federation* approach to temporary access. When you use web identity federation for your mobile or web application, you don't need to create custom sign-in code or manage your own user identities. Using web identity federation helps you keep your AWS account secure, because you don't have to distribute long-term security credentials, such as IAM user access keys, with your application. For more information, see [About web identity federation \(p. 177\)](#).

AWS STS web identity federation supports Login with Amazon, Facebook, Google, and any OpenID Connect (OIDC)-compatible identity provider.

Note

For mobile applications, we recommend that you use Amazon Cognito. You can use this service with the [AWS Mobile SDK for iOS](#) and the [AWS Mobile SDK for Android and Fire](#)

OS to create unique identities for users and authenticate them for secure access to your AWS resources. Amazon Cognito supports the same identity providers as AWS STS, and also supports unauthenticated (guest) access and lets you migrate user data when a user signs in. Amazon Cognito also provides API operations for synchronizing user data so that it is preserved as users move between devices. For more information, see the following:

- [Amazon Cognito Identity in the AWS Mobile SDK for iOS Developer Guide](#)
- [Amazon Cognito Identity in the AWS Mobile SDK for Android Developer Guide](#)

Roles for cross-account access

Many organizations maintain more than one AWS account. Using roles and cross-account access, you can define user identities in one account, and use those identities to access AWS resources in other accounts that belong to your organization. This is known as the *delegation* approach to temporary access. For more information about creating cross-account roles, see [Creating a role to delegate permissions to an IAM user \(p. 221\)](#). To learn whether principals in accounts outside of your zone of trust (trusted organization or account) have access to assume your roles, see [What is IAM Access Analyzer?](#).

Roles for Amazon EC2

If you run applications on Amazon EC2 instances and those applications need access to AWS resources, you can provide temporary security credentials to your instances when you launch them. These temporary security credentials are available to all applications that run on the instance, so you don't need to store any long-term credentials on the instance. For more information, see [Using an IAM role to grant permissions to applications running on Amazon EC2 instances \(p. 263\)](#).

Other AWS services

You can use temporary security credentials to access most AWS services. For a list of the services that accept temporary security credentials, see [AWS services that work with IAM \(p. 611\)](#).

Requesting temporary security credentials

To request temporary security credentials, you can use AWS Security Token Service (AWS STS) operations in the AWS API. These include operations to create and provide trusted users with temporary security credentials that can control access to your AWS resources. For more information about AWS STS, see [Temporary security credentials in IAM \(p. 301\)](#). To learn about the different methods that you can use to request temporary security credentials by assuming a role, see [Using IAM roles \(p. 246\)](#).

To call the API operations, you can use one of the [AWS SDKs](#). The SDKs are available for a variety of programming languages and environments, including Java, .NET, Python, Ruby, Android, and iOS. The SDKs take care of tasks such as cryptographically signing your requests, retrying requests if necessary, and handling error responses. You can also use the AWS STS Query API, which is described in the [AWS Security Token Service API Reference](#). Finally, two command line tools support the AWS STS commands: the [AWS Command Line Interface](#), and the [AWS Tools for Windows PowerShell](#).

The AWS STS API operations create a new session with temporary security credentials that include an access key pair and a session token. The access key pair consists of an access key ID and a secret key. Users (or an application that the user runs) can use these credentials to access your resources. You can create a role session and pass session policies and session tags programmatically using AWS STS API operations. The resulting session's permissions are the intersection of the role's identity-based policies and the session policies. For more information about session policies, see [Session policies \(p. 353\)](#). For more information about session tags, see [Passing session tags in AWS STS \(p. 293\)](#).

Note

The size of the security token that AWS STS API operations return is not fixed. We strongly recommend that you make no assumptions about the maximum size. The typical token size is less than 4096 bytes, but that can vary.

Using AWS STS with AWS regions

You can send AWS STS API calls either to a global endpoint or to one of the Regional endpoints. If you choose an endpoint closer to you, you can reduce latency and improve the performance of your API calls. You also can choose to direct your calls to an alternative Regional endpoint if you can no longer communicate with the original endpoint. If you are using one of the various AWS SDKs, then use that SDK's method to select a Region before you make the API call. If you are manually constructing HTTP API requests, then you must direct the request to the correct endpoint yourself. For more information, see the [AWS STS section of Regions and Endpoints](#) and [Managing AWS STS in an AWS Region \(p. 327\)](#).

The following are the API operations that you can use to acquire temporary credentials for use in your AWS environment and applications.

AssumeRole—cross-account delegation and federation through a custom identity broker

The `AssumeRole` API operation is useful for allowing existing IAM users to access AWS resources that they don't already have access to. For example, the user might need access to resources in another AWS account. It is also useful as a means to temporarily gain privileged access—for example, to provide multi-factor authentication (MFA). You must call this API using existing IAM user credentials. For more information, see [Creating a role to delegate permissions to an IAM user \(p. 221\)](#) and [Configuring MFA-protected API access \(p. 138\)](#).

This call must be made using valid AWS security credentials. When you make this call, you pass the following information:

- The Amazon Resource Name (ARN) of the role that the app should assume.
- (Optional) Duration, which specifies the duration of the temporary security credentials. Use the `DurationSeconds` parameter to specify the duration of the role session from 900 seconds (15 minutes) up to the maximum session duration setting for the role. To learn how to view the maximum value for your role, see [View the maximum session duration setting for a role \(p. 248\)](#). If you do not pass this parameter, the temporary credentials expire in one hour. The `DurationSeconds` parameter from this API is separate from the `SessionDuration` HTTP parameter that you use to specify the duration of a console session. Use the `SessionDuration` HTTP parameter in the request to the federation endpoint for a console sign-in token. For more information, see [Enabling custom identity broker access to the AWS console \(p. 206\)](#).
- Role session name, which is a string value that you can use to identify the session. For security purposes, administrators can view this field in [AWS CloudTrail logs \(p. 340\)](#) to learn who performed an action in AWS. Your administrator might require that you specify your IAM user name as the session name when you assume the role. For more information, see [aws:RoleSessionName \(p. 718\)](#).
- (Optional) Inline or managed session policies. These policies limit the permissions from the role's identity-based policy that are assigned to the role session. The resulting session's permissions are the intersection of the role's identity-based policies and the session policies. Session policies cannot be used to grant more permissions than those allowed by the identity-based policy of the role that is being assumed. For more information about role session permissions, see [Session policies \(p. 353\)](#).
- (Optional) Session tags. You can assume a role and then use the temporary credentials to make a request. When you do, the session's principal tags include the role's tags and the passed session tags. If you make this call using temporary credentials, the new session also inherits transitive session tags from the calling session. For more information about session tags, see [Passing session tags in AWS STS \(p. 293\)](#).
- (Optional) MFA information. If configured to use multi-factor authentication (MFA), then you include the identifier for an MFA device and the one-time code provided by that device.
- (Optional) `ExternalId` value that can be used when delegating access to your account to a third party. This value helps ensure that only the specified third party can access the role. For more

information, see [How to use an external ID when granting access to your AWS resources to a third party \(p. 225\)](#).

The following example shows a sample request and response using `AssumeRole`. This example request assumes the `demo` role for the specified duration with the included [session policy \(p. 353\)](#), [session tags \(p. 293\)](#), and [external ID \(p. 225\)](#). The resulting session is named `John-session`.

Example Example request

```
https://sts.amazonaws.com/  
?Version=2011-06-15  
&Action=AssumeRole  
&RoleSessionName=John-session  
&RoleArn=arn:aws::iam::123456789012:role/demo  
&Policy=%7B%22Version%22%3A%222012-10-17%22%2C%22Statement%22%3A%5B%7B%22Sid%22%3A  
%20%22Stmt1%22%2C%22Effect%22%3A%20%22Allow%22%2C%22Action%22%3A%20%22s3%3A*%22%2C  
%22Resource%22%3A%20%22*s3%22%7D%5D%7D  
&DurationSeconds=1800  
&Tags.member.1.Key=Project  
&Tags.member.1.Value=Pegasus  
&Tags.member.2.Key=Cost-Center  
&Tags.member.2.Value=12345  
&ExternalId=123ABC  
&AUTHPARAMS
```

The policy value shown in the preceding example is the URL-encoded version of the following policy:

```
{"Version": "2012-10-17", "Statement":  
[{"Sid": "Stmt1", "Effect": "Allow", "Action": "s3:*", "Resource": "*"}]}
```

The `AUTHPARAMS` parameter in the example is a placeholder for your *signature*. A signature is the authentication information that you must include with AWS HTTP API requests. We recommend using the [AWS SDKs](#) to create API requests, and one benefit of doing so is that the SDKs handle request signing for you. If you must create and sign API requests manually, see [Signing AWS Requests By Using Signature Version 4](#) in the [Amazon Web Services General Reference](#) to learn how to sign a request.

In addition to the temporary security credentials, the response includes the Amazon Resource Name (ARN) for the federated user and the expiration time of the credentials.

Example Example response

```
<AssumeRoleResponse xmlns="https://sts.amazonaws.com/doc/2011-06-15/">  
<AssumeRoleResult>  
<Credentials>  
<SessionToken>  
AQoDYXdzEPT//////////wEXAMPLEtc764bNrC9SAPBSM22wD0k4x4HIZ8j4FZTwdQW  
LWsKWHGBuFgwAeMicRXmxfpSPFleoiYRqTflfKD8YuuthAx7mSEI/qkPpKPi/kMcGd  
QrmGdeehM4IC1NtBmUpp2wUE8phUZampKsburEDy0KPkYQDYwt7WZ0wq5VSxDvp75YU  
9HFv1Rd8Tx6q6fE8YQcHNvXAkiv9q6d+xo0rKwT38xVqr7ZD0u0iPPkUL64lIZbqBAz  
+scqKmlz8FDrypNC9Yjc8fPOLn9FX9KSYvKTr4rvx3iSiLTJabiQwj2ICCR/oLxBA==  
</SessionToken>  
<SecretAccessKey>  
wJalrXUtnFEMI/K7MDENG/bPxRfiCYzEXAMPLEKEY  
</SecretAccessKey>  
<Expiration>2019-07-15T23:28:33.359Z</Expiration>  
<AccessKeyId>AKIAIOSFODNN7EXAMPLE</AccessKeyId>  
</Credentials>  
<AssumedRoleUser>  
<Arn>arn:aws:sts::123456789012:assumed-role/demo/John</Arn>  
<AssumedRoleId>ARO123EXAMPLE123:John</AssumedRoleId>  
</AssumedRoleUser>
```

```
<PackedPolicySize>8</PackedPolicySize>
</AssumeRoleResult>
<ResponseMetadata>
<RequestId>c6104cbe-af31-11e0-8154-cbc7ccf896c7</RequestId>
</ResponseMetadata>
</AssumeRoleResponse>
```

Note

An AWS conversion compresses the passed session policies and session tags into a packed binary format that has a separate limit. Your request can fail for this limit even if your plain text meets the other requirements. The `PackedPolicySize` response element indicates by percentage how close the policies and tags for your request are to the upper size limit.

AssumeRoleWithWebIdentity—federation through a web-based identity provider

The `AssumeRoleWithWebIdentity` API operation returns a set of temporary security credentials for federated users who are authenticated through a public identity provider. Examples of public identity providers include Login with Amazon, Facebook, Google, or any OpenID Connect (OIDC)-compatible identity provider. This operation is useful for creating mobile applications or client-based web applications that require access to AWS. Using this operation means that your users do not need their own AWS or IAM identities. For more information, see [About web identity federation \(p. 177\)](#).

Instead of directly calling `AssumeRoleWithWebIdentity`, we recommend that you use Amazon Cognito and the Amazon Cognito credentials provider with the AWS SDKs for mobile development. For more information, see the following:

- [Amazon Cognito Identity](#) in the *AWS Mobile SDK for Android Developer Guide*
- [Amazon Cognito Identity](#) in the *AWS Mobile SDK for iOS Developer Guide*

If you are not using Amazon Cognito, you call the `AssumeRoleWithWebIdentity` action of AWS STS. This is an unsigned call, meaning that the app does not need to have access to any AWS security credentials to make the call. When you make this call, you pass the following information:

- The Amazon Resource Name (ARN) of the role that the app should assume. If your app supports multiple ways for users to sign in, you must define multiple roles, one per identity provider. The call to `AssumeRoleWithWebIdentity` should include the ARN of the role that is specific to the provider through which the user signed in.
- The token that the app gets from the IdP after the app authenticates the user.
- You can configure your IdP to pass attributes into your token as [session tags \(p. 293\)](#).
- (Optional) Duration, which specifies the duration of the temporary security credentials. Use the `DurationSeconds` parameter to specify the duration of the role session from 900 seconds (15 minutes) up to the maximum session duration setting for the role. To learn how to view the maximum value for your role, see [View the maximum session duration setting for a role \(p. 248\)](#). If you do not pass this parameter, the temporary credentials expire in one hour. The `DurationSeconds` parameter from this API is separate from the `SessionDuration` HTTP parameter that you use to specify the duration of a console session. Use the `SessionDuration` HTTP parameter in the request to the federation endpoint for a console sign-in token. For more information, see [Enabling custom identity broker access to the AWS console \(p. 206\)](#).
- Role session name, which is a string value that you can use to identify the session. For security purposes, administrators can view this field in [AWS CloudTrail logs \(p. 340\)](#) to learn who performed an action in AWS. Your administrator might require that you provide a specific value for the session name when you assume the role. For more information, see [aws:RoleSessionName \(p. 718\)](#).
- (Optional) Inline or managed session policies. These policies limit the permissions from the role's identity-based policy that are assigned to the role session. The resulting session's permissions are the

intersection of the role's identity-based policies and the session policies. Session policies cannot be used to grant more permissions than those allowed by the identity-based policy of the role that is being assumed. For more information about role session permissions, see [Session policies \(p. 353\)](#).

Note

A call to `AssumeRoleWithWebIdentity` is not signed (encrypted). Therefore, you should only include optional session policies if the request is transmitted through a trusted intermediary. In this case, someone could alter the policy to remove the restrictions.

When you call `AssumeRoleWithWebIdentity`, AWS verifies the authenticity of the token. For example, depending on the provider, AWS might make a call to the provider and include the token that the app has passed. Assuming that the identity provider validates the token, AWS returns the following information to you:

- A set of temporary security credentials. These consist of an access key ID, a secret access key, and a session token.
- The role ID and the ARN of the assumed role.
- A `SubjectFromWebIdentityToken` value that contains the unique user ID.

When you have the temporary security credentials, you can use them to make AWS API calls. This is the same process as making an AWS API call with long-term security credentials. The difference is that you must include the session token, which lets AWS verify that the temporary security credentials are valid.

Your app should cache the credentials. As noted, by default the credentials expire after an hour. If you are not using the `AmazonSTSCredentialsProvider` operation in the AWS SDK, it's up to you and your app to call `AssumeRoleWithWebIdentity` again. Call this operation to get a new set of temporary security credentials before the old ones expire.

AssumeRoleWithSAML—federation through an enterprise Identity Provider compatible with SAML 2.0

The `AssumeRoleWithSAML` API operation returns a set of temporary security credentials for federated users who are authenticated by your organization's existing identity system. The users must also use [SAML 2.0 \(Security Assertion Markup Language\)](#) to pass authentication and authorization information to AWS. This API operation is useful in organizations that have integrated their identity systems (such as Windows Active Directory or OpenLDAP) with software that can produce SAML assertions. Such an integration provides information about user identity and permissions (such as Active Directory Federation Services or Shibboleth). For more information, see [About SAML 2.0-based federation \(p. 182\)](#).

This is an unsigned call, which means that the app does not need to have access to any AWS security credentials in order to make the call. When you make this call, you pass the following information:

- The Amazon Resource Name (ARN) of the role that the app should assume.
- The ARN of the SAML provider created in IAM that describes the identity provider.
- The SAML assertion, encoded in base64, that was provided by the SAML identity provider in its authentication response to the sign-in request from your app.
- You can configure your IdP to pass attributes into your SAML assertion as [session tags \(p. 293\)](#).
- (Optional) Duration, which specifies the duration of the temporary security credentials. Use the `DurationSeconds` parameter to specify the duration of the role session from 900 seconds (15 minutes) up to the maximum session duration setting for the role. To learn how to view the maximum value for your role, see [View the maximum session duration setting for a role \(p. 248\)](#). If you do not pass this parameter, the temporary credentials expire in one hour. The `DurationSeconds` parameter from this API is separate from the `SessionDuration` HTTP parameter that you use to specify the

duration of a console session. Use the `SessionDuration` HTTP parameter in the request to the federation endpoint for a console sign-in token. For more information, see [Enabling custom identity broker access to the AWS console \(p. 206\)](#).

- (Optional) Inline or managed session policies. These policies limit the permissions from the role's identity-based policy that are assigned to the role session. The resulting session's permissions are the intersection of the role's identity-based policies and the session policies. Session policies cannot be used to grant more permissions than those allowed by the identity-based policy of the role that is being assumed. For more information about role session permissions, see [Session policies \(p. 353\)](#).

When you call `AssumeRoleWithSAML`, AWS verifies the authenticity of the SAML assertion. Assuming that the identity provider validates the assertion, AWS returns the following information to you:

- A set of temporary security credentials. These consist of an access key ID, a secret access key, and a session token.
- The role ID and the ARN of the assumed role.
- An `Audience` value that contains the value of the `Recipient` attribute of the `SubjectConfirmationData` element of the SAML assertion.
- An `Issuer` value that contains the value of the `Issuer` element of the SAML assertion.
- A `NameQualifier` element that contains a hash value built from the `Issuer` value, the AWS account ID, and the friendly name of the SAML provider. When combined with the `Subject` element, they can uniquely identify the federated user.
- A `Subject` element that contains the value of the `NameID` element in the `Subject` element of the SAML assertion.
- A `SubjectType` element that indicates the format of the `Subject` element. The value can be `persistent`, `transient`, or the full `Format` URI from the `Subject` and `NameID` elements used in your SAML assertion. For information about the `NameID` element's `Format` attribute, see [Configuring SAML assertions for the authentication response \(p. 199\)](#).

When you have the temporary security credentials, you can use them to make AWS API calls. This is the same process as making an AWS API call with long-term security credentials. The difference is that you must include the session token, which lets AWS verify that the temporary security credentials are valid.

Your app should cache the credentials. By default the credentials expire after an hour. If you are not using the `AmazonSTSCredentialsProvider` action in the AWS SDK, it's up to you and your app to call `AssumeRoleWithSAML` again. Call this operation to get a new set of temporary security credentials before the old ones expire.

GetFederationToken—federation through a custom identity broker

The `GetFederationToken` API operation returns a set of temporary security credentials for federated users. This API differs from `AssumeRole` in that the default expiration period is substantially longer (12 hours instead of one hour). Additionally, you can use the `DurationSeconds` parameter to specify a duration for the temporary security credentials to remain valid. The resulting credentials are valid for the specified duration, between 900 seconds (15 minutes) to 129,600 seconds (36 hours). The longer expiration period can help reduce the number of calls to AWS because you do not need to get new credentials as often. For more information, see [Requesting temporary security credentials \(p. 303\)](#).

When you make this request, you use the credentials of a specific IAM user. The permissions for the temporary security credentials are determined by the session policies that you pass when you call `GetFederationToken`. The resulting session permissions are the intersection of the IAM user policies and the session policies that you pass. Session policies cannot be used to grant more permissions than those allowed by the identity-based policy of the IAM user that is requesting federation. For more information about role session permissions, see [Session policies \(p. 353\)](#).

When you use the temporary credentials that are returned by the `GetFederationToken` operation, the session's principal tags include the user's tags and the passed session tags. For more information about session tags, see [Passing session tags in AWS STS \(p. 293\)](#).

The `GetFederationToken` call returns temporary security credentials that consist of the security token, access key, secret key, and expiration. You can use `GetFederationToken` if you want to manage permissions inside your organization (for example, using the proxy application to assign permissions). To view a sample application that uses `GetFederationToken`, go to [Identity Federation Sample Application for an Active Directory Use Case](#) in the *AWS Sample Code & Libraries*.

The following example shows a sample request and response that uses `GetFederationToken`. This example request federates the calling user for the specified duration with the [session policy \(p. 353\)](#) ARN and [session tags \(p. 293\)](#). The resulting session is named Jane-session.

Example Example request

```
https://sts.amazonaws.com/  
?Version=2011-06-15  
&Action=GetFederationToken  
&Name=Jane-session  
&PolicyArns.member.1.arn==arn%3Aaws%3Aiam%3A%3A123456789012%3Apolicy%2FRole1policy  
&DurationSeconds=1800  
&Tags.member.1.Key=Project  
&Tags.member.1.Value=Pegasus  
&Tags.member.2.Key=Cost-Center  
&Tags.member.2.Value=12345  
&AUTHPARAMS
```

The policy ARN shown in the preceding example includes the following URL-encoded ARN:

```
arn:aws:iam::123456789012:policy/Role1policy
```

Also, note that the `&AUTHPARAMS` parameter in the example is meant as a placeholder for the authentication information. This is the *signature*, which you must include with AWS HTTP API requests. We recommend using the [AWS SDKs](#) to create API requests, and one benefit of doing so is that the SDKs handle request signing for you. If you must create and sign API requests manually, go to [Signing AWS Requests By Using Signature Version 4](#) in the *Amazon Web Services General Reference* to learn how to sign a request.

In addition to the temporary security credentials, the response includes the Amazon Resource Name (ARN) for the federated user and the expiration time of the credentials.

Example Example response

```
<GetFederationTokenResponse xmlns="https://sts.amazonaws.com/doc/2011-06-15/">  
<GetFederationTokenResult>  
<Credentials>  
<SessionToken>  
AQoDYXdzEPT///////////wEXAMPLEtc764bNrC9SAPBSM22wD0k4x4HIZ8j4FZTwQW  
LWsKWHGBuFqwAeMicRXmxfpSPfleoIYRqTflfKD8YUuwthAx7mSEI/qkPpKPi/kMcGd  
QrmGdeehM4IC1NtBmUpp2wUE8phUZampKsburEDy0KPkyQDYwt7WZ0wq5VSxDvp75YU  
9HFv1Rd8Tx6q6fE8YQcHNVXAk1Y9q6d+xoOrKwT38xVqr7ZD0u0iPPkUL641IZbqBAz  
+scqKmlzm8FDrypNC9Yjc8fPOLn9FX9KSvKTr4rvx3iSi1TJabiQwj2ICCEEXAMPLE==  
</SessionToken>  
<SecretAccessKey>  
wJalrXUtnFEMI/K7MDENG/bPxRfiCYzEXAMPLEKEY  
</SecretAccessKey>  
<Expiration>2019-04-15T23:28:33.359Z</Expiration>  
<AccessKeyId>AKIAIOSFODNN7EXAMPLE;</AccessKeyId>  
</Credentials>  
<FederatedUser>
```

```
<Arn>arn:aws:sts::123456789012:federated-user/Jean</Arn>
<FederatedUserId>123456789012:Jean</FederatedUserId>
</FederatedUser>
<PackedPolicySize>4</PackedPolicySize>
</GetFederationTokenResult>
<ResponseMetadata>
<RequestId>c6104cbe-af31-11e0-8154-cbc7ccf896c7</RequestId>
</ResponseMetadata>
</GetFederationTokenResponse>
```

Note

An AWS conversion compresses the passed session policies and session tags into a packed binary format that has a separate limit. Your request can fail for this limit even if your plain text meets the other requirements. The `PackedPolicySize` response element indicates by percentage how close the policies and tags for your request are to the upper size limit.

AWS recommends that you grant permissions at the resource level (for example, you attach a resource-based policy to an Amazon S3 bucket), you can omit the `Policy` parameter. However, if you do not include a policy for the federated user, the temporary security credentials will not grant any permissions. In this case, you *must* use resource policies to grant the federated user access to your AWS resources.

For example, assume your AWS account number is 111122223333, and you have an Amazon S3 bucket that you want to allow Susan to access. Susan's temporary security credentials don't include a policy for the bucket. In that case, you would need to ensure that the bucket has a policy with an ARN that matches Susan's ARN, such as `arn:aws:sts::111122223333:federated-user/Susan`.

GetSessionToken—temporary credentials for users in untrusted environments

The `GetSessionToken` API operation returns a set of temporary security credentials to an existing IAM user. This is useful for providing enhanced security, such as allowing AWS requests only when MFA is enabled for the IAM user. Because the credentials are temporary, they provide enhanced security when you have an IAM user who accesses your resources through a less secure environment. Examples of less secure environments include a mobile device or web browser. For more information, see [Requesting temporary security credentials \(p. 303\)](#) or [GetSessionToken](#) in the *AWS Security Token Service API Reference*.

By default, temporary security credentials for an IAM user are valid for a maximum of 12 hours. But you can request a duration as short as 15 minutes or as long as 36 hours using the `DurationSeconds` parameter. For security reasons, a token for an AWS account root user is restricted to a duration of one hour.

`GetSessionToken` returns temporary security credentials consisting of a security token, an access key ID, and a secret access key. The following example shows a sample request and response using `GetSessionToken`. The response also includes the expiration time of the temporary security credentials.

Example Example request

```
https://sts.amazonaws.com/
?Version=2011-06-15
&Action=GetSessionToken
&DurationSeconds=1800
&AUTHPARAMS
```

The `AUTHPARAMS` parameter in the example is a placeholder for your *signature*. A signature is the authentication information that you must include with AWS HTTP API requests. We recommend using the [AWS SDKs](#) to create API requests, and one benefit of doing so is that the SDKs handle request

signing for you. If you must create and sign API requests manually, go to [Signing AWS Requests By Using Signature Version 4](#) in the *Amazon Web Services General Reference* to learn how to sign a request.

Example Example response

```
<GetSessionTokenResponse xmlns="https://sts.amazonaws.com/doc/2011-06-15/">
<GetSessionTokenResult>
<Credentials>
<SessionToken>
AQoEXAMPLEH4aoAH0gNCAPyJxz4B1CFFxWNE1OPTgk5TthT+FvwqnKwRcOIfrrRh3c/L
To6UDdyJwOvEVPPVlXCr0rUtdnniCEXAMPLE/IvU1dYUg2RVAJBanLiHb4IgRmpRV3z
rkuWJOgQs8IZZaIv2BXIa2R4OlglgkBN9bkUDNCJiBeb/Ax1zBBko7b15fjrBs2+cTQtp
Z3CYWFVG8C5zqx37wnOE49mR1/+OtKIKGO7fAE
</SessionToken>
<SecretAccessKey>
wJalrXUtnFEMI/K7MDENG/bPxRfiCYzEXAMPLEKEY
</SecretAccessKey>
<Expiration>2011-07-11T19:55:29.611Z</Expiration>
<AccessKeyId>AKIAIOSFODNN7EXAMPLE</AccessKeyId>
</Credentials>
</GetSessionTokenResult>
<ResponseMetadata>
<RequestId>58c5dbea-abef-11e0-8cfe-09039844ac7d</RequestId>
</ResponseMetadata>
</GetSessionTokenResponse>
```

Optionally, the `GetSessionToken` request can include `SerialNumber` and `TokenCode` values for AWS multi-factor authentication (MFA) verification. If the provided values are valid, AWS STS provides temporary security credentials that include the state of MFA authentication. The temporary security credentials can then be used to access the MFA-protected API operations or AWS websites for as long as the MFA authentication is valid.

The following example shows a `GetSessionToken` request that includes an MFA verification code and device serial number.

```
https://sts.amazonaws.com/
?Version=2011-06-15
&Action=GetSessionToken
&DurationSeconds=7200
&SerialNumber=YourMFADeviceSerialNumber
&TokenCode=123456
&AUTHPARAMS
```

Note

The call to AWS STS can be to the global endpoint or to any of the Regional endpoints that you activate your AWS account. For more information, see the [AWS STS section of Regions and Endpoints](#).

The `AUTHPARAMS` parameter in the example is a placeholder for your *signature*. A signature is the authentication information that you must include with AWS HTTP API requests. We recommend using the [AWS SDKs](#) to create API requests, and one benefit of doing so is that the SDKs handle request signing for you. If you must create and sign API requests manually, see [Signing AWS Requests By Using Signature Version 4](#) in the *Amazon Web Services General Reference* to learn how to sign a request.

Comparing the AWS STS API operations

The following table compares features of the API operations in AWS STS that return temporary security credentials. To learn about the different methods you can use to request temporary security credentials by assuming a role, see [Using IAM roles \(p. 246\)](#). To learn about the different AWS STS API operations that allow you to pass session tags, see [Passing session tags in AWS STS \(p. 293\)](#).

Comparing your API options

AWS STS API	Who can call	Credential lifetime (min max default)	MFA support ¹	Session policy support ²	Restrictions on resulting temporary credentials
AssumeRole	IAM user or IAM role with existing temporary security credentials	15 m Maximum session duration setting ³ 1 hr	Yes	Yes	Cannot call <code>GetFederationToken</code> or <code>GetSessionToken</code> .
AssumeRoleWithSAML	Any user caller must pass a SAML authentication response that indicates authentication from a known identity provider	15 m Maximum session duration setting ³ 1 hr	No	Yes	Cannot call <code>GetFederationToken</code> or <code>GetSessionToken</code> .
AssumeRoleWithWebIdentity	Any user caller must pass a web identity token that indicates authentication from a known identity provider	15 m Maximum session duration setting ³ 1 hr	No	Yes	Cannot call <code>GetFederationToken</code> or <code>GetSessionToken</code> .
GetFederationToken	IAM user or AWS account root user	IAM user: 15 m 36 hr 12 hr Root user: 15 m 1 hr 1 hr	No	Yes	Cannot call IAM operations using the AWS CLI or AWS API. Cannot call AWS STS operations except <code>GetCallerIdentity</code> . ⁴ SSO to console is allowed. ⁵
GetSessionToken	IAM user or AWS account root user	IAM user: 15 m 36 hr 12 hr Root user: 15 m 1 hr 1 hr	Yes	No	Cannot call IAM API operations unless MFA information is included with the request. Cannot call AWS STS API operations except <code>AssumeRole</code> or <code>GetCallerIdentity</code> . SSO to console is not allowed. ⁶

¹ **MFA support.** You can include information about a multi-factor authentication (MFA) device when you call the `AssumeRole` and `GetSessionToken` API operations. This ensures that the temporary security credentials that result from the API call can be used only by users who are authenticated with an MFA device. For more information, see [Configuring MFA-protected API access \(p. 138\)](#).

² **Session policy support.** Session policies are policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. This policy limits the permissions from the role or user's identity-based policy that are assigned to the session. The resulting

session's permissions are the intersection of the entity's identity-based policies and the session policies. Session policies cannot be used to grant more permissions than those allowed by the identity-based policy of the role that is being assumed. For more information about role session permissions, see [Session policies \(p. 353\)](#).

³ **Maximum session duration setting.** Use the `DurationSeconds` parameter to specify the duration of your role session from 900 seconds (15 minutes) up to the maximum session duration setting for the role. To learn how to view the maximum value for your role, see [View the maximum session duration setting for a role \(p. 248\)](#).

⁴ **GetCallerIdentity.** No permissions are required to perform this operation. If an administrator adds a policy to your IAM user or role that explicitly denies access to the `sts:GetCallerIdentity` action, you can still perform this operation. Permissions are not required because the same information is returned when an IAM user or role is denied access. To view an example response, see [I am not authorized to perform: iam>DeleteVirtualMFADevice \(p. 569\)](#).

⁵ **Single sign-on (SSO) to the console.** To support SSO, AWS lets you call a federation endpoint (<https://signin.aws.amazon.com/federation>) and pass temporary security credentials. The endpoint returns a token that you can use to construct a URL that signs a user directly into the console without requiring a password. For more information, see [Enabling SAML 2.0 federated users to access the AWS Management Console \(p. 203\)](#) and [How to Enable Cross-Account Access to the AWS Management Console](#) in the AWS Security Blog.

⁶ After you retrieve your temporary credentials, you can't access the AWS Management Console by passing the credentials to the federation single sign-on endpoint. For more information, see [Enabling custom identity broker access to the AWS console \(p. 206\)](#).

Using temporary credentials with AWS resources

You can use temporary security credentials to make programmatic requests for AWS resources using the AWS CLI or AWS API (using the [AWS SDKs](#)). The temporary credentials provide the same permissions that you have with use long-term security credentials such as IAM user credentials. However, there are a few differences:

- When you make a call using temporary security credentials, the call must include a session token, which is returned along with those temporary credentials. AWS uses the session token to validate the temporary security credentials.
- The temporary credentials expire after a specified interval. After the credentials expire, any calls that you make with those credentials will fail, so you must get a new set of credentials.
- When you use temporary credentials to make a request, your principal might include a set of tags. These tags come from session tags and tags that are attached to the role that you assume. For more information about session tags, see [Passing session tags in AWS STS \(p. 293\)](#).

If you are using the [AWS SDKs](#), the [AWS Command Line Interface](#) (AWS CLI), or the [Tools for Windows PowerShell](#), the way to get and use temporary security credentials differs with the context. If you are running code, AWS CLI, or Tools for Windows PowerShell commands inside an EC2 instance, you can take advantage of roles for Amazon EC2. Otherwise, you can call an [AWS STS API](#) to get the temporary credentials, and then use them explicitly to make calls to AWS services.

Note

You can use AWS Security Token Service (AWS STS) to create and provide trusted users with temporary security credentials that can control access to your AWS resources. For more information about AWS STS, see [Temporary security credentials in IAM \(p. 301\)](#). AWS STS is a global service that has a default endpoint at <https://sts.amazonaws.com>. This endpoint is in the US East (Ohio) Region, although credentials that you get from this and other endpoints are valid globally. These credentials work with services and resources in any Region. You can

also choose to make AWS STS API calls to endpoints in any of the supported Regions. This can reduce latency by making the requests from servers in a Region that is geographically closer to you. No matter which Region your credentials come from, they work globally. For more information, see [Managing AWS STS in an AWS Region \(p. 327\)](#).

Contents

- [Using temporary credentials in Amazon EC2 instances \(p. 314\)](#)
- [Using temporary security credentials with the AWS SDKs \(p. 314\)](#)
- [Using temporary security credentials with the AWS CLI \(p. 315\)](#)
- [Using temporary security credentials with API operations \(p. 315\)](#)
- [More information \(p. 316\)](#)

Using temporary credentials in Amazon EC2 instances

If you want to run AWS CLI commands or code inside an EC2 instance, the recommended way to get credentials is to use [roles for Amazon EC2](#). You create an IAM role that specifies the permissions that you want to grant to applications that run on the EC2 instances. When you launch the instance, you associate the role with the instance.

Applications, AWS CLI, and Tools for Windows PowerShell commands that run on the instance can then get automatic temporary security credentials from the instance metadata. You do not have to explicitly get the temporary security credentials. The AWS SDKs, AWS CLI, and Tools for Windows PowerShell automatically get the credentials from the EC2 instance metadata service and use them. The temporary credentials have the permissions that you define for the role that is associated with the instance.

For more information and for examples, see the following:

- [Using IAM Roles to Grant Access to AWS Resources on Amazon Elastic Compute Cloud](#) — AWS SDK for Java
- [Granting Access Using an IAM Role](#) — AWS SDK for .NET
- [Creating a Role](#) — AWS SDK for Ruby

Using temporary security credentials with the AWS SDKs

To use temporary security credentials in code, you programmatically call an AWS STS API like `AssumeRole` and extract the resulting credentials and session token. You then use those values as credentials for subsequent calls to AWS. The following example shows pseudocode for how to use temporary security credentials if you're using an AWS SDK:

```
assumeRoleResult = AssumeRole(role-arn);
tempCredentials = new SessionAWSCredentials(
    assumeRoleResult.AccessKeyId,
    assumeRoleResult.SecretAccessKey,
    assumeRoleResult.SessionToken);
s3Request = CreateAmazonS3Client(tempCredentials);
```

For an example written in Python (using the [AWS SDK for Python \(Boto\)](#)), see [Switching to an IAM role \(AWS API\) \(p. 262\)](#). This example shows how to call `AssumeRole` to get temporary security credentials and then use those credentials to make a call to Amazon S3.

For details about how to call `AssumeRole`, `GetFederationToken`, and other API operations, see the [AWS Security Token Service API Reference](#). For information on getting the temporary security credentials and session token from the result, see the documentation for the SDK that you're working with. You can

find the documentation for all the AWS SDKs on the main [AWS documentation page](#), in the **SDKs and Toolkits** section.

You must make sure that you get a new set of credentials before the old ones expire. In some SDKs, you can use a provider that manages the process of refreshing credentials for you; check the documentation for the SDK you're using.

Using temporary security credentials with the AWS CLI

You can use temporary security credentials with the AWS CLI. This can be useful for testing policies.

Using the [AWS CLI](#), you can call an [AWS STS API](#) like `AssumeRole` or `GetFederationToken` and then capture the resulting output. The following example shows a call to `AssumeRole` that sends the output to a file. In the example, the `profile` parameter is assumed to be a profile in the AWS CLI configuration file. It is also assumed to reference credentials for an IAM user who has permissions to assume the role.

```
aws sts assume-role --role-arn arn:aws:iam::123456789012:role/role-name --role-session-name "RoleSession1" --profile IAM-user-name > assume-role-output.txt
```

When the command is finished, you can extract the access key ID, secret access key, and session token from wherever you've routed it. You can do this either manually or by using a script. You can then assign these values to environment variables.

When you run AWS CLI commands, the AWS CLI looks for credentials in a specific order—first in environment variables and then in the configuration file. Therefore, after you've put the temporary credentials into environment variables, the AWS CLI uses those credentials by default. (If you specify a `profile` parameter in the command, the AWS CLI skips the environment variables. Instead, the AWS CLI looks in the configuration file, which lets you override the credentials in the environment variables if you need to.)

The following example shows how you might set the environment variables for temporary security credentials and then call an AWS CLI command. Because no `profile` parameter is included in the AWS CLI command, the AWS CLI looks for credentials first in environment variables and therefore uses the temporary credentials.

Linux

```
$ export AWS_ACCESS_KEY_ID=AKIAI44QH8DHBXAMPLE
$ export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
$ export AWS_SESSION_TOKEN=AQoDYXdzEJr...<remainder of security token>
$ aws ec2 describe-instances --region us-west-1
```

Windows

```
C:\> SET AWS_ACCESS_KEY_ID=AKIAI44QH8DHBXAMPLE
C:\> SET AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
C:\> SET AWS_SESSION_TOKEN=AQoDYXdzEJr...<remainder of token>
C:\> aws ec2 describe-instances --region us-west-1
```

Using temporary security credentials with API operations

If you're making direct HTTPS API requests to AWS, you can sign those requests with the temporary security credentials that you get from the AWS Security Token Service (AWS STS). To do this, you use the access key ID and secret access key that you receive from AWS STS. You use the access key ID and secret access key the same way you would use long-term credentials to sign a request. You also add to your API request the session token that you receive from AWS STS. You add the session token to an HTTP

header or to a query string parameter named `X-Amz-Security-Token`. You add the session token to the HTTP header *or* the query string parameter, but not both. For more information about signing HTTPS API requests, see [Signing AWS API Requests](#) in the [AWS General Reference](#).

More information

For more information about using AWS STS with other AWS services, see the following links:

- **Amazon S3.** See [Making Requests Using IAM User Temporary Credentials](#) or [Making Requests Using Federated User Temporary Credentials](#) in the *Amazon Simple Storage Service Developer Guide*.
- **Amazon SNS.** See [Using Temporary Security Credentials](#) in the *Amazon Simple Notification Service Developer Guide*.
- **Amazon SQS.** See [Using Temporary Security Credentials](#) in the *Amazon Simple Queue Service Developer Guide*.
- **Amazon SimpleDB.** See [Using Temporary Security Credentials](#) in the *Amazon SimpleDB Developer Guide*.

Controlling permissions for temporary security credentials

You can use AWS Security Token Service (AWS STS) to create and provide trusted users with temporary security credentials that can control access to your AWS resources. For more information about AWS STS, see [Temporary security credentials in IAM \(p. 301\)](#). After AWS STS issues temporary security credentials, they are valid through the expiration period and cannot be revoked. However, the permissions assigned to temporary security credentials are evaluated each time a request is made that uses the credentials, so you can achieve the effect of revoking the credentials by changing their access rights after they have been issued.

The following topics assume you have a working knowledge of AWS permissions and policies. For more information on these topics, see [Access management for AWS resources \(p. 350\)](#).

Topics

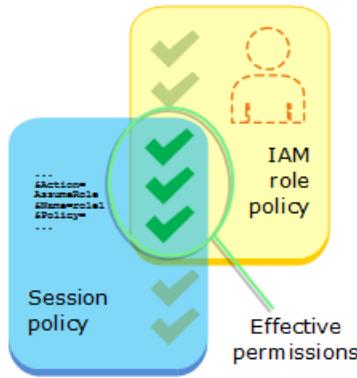
- [Permissions for AssumeRole, AssumeRoleWithSAML, and AssumeRoleWithWebIdentity \(p. 316\)](#)
- [Permissions for GetFederationToken \(p. 319\)](#)
- [Permissions for GetSessionToken \(p. 322\)](#)
- [Disabling permissions for temporary security credentials \(p. 323\)](#)
- [Granting permissions to create temporary security credentials \(p. 326\)](#)

Permissions for AssumeRole, AssumeRoleWithSAML, and AssumeRoleWithWebIdentity

The permissions policy of the role that is being assumed determines the permissions for the temporary security credentials that are returned by `AssumeRole`, `AssumeRoleWithSAML`, and `AssumeRoleWithWebIdentity`. You define these permissions when you create or update the role.

Optionally, you can pass inline or managed [session policies \(p. 353\)](#) as parameters of the `AssumeRole`, `AssumeRoleWithSAML`, or `AssumeRoleWithWebIdentity` API operations. Session policies limit the permissions for the role's temporary credential session. The resulting session's permissions are the intersection of the role's identity-based policy and the session policies. You can use the role's temporary credentials in subsequent AWS API calls to access resources in the account that owns the role. You cannot use session policies to grant more permissions than those allowed by the identity-based policy of the

role that is being assumed. To learn more about how AWS determines the effective permissions of a role, see [Policy evaluation logic \(p. 666\)](#).



The policies that are attached to the credentials that made the original call to `AssumeRole` are not evaluated by AWS when making the "allow" or "deny" authorization decision. The user temporarily gives up its original permissions in favor of the permissions assigned by the assumed role. In the case of the `AssumeRoleWithSAML` and `AssumeRoleWithWebIdentity` API operations, there are no policies to evaluate because the caller of the API is not an AWS identity.

Example: Assigning permissions using AssumeRole

You can use the `AssumeRole` API operation with different kinds of policies. Here are a few examples.

Role permissions policy

In this example, you call the `AssumeRole` API operation without specifying the session policy in the optional `Policy` parameter. The permissions assigned to the temporary credentials are determined by the permissions policy of the role being assumed. The following example permissions policy grants the role permission to list all objects that are contained in an S3 bucket named `productionapp`. It also allows the role to get, put, and delete objects within that bucket.

Example Example role permissions policy

```
{
  "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/*"
    }
  ]
}
```

Session policy passed as a parameter

Imagine that you want to allow a user to assume the same role as in the previous example. But in this case you want the role session to have permission only to get and put objects in the `productionapp` S3

bucket. You do not want to allow them to delete objects. One way to accomplish this is to create a new role and specify the desired permissions in that role's permissions policy. Another way to accomplish this is to call the `AssumeRole` API and include session policies in the optional `Policy` parameter as part of the API operation. The resulting session's permissions are the intersection of the role's identity-based policies and the session policies. Session policies cannot be used to grant more permissions than those allowed by the identity-based policy of the role that is being assumed. For more information about role session permissions, see [Session policies \(p. 353\)](#).

After you retrieve the new session's temporary credentials, you can pass them to the user that you want to have those permissions.

For example, imagine that the following policy is passed as a parameter of the API call. The person using the session has permissions to perform only these actions:

- List all objects in the `productionapp` bucket.
- Get and put objects in the `productionapp` bucket.

In the following session policy, the `s3>DeleteObject` permission is filtered out and the assumed session is not granted the `s3>DeleteObject` permission. The policy sets the maximum permissions for the role session so that it overrides any existing permissions policies on the role.

Example Example session policy passed with `AssumeRole` API call

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "s3>ListBucket",  
            "Resource": "arn:aws:s3:::productionapp"  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:GetObject",  
                "s3:PutObject"  
            ],  
            "Resource": "arn:aws:s3:::productionapp/*"  
        }  
    ]  
}
```

Resource-based policy

Some AWS resources support resource-based policies, and these policies provide another mechanism to define permissions that affect temporary security credentials. Only a few resources, like Amazon S3 buckets, Amazon SNS topics, and Amazon SQS queues support resource-based policies. The following example expands on the previous examples, using an S3 bucket named `productionapp`. The following policy is attached to the bucket.

When you attach the following resource-based policy to the `productionapp` bucket, *all* users are denied permission to delete objects from the bucket. (See the `Principal` element in the policy.) This includes all assumed role users, even though the role permissions policy grants the `DeleteObject` permission. An explicit Deny statement always takes precedence over an Allow statement.

Example Example bucket policy

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Deny",  
            "Principal": "*",  
            "Action": "s3>DeleteObject",  
            "Resource": "arn:aws:s3:::productionapp/*"  
        }  
    ]  
}
```

```
"Statement": {  
    "Principal": {"AWS": "*"},  
    "Effect": "Deny",  
    "Action": "s3>DeleteObject",  
    "Resource": "arn:aws:s3:::productionapp/*"  
}  
}
```

For more information about how multiple policy types are combined and evaluated by AWS, see [Policy evaluation logic \(p. 666\)](#).

Permissions for GetFederationToken

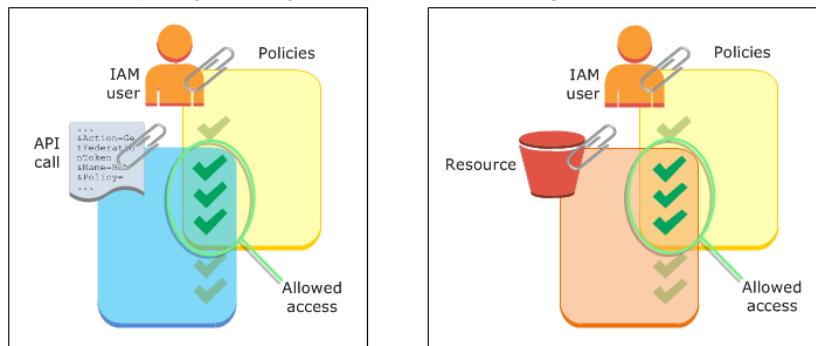
The `GetFederationToken` operation is called by an IAM user and returns temporary credentials for that user. This operation *federates* the user. The permissions assigned a federated user are defined in one of two places:

- The session policies passed as a parameter of the `GetFederationToken` API call. (This is most common.)
- A resource-based policy that explicitly names the federated user in the `Principal` element of the policy. (This is less common.)

Session policies are advanced policies that you pass as parameters when you programmatically create a temporary session. When you create a federated user session and pass session policies, the resulting session's permissions are the intersection of the IAM user's identity-based policy and the session policies. You cannot use the session policy to grant more permissions than those allowed by the identity-based policy of the user that is being federated.

In most cases if you do not pass a policy with the `GetFederationToken` API call, the resulting temporary security credentials have no permissions. However, a resource-based policy can provide additional permissions for the session. You can access a resource with a resource-based policy that specifies your session as the allowed principal.

The following figures show a visual representation of how the policies interact to determine permissions for the temporary security credentials returned by a call to `GetFederationToken`.



Example: Assigning permissions using GetFederationToken

You can use the `GetFederationToken` API action with different kinds of policies. Here are a few examples.

Policy attached to the IAM user

In this example, you have a browser-based client application that relies on two backend web services. One backend service is your own authentication server that uses your own identity system

to authenticate the client application. The other backend service is an AWS service that provides some of the client application's functionality. The client application is authenticated by your server, and your server creates or retrieves the appropriate permissions policy. Your server then calls the `GetFederationToken` API to obtain temporary security credentials, and returns those credentials to the client application. The client application can then make requests directly to the AWS service with the temporary security credentials. This architecture allows the client application to make AWS requests without embedding long-term AWS credentials.

Your authentication server calls the `GetFederationToken` API with the long-term security credentials of an IAM user named `token-app`. But the long-term IAM user credentials remain on your server and are never distributed to the client. The following example policy is attached to the `token-app` IAM user and defines the broadest set of permissions that your federated users (clients) will need. Note that the `sts:GetFederationToken` permission is required for your authentication service to obtain temporary security credentials for the federated users.

Note

AWS provides a sample Java application to serve this purpose, which you can download here:
[Token Vending Machine for Identity Registration - Sample Java Web Application](#).

Example Example policy attached to IAM user `token-app` that calls `GetFederationToken`

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "sts:GetFederationToken",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "dynamodb>ListTables",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "sns:ReceiveMessage",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "s3>ListBucket",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "sns>ListSubscriptions",
            "Resource": "*"
        }
    ]
}
```

The preceding policy grants several permissions to the IAM user. However, this policy alone doesn't grant any permissions to the federated user. If this IAM user calls `GetFederationToken` and does not pass a policy as a parameter of the API call, the resulting federated user has no effective permissions.

Session policy passed as parameter

The most common way to ensure that the federated user is assigned appropriate permission is to pass session policies in the `GetFederationToken` API call. Expanding on the previous example, imagine that `GetFederationToken` is called with the credentials of the IAM user `token-app`. Then imagine

that the following session policy is passed as a parameter of the API call. The resulting federated user has permission to list the contents of the Amazon S3 bucket named `productionapp`. The user can't perform the Amazon S3 `GetObject`, `PutObject`, and `DeleteObject` actions on items in the `productionapp` bucket.

The federated user is assigned these permissions because the permissions are the intersection of the IAM user policies and the session policies that you pass.

The federated user could not perform actions in Amazon SNS, Amazon SQS, Amazon DynamoDB, or in any S3 bucket except `productionapp`. These actions are denied even though those permissions are granted to the IAM user that is associated with the `GetFederationToken` call.

Example Example session policy passed as parameter of `GetFederationToken` API call

```
{  
    "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/*"]  
        }  
    ]  
}
```

Resource-based policies

Some AWS resources support resource-based policies, and these policies provide another mechanism to grant permissions directly to a federated user. Only some AWS services support resource-based policies. For example, Amazon S3 has buckets, Amazon SNS has topics, and Amazon SQS has queues that you can attach policies to. For a list of all services that support resource-based policies, see [AWS services that work with IAM \(p. 611\)](#) and review the "Resource-based policies" column of the tables. You can use resource-based policies to assign permissions directly to a federated user. Do this by specifying the Amazon Resource Name (ARN) of the federated user in the `Principal` element of the resource-based policy. The following example illustrates this and expands on the previous examples, using an S3 bucket named `productionapp`.

The following resource-based policy is attached to the bucket. This bucket policy allows a federated user named Carol to access the bucket. When the example policy described earlier is attached to the `token-app` IAM user, the federated user named Carol has permission to perform the `s3:GetObject`, `s3:PutObject`, and `s3:DeleteObject` actions on the bucket named `productionapp`. This is true even when no session policy is passed as a parameter of the `GetFederationToken` API call. That's because in this case the federated user named Carol has been explicitly granted permissions by the following resource-based policy.

Remember, a federated user is granted permissions only when those permissions are explicitly granted to both the IAM user **and** the federated user. Permissions can be granted to the federated user by the session policy passed as a parameter of the `GetFederationToken` API call. They can also be granted by a resource-based policy that explicitly names the federated user in the `Principal` element of the policy, as in the following example.

Example Example bucket policy that allows access to federated user

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {"Principal": {"AWS": "arn:aws:sts::ACCOUNT-ID-WITHOUT-HYPHENS:federated-user/Carol"},  
         "Effect": "Allow",  
         "Action": [  
             "s3:GetObject",  
             "s3:PutObject",  
             "s3:DeleteObject"  
         ],  
         "Resource": ["arn:aws:s3:::productionapp/*"]  
    }  
}
```

Permissions for GetSessionToken

The primary occasion for calling the `GetSessionToken` API operation or the `get-session-token` CLI command is when a user must be authenticated with multi-factor authentication (MFA). It is possible to write a policy that allows certain actions only when those actions are requested by a user who has been authenticated with MFA. In order to successfully pass the MFA authorization check, a user must first call `GetSessionToken` and include the optional `SerialNumber` and `TokenCode` parameters. If the user is successfully authenticated with an MFA device, the credentials returned by the `GetSessionToken` API operation include the MFA context. This context indicates that the user is authenticated with MFA and is authorized for API operations that require MFA authentication.

Permissions required for GetSessionToken

No permissions are required for a user to get a session token. The purpose of the `GetSessionToken` operation is to authenticate the user using MFA. You cannot use policies to control authentication operations.

To grant permissions to perform most AWS operations, you add the action with the same name to a policy. For example, to create a user, you must use the `CreateUser` API operation, the `create-user` CLI command, or the AWS Management Console. To perform these operations, you must have a policy that allows you to access the `CreateUser` action.

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

You can include the `GetSessionToken` action in your policies, but it has no effect on a user's ability to perform the `GetSessionToken` operation.

Permissions granted by GetSessionToken

If `GetSessionToken` is called with the credentials of an IAM user, the temporary security credentials have the same permissions as the IAM user. Similarly, if `GetSessionToken` is called with AWS account root user credentials, the temporary security credentials have root user permissions.

Note

We recommend that you do not call `GetSessionToken` with root user credentials. Instead, follow our [best practices \(p. 527\)](#) and create IAM users with the permissions they need. Then use these IAM users for everyday interaction with AWS.

The temporary credentials that you get when you call `GetSessionToken` have the following capabilities and limitations:

- You can use the credentials to access the AWS Management Console by passing the credentials to the federation single sign-on endpoint at `https://signin.aws.amazon.com/federation`. For more information, see [Enabling custom identity broker access to the AWS console \(p. 206\)](#).
- You **cannot** use the credentials to call IAM or AWS STS API operations. You **can** use them to call API operations for other AWS services.

Compare this API operation and its limitations and capability with the other API operations that create temporary security credentials at [Comparing the AWS STS API operations \(p. 311\)](#)

For more information about MFA-protected API access using `GetSessionToken`, see [Configuring MFA-protected API access \(p. 138\)](#).

Disabling permissions for temporary security credentials

Temporary security credentials are valid until they expire, and they cannot be revoked. However, because permissions are evaluated each time an AWS request is made using the credentials, you can achieve the effect of revoking the credentials by changing the permissions for the credentials even after they have been issued. If you remove all permissions from the temporary security credentials, subsequent AWS requests that use those credentials will fail. The mechanisms for changing or removing the permissions assigned to temporary security credentials are explained in the following sections.

Note

When you update existing policy permissions, or when you apply a new policy to a user or a resource, it may take a few minutes for policy updates to take effect.

Topics

- [Denying access to the creator of the temporary security credentials \(p. 323\)](#)
- [Denying access to temporary security credentials by name \(p. 324\)](#)
- [Denying access to temporary security credentials issued before a specific time \(p. 325\)](#)

Denying access to the creator of the temporary security credentials

To change or remove the permissions assigned to temporary security credentials, you can change or remove the permissions that are associated with the creator of the credentials. The creator of the credentials is determined by the AWS STS API that was used to obtain the credentials. The mechanisms for changing or removing the permissions associated with this creator are explained in the following sections.

Denying access to credentials created by `AssumeRole`, `AssumeRoleWithSAML`, or `AssumeRoleWithWebIdentity`

To change or remove the permissions assigned to the temporary security credentials obtained by calling the `AssumeRole`, `AssumeRoleWithSAML`, or `AssumeRoleWithWebIdentity` API operations, you edit or delete the role permission policy that defines the permissions for the assumed role. The temporary security credentials obtained by assuming a role can never have more permissions than those defined in the permissions policy of the assumed role, and the permissions assigned to temporary security credentials are evaluated each time they are used to make an AWS request. When you edit or delete the permission policy of a role, the changes affect the permissions of *all* temporary security

credentials associated with that role, including credentials that were issued before you changed the role's permissions policy. You can immediately revoke all permissions to a session by following the steps at [Revoking IAM role temporary security credentials \(p. 271\)](#).

For more information about editing a role permission policy, see [Modifying a role \(p. 273\)](#).

Denying access to credentials created by GetFederationToken or GetSessionToken

To change or remove the permissions assigned to the temporary security credentials obtained by calling the GetFederationToken or GetSessionToken API operations, you edit or delete the policies that are attached to the IAM user whose credentials were used to call GetFederationToken or GetSessionToken. The temporary security credentials that were obtained by calling GetFederationToken or GetSessionToken can never have more permissions than the IAM user whose credentials were used to obtain them. In addition the permissions assigned to temporary security credentials are evaluated each time they are used to make an AWS request. It is important to note that when you edit or delete the permissions of an IAM user, the changes affect the IAM user as well as all temporary security credentials created by that user.

Important

You cannot change the permissions for an AWS account root user. Likewise, you cannot change the permissions for the temporary security credentials that were created by calling GetFederationToken or GetSessionToken while signed in as the root user. For this reason, we recommend that you do not call GetFederationToken or GetSessionToken as a root user.

For information about how to change or remove the policies associated with the IAM user whose credentials were used to call GetFederationToken or GetSessionToken, see [Managing IAM policies \(p. 438\)](#).

Denying access to temporary security credentials by name

You can deny access to temporary security credentials without affecting the permissions of the IAM user or role that created the credentials. You do this by specifying the Amazon Resource Name (ARN) of the temporary security credentials in the Principal element of a resource-based policy. (Only some AWS services support resource-based policies.)

Denying access to federated users

For example, imagine you have an IAM user named token-app whose credentials are used to call GetFederationToken. The GetFederationToken API call resulted in temporary security credentials associated with a federated user named Bob (the federated user's name is taken from the Name parameter of the API call). To deny federated user Bob's access to an S3 bucket called EXAMPLE-BUCKET, you attach the following example bucket policy to EXAMPLE-BUCKET. It is important to note that this affects the federated user's Amazon S3 permissions only—any other permissions granted to the federated user remain intact.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {"Principal": {"AWS": "arn:aws:sts::ACCOUNT-ID-WITHOUT-HYPHENS:federated-user/Bob"},  
         "Effect": "Deny",  
         "Action": "s3:*",  
         "Resource": "arn:aws:s3:::EXAMPLE-BUCKET"  
    ]  
}
```

You can specify the ARN of the IAM user whose credentials were used to call GetFederationToken in the Principal element of the bucket policy, instead of specifying the federated user. In that case, the Principal element of the preceding policy would look like this:

```
"Principal": {"AWS": "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:user/token-app"}
```

It is important to note that specifying the ARN of IAM user token-app in the policy will result in denying access to *all* federated users created by token-app, not only the federated user named Bob.

Denying access to assumed role users

It is also possible to specify the ARN of the temporary security credentials that were created by assuming a role. The difference is the syntax used in the Principal element of the resource-based policy. For example, a user assumes a role called Accounting-Role and specifies a RoleSessionName of Mary (RoleSessionName is a parameter of the AssumeRole API call). To deny access to the temporary security credentials that resulted from this API call, the Principal element of the resource-based policy would look like this:

```
"Principal": {"AWS": "arn:aws:sts::ACCOUNT-ID-WITHOUT-HYPHENS:assumed-role/Accounting-Role/Mary"}
```

You can also specify the ARN of the IAM role in the Principal element of a resourced-based policy, as in the following example. In this case, the policy will result in denying access to *all* temporary security credentials associated with the role named Accounting-Role.

```
"Principal": {"AWS": "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:role/Accounting-Role"}
```

Denying access to temporary security credentials issued before a specific time

It is possible to deny access only to temporary security credentials that were created before a specific time and date. You do this by specifying a value for the aws:TokenIssueTime key in the Condition element of a policy. The following policy shows an example. You attach a policy similar to the following example to the IAM user that created the temporary security credentials. The policy denies all permissions but only when the value of aws:TokenIssueTime is earlier than the specified date and time. The value of aws:TokenIssueTime corresponds to the exact time at which the temporary security credentials were created. The aws:TokenIssueTime value is only present in the context of AWS requests that are signed with temporary security credentials, so the Deny statement in the policy will not affect requests that are signed with the long-term credentials of the IAM user.

The following policy can also be attached to a role. In that case, the policy affects only the temporary security credentials that were created by the role before the specified date and time. If the credentials were created by the role after the specified date and time, the Condition element in the policy is evaluated to false, so the Deny statement has no effect.

Example Example policy that denies all permissions to temporary credentials by issue time

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": "*",
      "Resource": "*",
      "Condition": {"DateLessThan": {"aws:TokenIssueTime": "2014-05-07T23:47:00Z"}}
    }
  ]
}
```

Valid users whose sessions are revoked in this way must acquire temporary credentials for a new session to continue working. Note that the AWS CLI caches credentials until they expire. To force the CLI to delete and refresh cached credentials that are no longer valid, run one of the following commands:

Linux, MacOS, or Unix

```
$ rm -r ~/.aws/cli/cache
```

Windows

```
C:\> del /s /q %UserProfile%\aws\cli\cache
```

Granting permissions to create temporary security credentials

By default, IAM users do not have permission to create temporary security credentials for federated users and roles. You must use a policy to provide your users with these permissions. Although you can grant permissions directly to a user, we strongly recommend that you grant permissions to a group. This makes management of the permissions much easier. When someone no longer needs to perform the tasks associated with the permissions, you simply remove them from the group. If someone else needs to perform that task, add them to the group to grant the permissions.

To grant an IAM group permission to create temporary security credentials for federated users or roles, you attach a policy that grants one or both of the following privileges:

- For federated users to access an IAM role, grant access to AWS STS AssumeRole.
- For federated users that don't need a role, grant access to AWS STS GetFederationToken.

For more information about the differences between the AssumeRole and GetFederationToken API operations, see [Requesting temporary security credentials \(p. 303\)](#).

IAM users can also call [GetSessionToken](#) to create temporary security credentials. No permissions are required for a user to call GetSessionToken. The purpose of this operation is to authenticate the user using MFA. You cannot use policies to control authentication. This means that you cannot prevent IAM users from calling GetSessionToken to create temporary credentials.

Example Example policy that grants permission to assume a role

The following example policy grants permission to call AssumeRole for the UpdateApp role in AWS account 123123123123. When AssumeRole is used, the user (or application) that creates the security credentials on behalf of a federated user cannot delegate any permissions that are not already specified in the role permission policy.

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

Example Example policy that grants permission to create temporary security credentials for a federated user

The following example policy grants permission to access GetFederationToken.

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

}

Important

When you give IAM users permission to create temporary security credentials for federated users with `GetFederationToken`, be aware that this permits those users to delegate their own permissions. For more information about delegating permissions across IAM users and AWS accounts, see [Examples of policies for delegating access \(p. 243\)](#). For more information about controlling permissions in temporary security credentials, see [Controlling permissions for temporary security credentials \(p. 316\)](#).

Example Example policy that grants a user limited permission to create temporary security credentials for federated users

When you let an IAM user call `GetFederationToken`, it is a best practice to restrict the permissions that the IAM user can delegate. For example, the following policy shows how to let an IAM user create temporary security credentials only for federated users whose names start with *Manager*.

```
{  
    "Version": "2012-10-17",  
    "Statement": [{  
        "Effect": "Allow",  
        "Action": "sts:GetFederationToken",  
        "Resource": ["arn:aws:sts::123456789012:federated-user/Manager*"]  
    }]  
}
```

Managing AWS STS in an AWS Region

By default, the AWS Security Token Service (AWS STS) is available as a global service, and all AWS STS requests go to a single endpoint at <https://sts.amazonaws.com>. AWS recommends using Regional AWS STS endpoints instead of the global endpoint to reduce latency, build in redundancy, and increase session token validity.

- **Reduce latency** – By making your AWS STS calls to an endpoint that is geographically closer to your services and applications, you can access AWS STS services with lower latency and better response times.
- **Build in redundancy** – You can add code to your application that switches your AWS STS API calls to a different Region. This ensures that if the first Region stops responding, your application continues to operate. This redundancy is not automatic; you must build the functionality into your code.
- **Increase session token validity** – Session tokens from Regional AWS STS endpoints are valid in all AWS Regions. Session tokens from the global STS endpoint are valid only in AWS Regions that are enabled by default. If you intend to enable a new Region for your account, you can use session tokens from Regional STS endpoints. If you choose to use the global endpoint, you must change the Region compatibility of STS session tokens for the global endpoint. Doing so ensures that tokens are valid in all AWS Regions.

Managing global endpoint session tokens

Most AWS Regions are enabled for operations in all AWS services by default. Those Regions are automatically activated for use with AWS STS. Some Regions, such as Asia Pacific (Hong Kong), must be manually enabled. To learn more about enabling and disabling AWS Regions, see [Managing AWS Regions in the AWS General Reference](#). When you enable these AWS Regions, they are automatically activated for use with AWS STS. You cannot activate the STS endpoint for a Region that is disabled. Tokens that are valid in all AWS Regions include more characters than tokens that are valid in Regions that are enabled by default. Changing this setting might affect existing systems where you temporarily store tokens.

You can change this setting using the AWS Management Console, AWS CLI, or AWS API.

To change the Region compatibility of session tokens for the global endpoint (console)

1. Sign in as a root user or an IAM user with permissions to perform IAM administration tasks. To change the compatibility of session tokens, you must have a policy that allows the `iam:SetSecurityTokenServicePreferences` action.
2. Open the [IAM console](#). In the navigation pane, choose **Account settings**.
3. If necessary, expand the **Security Token Service (STS)** section. In the first table next to **Global endpoint**, the **Region compatibility of session tokens** column indicates **Valid** only in AWS Regions enabled by default. Choose **Change**.
4. In the **Change region compatibility of session tokens for global endpoint** dialog box, select **Valid in all AWS Regions**. Then choose **Save changes**.

Note

Tokens that are valid in all AWS Regions include more characters than tokens that are valid in Regions that are enabled by default. Changing this setting might affect existing systems where you temporarily store tokens.

To change the Region compatibility of session tokens for the global endpoint (AWS CLI)

Set the security token version. Version 1 tokens are valid only in AWS Regions that are available by default. These tokens do not work in manually enabled Regions, such as Asia Pacific (Hong Kong). Version 2 tokens are valid in all Regions. However, version 2 tokens include more characters and might affect systems where you temporarily store tokens.

- `aws iam set-security-token-service-preferences`

To change the Region compatibility of session tokens for the global endpoint (AWS API)

Set the security token version. Version 1 tokens are valid only in AWS Regions that are available by default. These tokens do not work in manually enabled Regions, such as Asia Pacific (Hong Kong). Version 2 tokens are valid in all Regions. However, version 2 tokens include more characters and might affect systems where you temporarily store tokens.

- `SetSecurityTokenServicePreferences`

Activating and deactivating AWS STS in an AWS Region

When you activate STS endpoints for a Region, AWS STS can issue temporary credentials to users and roles in your account that make an AWS STS request. Those credentials can then be used in any Region that is enabled by default or is manually enabled. You must activate the Region in the account where the temporary credentials are generated. It does not matter whether a user is signed into the same account or a different account when they make the request.

For example, imagine a user in account A wants to send an `sts:AssumeRole` API request to the STS Regional endpoint `https://sts.us-west-2.amazonaws.com`. The request is for temporary credentials for the role named `Developer` in account B. Because the request is to create credentials for an entity in account B, account B must activate the `us-west-2` Region. Users from account A (or any other account) can call the `us-west-2` endpoint to request credentials for account B whether or not the Region is activated in their accounts.

Note

Active Regions are available to everyone that uses temporary credentials in that account. To control which IAM users or roles can access the Region, use the `aws:RequestedRegion` (p. 701) condition key in your permissions policies.

To activate or deactivate AWS STS in a Region that is enabled by default (console)

1. Sign in as a root user or an IAM user with permissions to perform IAM administration tasks.
2. Open the [IAM console](#) and in the navigation pane choose [Account settings](#).
3. If necessary, expand **Security Token Service (STS)**, find the Region that you want to activate, and then choose **Activate** or **Deactivate**. For Regions that must be enabled, we activate STS automatically when you enable the Region. After you enable a Region, AWS STS is always active for the Region and you cannot deactivate it. To learn how to enable a Region, see [Managing AWS Regions](#) in the *AWS General Reference*.

Writing code to use AWS STS regions

After you activate a Region, you can direct AWS STS API calls to that Region. The following Java code snippet demonstrates how to configure an `AWSSecurityTokenService` object to make requests to the Europe (Ireland) (eu-west-1) Region.

```
EndpointConfiguration regionEndpointConfig = new EndpointConfiguration("https://sts.eu-west-1.amazonaws.com", "eu-west-1");
AWSSecurityTokenService stsRegionalClient = AWSSecurityTokenServiceClientBuilder.standard()
    .withCredentials(credentials)
    .withEndpointConfiguration(regionEndpointConfig)
    .build();
```

AWS STS recommends that you make calls to a Regional endpoint. To learn how to manually enable a Region, see [Managing AWS Regions](#) in the *AWS General Reference*.

In the example, the first line instantiates an `EndpointConfiguration` object called `regionEndpointConfig`, passing the URL of the endpoint and the Region as the parameters.

For all other language and programming environment combinations, refer to the [documentation for the relevant SDK](#).

Regions and endpoints

The following table lists the Regions and their endpoints. It indicates which ones are activated by default and which ones you can activate or deactivate.

Region name	Endpoint	Active by default	Manually activate/deactivate
--Global--	sts.amazonaws.com	✓ Yes	✗ No
US East (Ohio)	sts.us-east-2.amazonaws.com	✓ Yes	✓ Yes
US East (N. Virginia)	sts.us-east-1.amazonaws.com	✓ Yes	✗ No
US West (N. California)	sts.us-west-1.amazonaws.com	✓ Yes	✓ Yes
US West (Oregon)	sts.us-west-2.amazonaws.com	✓ Yes	✓ Yes
Africa (Cape Town)	sts.af-south-1.amazonaws.com	✗ No ¹	✗ No

Region name	Endpoint	Active by default	Manually activate/deactivate
Asia Pacific (Hong Kong)	sts.ap-east-1.amazonaws.com	✗ No ¹	✗ No
Asia Pacific (Mumbai)	sts.ap-south-1.amazonaws.com	✓ Yes	✓ Yes
Asia Pacific (Seoul)	sts.ap-northeast-2.amazonaws.com	✓ Yes	✓ Yes
Asia Pacific (Singapore)	sts.ap-southeast-1.amazonaws.com	✓ Yes	✓ Yes
Asia Pacific (Sydney)	sts.ap-southeast-2.amazonaws.com	✓ Yes	✓ Yes
Asia Pacific (Tokyo)	sts.ap-northeast-1.amazonaws.com	✓ Yes	✓ Yes
Canada (Central)	sts.ca-central-1.amazonaws.com	✓ Yes	✓ Yes
Europe (Frankfurt)	sts.eu-central-1.amazonaws.com	✓ Yes	✓ Yes
Europe (Ireland)	sts.eu-west-1.amazonaws.com	✓ Yes	✓ Yes
Europe (London)	sts.eu-west-2.amazonaws.com	✓ Yes	✓ Yes
Europe (Milan)	sts.eu-south-1.amazonaws.com	✗ No ¹	✗ No
Europe (Paris)	sts.eu-west-3.amazonaws.com	✓ Yes	✓ Yes
Europe (Stockholm)	sts.eu-north-1.amazonaws.com	✓ Yes	✓ Yes
Middle East (Bahrain)	sts.me-south-1.amazonaws.com	✗ No ¹	✗ No
South America (São Paulo)	sts.sa-east-1.amazonaws.com	✓ Yes	✓ Yes

¹You must [enable the Region](#) to use it. This automatically activates STS. You cannot manually activate or deactivate STS in these Regions.

AWS CloudTrail and Regional endpoints

Calls to Regional endpoints, such as `us-east-2.amazonaws.com`, are logged in AWS CloudTrail the same as any call to a Regional service. Calls to the global endpoint, `sts.amazonaws.com`, are logged as calls to a global service. For more information, see [Logging IAM and AWS STS API calls with AWS CloudTrail \(p. 336\)](#).

Using AWS STS interface VPC endpoints

If you use Amazon Virtual Private Cloud (Amazon VPC) to host your AWS resources, you can establish a private connection between your VPC and AWS STS. You can use this connection to enable AWS STS to communicate with your resources on your VPC without going through the public internet.

Amazon VPC is an AWS service that you can use to launch AWS resources in a virtual network that you define. With a VPC, you have control over your network settings, such as the IP address range,

subnets, route tables, and network gateways. To connect your VPC to AWS STS, you define an *interface VPC endpoint* for AWS STS. The endpoint provides reliable, scalable connectivity to AWS STS without requiring an internet gateway, network address translation (NAT) instance, or VPN connection. For more information, see [What Is Amazon VPC?](#) in the *Amazon VPC User Guide*.

Interface VPC endpoints are powered by AWS PrivateLink, an AWS technology that enables private communication between AWS services using an elastic network interface with private IP addresses. For more information, see [AWS PrivateLink for AWS Services](#).

The following steps are for users of Amazon VPC. For more information, see [Getting Started with Amazon VPC](#) in the *Amazon VPC User Guide*.

Availability

AWS STS currently supports VPC endpoints in the following Regions:

- US East (Ohio)
- US East (N. Virginia)
- US West (N. California)
- US West (Oregon)
- Asia Pacific (Hong Kong)
- Asia Pacific (Mumbai)
- Asia Pacific (Osaka-Local)
- Asia Pacific (Seoul)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- Canada (Central)
- Europe (Frankfurt)
- Europe (Ireland)
- Europe (London)
- Europe (Paris)
- Europe (Stockholm)
- Middle East (Bahrain)
- South America (São Paulo)

Create a VPC for AWS STS

To start using AWS STS with your VPC, create an interface VPC endpoint for AWS STS. For more information, see [Creating an Interface Endpoint](#) in the *Amazon VPC User Guide*.

After you create the VPC endpoint, you must use the matching regional endpoint to send your AWS STS requests. AWS STS recommends that you use both the `setRegion` and `setEndpoint` methods to make calls to a Regional endpoint. You can use the `setRegion` method alone for manually enabled Regions, such as Asia Pacific (Hong Kong). In this case, the calls are directed to the STS Regional endpoint. To learn how to manually enable a Region, see [Managing AWS Regions](#) in the *AWS General Reference*. If you use the `setRegion` method alone for Regions enabled by default, the calls are directed to the global endpoint of <https://sts.amazonaws.com>.

When you use regional endpoints, AWS STS calls other AWS services using either public endpoints or private interface VPC endpoints, whichever are in use. For example, assume that you have created an interface VPC endpoint for AWS STS and have already requested temporary credentials from AWS STS from resources that are located in your VPC. In that case, these credentials begin flowing through the

interface VPC endpoint by default. For more information about making Regional requests using AWS STS, see [Managing AWS STS in an AWS Region \(p. 327\)](#).

Using bearer tokens

Some AWS services require that you have permission to get an AWS STS service bearer token before you can access their resources programmatically. These services support a protocol that requires you to use a bearer token instead of using a traditional [Signature Version 4 signed request](#). When you perform AWS CLI or AWS API operations that require bearer tokens, the AWS service requests a bearer token on your behalf. The service provides you with the token, which you can then use to perform subsequent operations in that service.

AWS STS service bearer tokens include information from your original principal authentication that might affect your permissions. This information can include principal tags, session tags, and session policies. The token's access key ID begins with the `ABIA` prefix. This helps you to identify operations that were performed using service bearer tokens in your CloudTrail logs.

Important

The bearer token can be used only for calls to the service that generates it and in the Region where it was generated. You can't use the bearer token to perform operations in other services or Regions.

An example of a service that supports bearer tokens is AWS CodeArtifact. Before you can interact with AWS CodeArtifact using a package manager such as NPM, Maven, or PIP, you must call the `aws codeartifact get-authorization-token` operation. This operation returns a bearer token that you can use to perform AWS CodeArtifact operations. Alternatively, you can use the `aws codeartifact login` command that completes the same operation and then configures your client automatically.

If you perform an action in an AWS service that generates a bearer token for you, you must have the following permissions in your IAM policy:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowServiceBearerToken",  
            "Effect": "Allow",  
            "Action": "sts:GetServiceBearerToken",  
            "Resource": "arn:aws:iam::111122223333:user/TestUser1"  
        }  
    ]  
}
```

Sample applications that use temporary credentials

You can use AWS Security Token Service (AWS STS) to create and provide trusted users with temporary security credentials that can control access to your AWS resources. For more information about AWS STS, see [Temporary security credentials in IAM \(p. 301\)](#). To see how you can use AWS STS to manage temporary security credentials, you can download the following sample applications that implement complete example scenarios:

- [Identity Federation Sample Application for an Active Directory Use Case](#). Demonstrates how to use permissions that are tied to a user defined in Active Directory (.NET/C#) to issue temporary security credentials for accessing Amazon S3 files and buckets.
- [AWS Management Console Federation Proxy Sample Use Case](#). Demonstrates how to create a custom federation proxy that enables single sign-on (SSO) so that existing Active Directory users can sign into the AWS Management Console (.NET/C#).

- [Integrate Shibboleth with AWS Identity and Access Management](#). Shows how to use **Shibboleth** and **SAML** (p. 182) to provide users with single sign-on (SSO) access to the AWS Management Console.

Samples for web identity federation

The following sample applications illustrate how to use web identity federation with providers like Login with Amazon, Amazon Cognito, Facebook, or Google. You can trade authentication from these providers for temporary AWS security credentials to access AWS services.

- [Amazon Cognito Tutorials](#) – We recommend that you use Amazon Cognito with the AWS SDKs for mobile development. Amazon Cognito is the simplest way to manage identity for mobile apps, and it provides additional features like synchronization and cross-device identity. For more information about Amazon Cognito, see [Amazon Cognito Identity](#) in the *AWS Mobile SDK for Android Developer Guide* and [Authenticate Users with Amazon Cognito Identity](#) in the *AWS Mobile SDK for iOS Developer Guide*.
- [Web Identity Federation Playground](#). This website provides an interactive demonstration of [web identity federation](#) (p. 177) and the `AssumeRoleWithWebIdentity` API.
- [Build and Deploy a Federated Web Identity Application with AWS Elastic Beanstalk and Login with Amazon](#). This blog post describes how to use `AssumeRoleWithWebIdentity` to obtain temporary security credentials through web identity federation and Login with Amazon. It also explains how to use those credentials in a Python web application that runs on Elastic Beanstalk to make calls to AWS.

Additional resources for temporary security credentials

The following scenarios and applications can guide you in using temporary security credentials:

- [About web identity federation](#) (p. 177). This section discusses how to configure IAM roles when you use web identity federation and the `AssumeRoleWithWebIdentity` API.
- [Configuring MFA-protected API access](#) (p. 138). This topic explains how to use roles to require multi-factor authentication (MFA) to protect sensitive API actions in your account.
- [Token Vending Machine for Identity Registration](#). This sample Java web application uses the `GetFederationToken` API to serve temporary security credentials to remote clients.

For more information on policies and permissions in AWS see the following topics:

- [Access management for AWS resources](#) (p. 350)
- [Policy evaluation logic](#) (p. 666).
- [Managing Access Permissions to Your Amazon S3 Resources](#) in *Amazon Simple Storage Service Developer Guide*.
- To learn whether principals in accounts outside of your zone of trust (trusted organization or account) have access to assume your roles, see [What is IAM Access Analyzer?](#).

AWS account root user

When you first create an Amazon Web Services (AWS) account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the **AWS account root user**. You can sign in as the root user using the email address and password that you used to create the account.

Important

We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the [best practice of using the root user only to create your first IAM user \(p. 528\)](#). Then securely lock away the root user credentials and use them to perform only a few account and service management tasks. To view the tasks that require you to sign in as the root user, see [AWS Tasks That Require Root User](#). For a tutorial on how to set up an administrator for daily use, see [Creating your first IAM admin user and group \(p. 20\)](#).

You can create, rotate, disable, or delete access keys (access key IDs and secret access keys) for your AWS account root user. You can also change your root user password. Anyone who has root user credentials for your AWS account has unrestricted access to all the resources in your account, including billing information.

When you create access keys, you create the access key ID and secret access key as a set. During access key creation, AWS gives you one opportunity to view and download the secret access key part of the access key. If you don't download it or if you lose it, you can delete the access key and then create a new one. You can create root user access keys with the [IAM console](#), AWS CLI, or AWS API.

A newly created access key has the status of *active*, which means that you can use the access key for CLI and API calls. You are [limited to two access keys](#) for each IAM user, which is useful when you want to [rotate the access keys](#). You can also assign up to two access keys to the root user. When you disable an access key, you can't use it for API calls, and inactive keys do count toward your limit. You can create or delete an access key any time. However, when you delete an access key, it's gone forever and can't be retrieved.

You can change the email address and password on the [Security Credentials](#) page. You can also choose [Forgot password?](#) on the AWS sign-in page to reset your password.

Tasks

- [Create or delete an AWS account \(p. 334\)](#)
- [Enable MFA on the AWS account root user \(p. 334\)](#)
- [Creating access keys for the root user \(p. 335\)](#)
- [Deleting access keys for the root user \(p. 335\)](#)
- [Changing the password for the root user \(p. 336\)](#)
- [Securing the credentials for the root user \(p. 336\)](#)

Create or delete an AWS account

For more information, see the following articles in the AWS Knowledge Center:

- [How do I create and activate an AWS account?](#)
- [How do I close my AWS account?](#)

Enable MFA on the AWS account root user

If you continue to use the root user credentials, we recommend that you follow the security best practice to enable multi-factor authentication (MFA) for your account. Because your root user can perform sensitive operations in your account, adding an additional layer of authentication helps you to better secure your account. Multiple types of MFA are available. For more information about enabling MFA, see the following:

- [Enable a virtual MFA device for your AWS account root user \(console\) \(p. 115\)](#)
- [Enable a hardware MFA device for the AWS account root user \(console\) \(p. 125\)](#)

Creating access keys for the root user

You can use the AWS Management Console or AWS programming tools to create access keys for the root user.

To create an access key for the AWS account root user (console)

1. Sign in to the [IAM console](#) as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

Note

If you see three text boxes, then you previously signed in to the console with [IAM user](#) credentials. Your browser might remember this preference and open this account-specific sign-in page every time that you try to sign in. You cannot use the IAM user sign-in page to sign in as the account owner. If you see the [IAM user sign-in page](#), choose **Sign in using root user email** near the bottom of the page. This returns you to the main sign-in page. From there, you can sign in as the root user using your AWS account email address and password.

2. Choose your account name in the navigation bar, and then choose **My Security Credentials**.
3. If you see a warning about accessing the security credentials for your AWS account, choose **Continue to Security Credentials**.
4. Expand the **Access keys (access key ID and secret access key)** section.
5. Choose **Create New Access Key**. If this feature is disabled, then you must delete one of the existing access keys before you can create a new key. For more information, see [IAM Entity Object Limits](#) in the [IAM User Guide](#).

A warning explains that you have only this one opportunity to view or download the secret access key. It cannot be retrieved later.

- If you choose **Show Access Key**, you can copy the access key ID and secret key from your browser window and paste it somewhere else.
 - If you choose **Download Key File**, you receive a file named `rootkey.csv` that contains the access key ID and the secret key. Save the file somewhere safe.
6. When you no longer use the access key [we recommend that you delete it \(p. 533\)](#), or at least mark it inactive by choosing **Make Inactive** so that it cannot be misused.

To create an access key for the root user (AWS CLI or AWS API)

Use one of the following:

- AWS CLI: `aws iam create-access-key`
- AWS API: `CreateAccessKey`

Deleting access keys for the root user

You can use the AWS Management Console to delete access keys for the root user. You cannot use the AWS CLI or the AWS API to delete the root user access keys.

1. Use your AWS account email address and password to sign in to the [AWS Management Console](#) as the AWS account root user.

Note

If you see three text boxes, then you previously signed in to the console with [IAM user](#) credentials. Your browser might remember this preference and open this account-specific sign-in page every time that you try to sign in. You cannot use the IAM user sign-in page to

sign in as the account owner. If you see the [IAM user sign-in page](#), choose **Sign in using root user email** near the bottom of the page. This returns you to the main sign-in page. From there, you can sign in as the root user using your AWS account email address and password.

2. Choose your account name in the navigation bar, and then choose **My Security Credentials**.
3. If you see a warning about accessing the security credentials for your AWS account, choose **Continue to Security Credentials**.
4. Expand the **Access keys (access key ID and secret access key)** section.
5. Find the access key that you want to delete, and then, under the **Actions** column, choose **Delete**.

Note

You can mark an access key as inactive instead of deleting it. This enables you to resume use of it in the future without having to change either the key ID or secret key. While it is inactive, any attempts to use it in requests to the AWS API fail with the status of access denied.

Changing the password for the root user

For information about changing the root user's password, see [Changing the AWS account root user password \(p. 92\)](#). To change the root user, you must log in using the root user credentials. To view the tasks that require you to sign in as the root user, see [AWS Tasks that Require Root User](#)

Securing the credentials for the root user

For more information about securing the credentials for the AWS account root user, see [Lock away your AWS account root user access keys \(p. 528\)](#).

Logging IAM and AWS STS API calls with AWS CloudTrail

IAM and AWS STS are integrated with AWS CloudTrail, a service that provides a record of actions taken by an IAM user or role. CloudTrail captures all API calls for IAM and AWS STS as events, including calls from the console and from API calls. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. You can use CloudTrail to get information about the request that was made to IAM or AWS STS. For example, you can view the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the [AWS CloudTrail User Guide](#).

Topics

- [IAM and AWS STS information in CloudTrail \(p. 337\)](#)
- [Logging IAM and AWS STS API requests \(p. 337\)](#)
- [Logging API requests to other AWS services \(p. 337\)](#)
- [Logging regional sign-in events \(p. 338\)](#)
- [Logging user sign-in events \(p. 340\)](#)
- [Logging sign-in events for temporary credentials \(p. 340\)](#)
- [Example IAM API events in CloudTrail log \(p. 341\)](#)
- [Example AWS STS API events in CloudTrail log \(p. 342\)](#)
- [Example sign-in events in CloudTrail log \(p. 348\)](#)

IAM and AWS STS information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in IAM or AWS STS, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see [Viewing Events with CloudTrail Event History](#).

For an ongoing record of events in your AWS account, including events for IAM and AWS STS, create a trail. A trail enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see:

- [Overview for Creating a Trail](#)
- [CloudTrail Supported Services and Integrations](#)
- [Configuring Amazon SNS Notifications for CloudTrail](#)
- [Receiving CloudTrail Log Files from Multiple Regions](#) and [Receiving CloudTrail Log Files from Multiple Accounts](#)

All IAM and AWS STS actions are logged by CloudTrail and are documented in the [IAM API Reference](#) and the [AWS Security Token Service API Reference](#).

Logging IAM and AWS STS API requests

CloudTrail logs all authenticated API requests (made with credentials) to IAM and AWS STS API operations. CloudTrail also logs nonauthenticated requests to the AWS STS actions, `AssumeRoleWithSAML` and `AssumeRoleWithWebIdentity`, and logs information provided by the identity provider. You can use this information to map calls made by a federated user with an assumed role back to the originating external federated caller. In the case of `AssumeRole`, you can map calls back to the originating AWS service or to the account of the originating user. The `userIdentity` section of the JSON data in the CloudTrail log entry contains the information that you need to map the `AssumeRole*` request with a specific federated user. For more information, see [CloudTrail userIdentity Element](#) in the [AWS CloudTrail User Guide](#).

For example, calls to the IAM `CreateUser`, `DeleteRole`, `ListGroups`, and other API operations are all logged by CloudTrail.

Examples for this type of log entry are presented later in this topic.

Important

If you activate AWS STS endpoints in Regions other than the default global endpoint, then you must also turn on CloudTrail logging in those Regions. This is necessary to record any AWS STS API calls that are made in those Regions. For more information, see [Turning On CloudTrail in Additional Regions](#) in the AWS CloudTrail User Guide.

Logging API requests to other AWS services

Authenticated requests to other AWS service API operations are logged by CloudTrail, and these log entries contain information about who generated the request.

For example, assume that you made a request to list Amazon EC2 instances or create an CodeDeploy deployment group. Details about the person or service that made the request are contained in the log entry for that request. This information helps you determine whether the request was made by the AWS account root user, an IAM user, a role, or another AWS service.

For more details about the user identity information in CloudTrail log entries, see [userIdentity Element](#) in the *AWS CloudTrail User Guide*.

Logging regional sign-in events

If you enable CloudTrail to log sign-in events to your logs, you need to be aware of how CloudTrail chooses where to log the events.

- If your users sign in directly to a console, they are redirected to either a global or a Regional sign-in endpoint. The endpoint depends on whether the selected service console supports Regions. For example, the main console home page supports Regions. If you sign in to <https://alias.signin.aws.amazon.com/console>, you are redirected to a Regional sign-in endpoint such as <https://us-east-2.signin.aws.amazon.com>. This redirection creates a Regional CloudTrail log entry in the user's Region's log.

On the other hand, the Amazon S3 console does not support Regions, so if you sign in to <https://alias.signin.aws.amazon.com/console/s3>, AWS redirects you to the global sign-in endpoint at <https://signin.aws.amazon.com>. This redirection creates a global CloudTrail log entry.

- You can manually request a certain Regional sign-in endpoint by signing in to the Region-enabled main console home page using a URL like <https://alias.signin.aws.amazon.com/console?region=ap-southeast-1>. In this case, AWS redirects you to the ap-southeast-1 Regional sign-in endpoint and results in a Regional CloudTrail log event.

Whether the sign-in event is considered Regional or global depends on the console the user signs into and how the user constructs the sign-in URL.

- Is the service console Regionalized? If so, then the sign-in request is automatically redirected to a Regional sign-in endpoint and the event is logged in that Region's CloudTrail log. For example, if you sign in to <https://alias.signin.aws.amazon.com/console>, which is Regionalized, you are redirected to a sign-in endpoint in your Region, such as <https://us-east-2.signin.aws.amazon.com>. The event is logged in that Region's log.

However, some services are not Regionalized yet. For example, the Amazon S3 service is *not* currently Regionalized. If you sign in to <https://alias.signin.aws.amazon.com/console/s3>, you are redirected to the global sign-in endpoint at <https://signin.aws.amazon.com>. This redirection creates an event in your global log.

- You can also manually request a specific Regional sign-in endpoint by using a URL such as <https://alias.signin.aws.amazon.com/console?region=ap-southeast-1>. This URL redirects to the ap-southeast-1 Regional sign-in endpoint. This redirection results in an event in the Regional log.

Preventing duplicate regional log entries

CloudTrail creates separate trails in each Region. These trails include information for events that occur in those Regions, plus global events and events that are not region-specific. Examples include IAM API calls, AWS STS calls to the global endpoint, and AWS sign-in events. For example, assume that you have two trails, each in a different Region. If you then create a new IAM user, the `CreateUser` event is added to the log files in both Regions, creating a duplicate log entry.

AWS Security Token Service (STS) is a global service with a single global endpoint at <https://sts.amazonaws.com>. Calls to this endpoint are logged as calls to a global service. However, because this endpoint is physically located in the US East (N. Virginia) Region, your logs list us-east-1 as the event Region. CloudTrail does not write these logs to the US East (Ohio) Region unless you choose to include global service logs in that Region. AWS STS also allows calls to Regional endpoints, such as `sts.eu-central-1.amazonaws.com`. CloudTrail writes calls to all Regional endpoints to their respective Regions. For example, calls to `sts.us-east-2.amazonaws.com` are published to the US East (Ohio)

Region. Calls to `sts.eu-central-1.amazonaws.com` are published in the Europe (Frankfurt) Region logs.

For more information about multiple Regions and AWS STS, see [Managing AWS STS in an AWS Region \(p. 327\)](#).

The following table lists the Regions and how CloudTrail logs AWS STS requests in each Region. The "Location" column indicates which logs CloudTrail writes to. "Global" means that the event is logged in any Region for which you choose to include global service logs in that Region. "Region" means that the event is logged only in the Region where the endpoint is located. The last column indicates how the request's Region is identified in the log entry.

Region name	Region identity in CloudTrail log	Endpoint	Location of CloudTrail logs
n/a - global	us-east-1	sts.amazonaws.com	Global
US East (Ohio)	us-east-2	sts.us-east-2.amazonaws.com	Region
US East (N. Virginia)	us-east-1	sts.us-east-1.amazonaws.com	Region
US West (N. California)	us-west-1	sts.us-west-1.amazonaws.com	Region
US West (Oregon)	us-west-2	sts.us-west-2.amazonaws.com	Region
Canada (Central)	ca-central-1	sts.ca-central-1.amazonaws.com	Region
Europe (Frankfurt)	eu-central-1	sts.eu-central-1.amazonaws.com	Region
Europe (Ireland)	eu-west-1	sts.eu-west-1.amazonaws.com	Region
Europe (London)	eu-west-2	sts.eu-west-2.amazonaws.com	Region
Asia Pacific (Tokyo)	ap-northeast-1	sts.ap-northeast-1.amazonaws.com	Region
Asia Pacific (Seoul)	ap-northeast-2	sts.ap-northeast-2.amazonaws.com	Region
Asia Pacific (Mumbai)	ap-south-1	sts.ap-south-1.amazonaws.com	Region
Asia Pacific (Singapore)	ap-southeast-1	sts.ap-southeast-1.amazonaws.com	Region
Asia Pacific (Sydney)	ap-southeast-2	sts.ap-southeast-2.amazonaws.com	Region
South America (São Paulo)	sa-east-1	sts.sa-east-1.amazonaws.com	Region

When you configure CloudTrail to aggregate trail information from multiple Regions in your account into a single Amazon S3 bucket, IAM events are duplicated in the logs. In other words, the trail for each Region writes the same IAM event to the aggregated log. To prevent this duplication, you can include global events selectively. A typical approach is to enable global events in one trail. Then disable global events in all other trails that write to the same Amazon S3 bucket. That way only one set of global events is written.

For more information, see [Aggregating Logs](#) in the *AWS CloudTrail User Guide*.

Logging user sign-in events

CloudTrail logs sign-in events to the AWS Management Console, the AWS discussion forums, and AWS Marketplace. CloudTrail logs successful and failed sign-in attempts for IAM users and federated users.

For AWS account root users, only successful sign-in events are logged. Unsuccessful sign-in events by the root user are *not* logged by CloudTrail.

As a security best practice, AWS does not log the entered IAM user name text when the sign-in failure is caused by *an incorrect user name*. The user name text is masked by the value HIDDEN_DUE_TO_SECURITY_REASONS. For an example of this, see [Example sign-in failure event caused by incorrect user name \(p. 349\)](#), later in this topic. The user name text is obscured because such failures might be caused by user errors. Logging these errors could expose potentially sensitive information. For example:

- You accidentally type your password in the user name box.
- You choose the link for one AWS account's sign-in page, but then type the account number for a different one.
- You forget which account you are signing in to and accidentally type the account name of your personal email account, your bank sign-in identifier, or some other private ID.

Logging sign-in events for temporary credentials

When a principal requests temporary credentials, the principal type determines how CloudTrail logs the event. This can be complicated when a principal assumes a role in another account. There are multiple API calls to perform operations related to role cross-account operations. First, the principal calls an AWS STS API to retrieve the temporary credentials. That operation is logged in the calling account and the account where the AWS STS operation is performed. Then the principal then uses the role to perform other API calls in the assumed role's account.

You can use the `aws:RoleSessionName` condition key to require that your users provide a specific session name when they assume a role. For example, you can require that IAM users specify their own user name as their session name. This makes it easier for administrators that are reviewing AWS CloudTrail logs to learn who performed an action. For more information, see [aws:RoleSessionName \(p. 718\)](#).

The following table shows how CloudTrail logs different information for each of the API calls that generate temporary credentials.

Principal type	STS API	User identity in CloudTrail log for caller's account	User identity in CloudTrail log for the assumed role's account	User identity in CloudTrail log for the role's subsequent API calls
AWS account root user credentials	GetSessionToken	Root user identity	Role owner account is same as calling account	Root user identity
IAM user	GetSessionToken	IAM user identity	Role owner account is same as calling account	IAM user identity
IAM user	GetFederationToken	IAM user identity	Role owner account is same as calling account	IAM user identity

Principal type	STS API	User identity in CloudTrail log for caller's account	User identity in CloudTrail log for the assumed role's account	User identity in CloudTrail log for the role's subsequent API calls
IAM user	AssumeRole	IAM user identity	Account number and principal ID (if a user), or AWS service principal	Role identity only (no user)
Externally authenticated user	AssumeRoleWithSAMLMToken	SAML user identity	SAML user identity	Role identity only (no user)
Externally authenticated user	AssumeRoleWithWebIdentity	OIDC/Web user identity	OIDC/Web user identity	Role identity only (no user)

Example IAM API events in CloudTrail log

CloudTrail log files contain events that are formatted using JSON. An API event represents a single API request and includes information about the principal, the requested action, any parameters, and the date and time of the action.

Example IAM API event in CloudTrail log file

The following example shows a CloudTrail log entry for a request made for the IAM `GetUserPolicy` action.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE",
    "arn": "arn:aws:iam::444455556666:user/JaneDoe",
    "accountId": "444455556666",
    "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
    "userName": "JaneDoe",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2014-07-15T21:39:40Z"
      }
    },
    "invokedBy": "signin.amazonaws.com"
  },
  "eventTime": "2014-07-15T21:40:14Z",
  "eventSource": "iam.amazonaws.com",
  "eventName": " GetUserPolicy",
  "awsRegion": "us-east-2",
  "sourceIPAddress": "signin.amazonaws.com",
  "userAgent": "signin.amazonaws.com",
  "requestParameters": {
    "userName": "JaneDoe",
    "policyName": "ReadOnlyAccess-JaneDoe-201407151307"
  },
  "responseElements": null,
  "requestID": "9EXAMPLE-0c68-11e4-a24e-d5e16EXAMPLE",
  "eventID": "cEXAMPLE-127e-4632-980d-505a4EXAMPLE"
}
```

From this event information, you can determine that the request was made to get a user policy named `ReadOnlyAccess-JaneDoe-201407151307` for user `JaneDoe`, as specified in the `requestParameters` element. You can also see that the request was made by an IAM user named `JaneDoe` on July 15, 2014 at 9:40 PM (UTC). In this case, the request originated in the AWS Management Console, as you can tell from the `userAgent` element.

Example AWS STS API events in CloudTrail log

CloudTrail log files contain events that are formatted using JSON. An API event represents a single API request and includes information about the principal, the requested action, any parameters, and the date and time of the action.

Example cross-account AWS STS API events in CloudTrail log files

The IAM user named `JohnDoe` in account `777788889999` calls the AWS STS `AssumeRole` action to assume the role `EC2-dev` in account `111122223333`.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDAQRSTUVWXYZEXAMPLE",
    "arn": "arn:aws:iam::777788889999:user/JohnDoe",
    "accountId": "777788889999",
    "accessKeyId": "AKIAQRSTUVWXYZEXAMPLE",
    "userName": "JohnDoe"
  },
  "eventTime": "2014-07-18T15:07:39Z",
  "eventSource": "sts.amazonaws.com",
  "eventName": "AssumeRole",
  "awsRegion": "us-east-2",
  "sourceIPAddress": "192.0.2.101",
  "userAgent": "aws-cli/1.11.10 Python/2.7.8 Linux/3.2.45-0.6.wd.865.49.315.metal1.x86_64 botocore/1.4.67",
  "requestParameters": {
    "roleArn": "arn:aws:iam::111122223333:role/EC2-dev",
    "roleSessionName": "JohnDoe-EC2-dev",
    "serialNumber": "arn:aws:iam::777788889999:mfa"
  },
  "responseElements": {
    "credentials": {
      "sessionToken": "<encoded session token blob>",
      "accessKeyId": "AKIAQRSTUVWXYZEXAMPLE",
      "expiration": "Jul 18, 2014 4:07:39 PM"
    },
    "assumedRoleUser": {
      "assumedRoleId": "AIDAQRSTUVWXYZEXAMPLE:JohnDoe-EC2-dev",
      "arn": "arn:aws:sts::111122223333:assumed-role/EC2-dev/JohnDoe-EC2-dev"
    }
  },
  "resources": [
    {
      "ARN": "arn:aws:iam::111122223333:role/EC2-dev",
      "accountId": "111122223333",
      "type": "AWS::IAM::Role"
    }
  ],
  "requestID": "4EXAMPLE-0e8d-11e4-96e4-e55c0EXAMPLE",
  "sharedEventID": "bEXAMPLE-efea-4a70-b951-19a88EXAMPLE",
  "eventID": "dEXAMPLE-ac7f-466c-a608-4ac8dEXAMPLE",
  "eventType": "AwsApiCall",
```

```
    "recipientAccountId": "111122223333"
}
```

The second example shows the assumed role account's (111122223333) CloudTrail log entry for the same request.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AWSAccount",
    "principalId": "AIDAQRSTUVWXYZEXAMPLE",
    "accountId": "777788889999"
  },
  "eventTime": "2014-07-18T15:07:39Z",
  "eventSource": "sts.amazonaws.com",
  "eventName": "AssumeRole",
  "awsRegion": "us-east-2",
  "sourceIPAddress": "192.0.2.101",
  "userAgent": "aws-cli/1.11.10 Python/2.7.8 Linux/3.2.45-0.6.wd.865.49.315.metal1.x86_64 botocore/1.4.67",
  "requestParameters": {
    "roleArn": "arn:aws:iam::111122223333:role/EC2-dev",
    "roleSessionName": "JohnDoe-EC2-dev",
    "serialNumber": "arn:aws:iam::777788889999:mfa"
  },
  "responseElements": {
    "credentials": {
      "sessionToken": "<encoded session token blob>",
      "accessKeyId": "AKIAQRSTUVWXYZEXAMPLE",
      "expiration": "Jul 18, 2014 4:07:39 PM"
    },
    "assumedRoleUser": {
      "assumedRoleId": "AIDAQRSTUVWXYZEXAMPLE:JohnDoe-EC2-dev",
      "arn": "arn:aws:sts::111122223333:assumed-role/EC2-dev/JohnDoe-EC2-dev"
    }
  },
  "requestID": "4EXAMPLE-0e8d-11e4-96e4-e55c0EXAMPLE",
  "sharedEventID": "bEXAMPLE-efea-4a70-b951-19a88EXAMPLE",
  "eventID": "dEXAMPLE-ac7f-466c-a608-4ac8dEXAMPLE"
}
```

Example AWS STS role chaining API event in CloudTrail log file

The following example shows a CloudTrail log entry for a request made by John Doe in account 111111111111. John previously used his JohnDoe user to assume the JohnRole1 role. For this request, he uses the credentials from that role to assume the JonRole2 role. This is known as [role chaining \(p. 169\)](#). John passes two [session tags \(p. 293\)](#) into the request. He sets those two tags as transitive. The request inherits the Department tag as transitive because John set it as transitive when he assumed JohnRole1. For more information about transitive keys in role chains, see [Chaining roles with session tags \(p. 299\)](#).

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AROAIN5ATK5U7KEXAMPLE:JohnRole1",
    "arn": "arn:aws:sts::111111111111:assumed-role/JohnDoe/JohnRole1",
    "accountId": "111111111111",
    "accessKeyId": "AKIAI44QH8DHBEEXAMPLE",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "transitiveTags": [
          "Department:Finance"
        ]
      }
    }
  }
}
```

```

        "creationDate": "2019-10-02T21:50:54Z"
    },
    "sessionIssuer": {
        "type": "Role",
        "principalId": "AROAIN5ATK5U7KEXAMPLE",
        "arn": "arn:aws:iam::111111111111:role/JohnRole1",
        "accountId": "111111111111",
        "userName": "JohnDoe"
    }
},
{
    "eventTime": "2019-10-02T22:12:29Z",
    "eventSource": "sts.amazonaws.com",
    "eventName": "AssumeRole",
    "awsRegion": "us-east-2",
    "sourceIPAddress": "123.145.67.89",
    "userAgent": "aws-cli/1.16.248 Python/3.4.7
Linux/4.9.184-0.1.ac.235.83.329.metal1.x86_64 botocore/1.12.239",
    "requestParameters": {
        "incomingTransitiveTags": {
            "Department": "Engineering"
        },
        "tags": [
            {
                "value": "johndoe@example.com",
                "key": "Email"
            },
            {
                "value": "12345",
                "key": "CostCenter"
            }
        ],
        "roleArn": "arn:aws:iam::111111111111:role/JohnRole2",
        "roleSessionName": "Role2WithTags",
        "transitiveTagKeys": [
            "Email",
            "CostCenter"
        ],
        "durationSeconds": 3600
    },
    "responseElements": {
        "credentials": {
            "accessKeyId": "ASIAWHOJDLGPOEXAMPLE",
            "expiration": "Oct 2, 2019 11:12:29 PM",
            "sessionToken": "AgoJb3JpZ2luX2VjEB4aCXVzLXd1c3QtMSJHMEXAMPLETOKEN
+//rJb8Lo30mFc5MlhFCEbubZvEj0wHB/mDMwIgSEe9gk/Zjr09tzV7F1HDTMhmEXAMPLETOKEN/iEJ
rkqngII9/////////
ARABGgw0MjgzMDc4NjM5NjYiDLZjZFKwP4qxQG5sFCryASo4UPz5qE97wPPH1eLMvs7CgSDBSWfonmRTcfokm2FN1+hWUDQQH6adjbb
+C+WKFZb701eiv9J5La2EXAMPLETOKEN/c7S5Iro1WUJ0q3Cxuo/8HUoSxvhQHM7zF7mWWLhXLEQ52ivL
+F6q5dpXu4aTFedpMfnJa8JtkWwG9x1Axj0Ypy2ok8v5unpQGWych1vwdvj6ez1Dm8Xg1+qIzXILiEXAMPLETOKEN/
vQGqu8H+nxp3kabcrOvTFTvxX6vsc8OGwUfHzAfYGEEXAMPLETOKEN/
L6vlyMM3B1OwFOrQBno1HEjf1oNI8RnQiMNfduOtwyj7HUZIOCZmjfN8PPHq77N7GJ19lzvIZKQA00wcjg
+mc78zHCj8yOs1Y8C96paEXAMPLETOKEN/
E3cpksxWdgs91HRzJWScjN2+r2LTGjYhyPqcmFzzo2mCE7mBNEXAMPLETOKEN/oJy
+2o83YNw5t0iDmczgDzJZ4UKR84yGYOMfSnF4XcEJrDgAJ3OJFwmTcTQICALSwLEXAMPLETOKEN"
        },
        "assumedRoleUser": {
            "assumedRoleId": "AROAIFR7WHDTSOYQYHFUE:Role2WithTags",
            "arn": "arn:aws:sts::111111111111:assumed-role/test-role/Role2WithTags"
        }
    },
    "requestID": "b96b0e4e-e561-11e9-8b3f-7b396EXAMPLE",
    "eventID": "1917948f-3042-46ec-98e2-62865EXAMPLE",
    "resources": [
        {
            "ARN": "arn:aws:iam::111122223333:role/JohnRole2",

```

```
        "accountId": "111111111111",
        "type": "AWS::IAM::Role"
    },
],
"eventType": "AwsApiCall",
"recipientAccountId": "111111111111"
}
```

Example AWS service AWS STS API event in CloudTrail log file

The following example shows a CloudTrail log entry for a request made by an AWS service calling another service API using permissions from a service role. It shows the CloudTrail log entry for the request made in account 777788889999.

```
{
    "eventVersion": "1.04",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AIDAQRSTUVWXYZEXAMPLE:devdsk",
        "arn": "arn:aws:sts::777788889999:assumed-role/AssumeNothing/devdsk",
        "accountId": "777788889999",
        "accessKeyId": "AKIAQRSTUVWXYZEXAMPLE",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2016-11-14T17:25:26Z"
            },
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AIDAQRSTUVWXYZEXAMPLE",
                "arn": "arn:aws:iam::777788889999:role/AssumeNothing",
                "accountId": "777788889999",
                "userName": "AssumeNothing"
            }
        }
    },
    "eventTime": "2016-11-14T17:25:45Z",
    "eventSource": "s3.amazonaws.com",
    "eventName": "DeleteBucket",
    "awsRegion": "us-east-2",
    "sourceIPAddress": "192.0.2.1",
    "userAgent": "[aws-cli/1.11.10 Python/2.7.8 Linux/3.2.45-0.6.wd.865.49.315.metal1.x86_64 botocore/1.4.67]",
    "requestParameters": {
        "bucketName": "my-test-bucket-cross-account"
    },
    "responseElements": null,
    "requestID": "EXAMPLE463D56D4C",
    "eventID": "dEXAMPLE-265a-41e0-9352-4401bEXAMPLE",
    "eventType": "AwsApiCall",
    "recipientAccountId": "777788889999"
}
```

Example SAML AWS STS API event in CloudTrail log file

The following example shows a CloudTrail log entry for a request made for the AWS STS AssumeRoleWithSAML action. The request includes the SAML attributes CostCenter and Project that are passed through the SAML assertion as [session tags \(p. 293\)](#). Those tags are set as transitive so that they [persist in role chaining scenarios \(p. 299\)](#).

```
{
    "eventVersion": "1.05",
```

```

"userIdentity": {
    "type": "SAMLUser",
    "principalId": "SampleUkh1i4+ExampLexL/jEvs=:SamlExample",
    "userName": "SamlExample",
    "identityProvider": "bdGOnTesti4+ExampLexL/jEvs="
},
"eventTime": "2019-11-01T19:14:36Z",
"eventSource": "sts.amazonaws.com",
"eventName": "AssumeRoleWithSAML",
"awsRegion": "us-east-2",
"sourceIPAddress": "192.0.2.101",
"userAgent": "aws-cli/1.16.263 Python/3.4.7
Linux/4.9.184-0.1.ac.235.83.329.metal1.x86_64 botocore/1.12.253",
"requestParameters": {
    "SAMLAssertionID": "_c0046cEXAMPLEb9d4b8eEXAMPLE2619aEXAMPLE",
    "roleSessionName": "MyAssignedRoleSessionName",
    "principalTags": {
        "CostCenter": "987654",
        "Project": "Unicorn",
        "Department": "Engineering"
    },
    "transitiveTagKeys": [
        "CostCenter",
        "Project"
    ],
    "durationSeconds": 3600,
    "roleArn": "arn:aws:iam::444455556666:role/SAMLTestRoleShibboleth",
    "principalArn": "arn:aws:iam::444455556666:saml-provider/Shibboleth"
},
"responseElements": {
    "subjectType": "transient",
    "issuer": "https://server.example.com/idp/shibboleth",
    "credentials": {
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "expiration": "Mar 23, 2016 2:39:57 AM",
        "sessionToken": "<encoded session token blob>"
    },
    "nameQualifier": "bdGOnTesti4+ExampLexL/jEvs=",
    "assumedRoleUser": {
        "assumedRoleId": "AROAD35QRSTUVWEXAMPLE:MyAssignedRoleSessionName",
        "arn": "arn:aws:sts::444455556666:assumed-role/SAMLTestRoleShibboleth/
MyAssignedRoleSessionName"
    },
    "subject": "SamlExample",
    "audience": "https://signin.aws.amazon.com/saml"
},
"resources": [
{
    "ARN": "arn:aws:iam::444455556666:role/SAMLTestRoleShibboleth",
    "accountId": "444455556666",
    "type": "AWS::IAM::Role"
},
{
    "ARN": "arn:aws:iam::444455556666:saml-provider/test-saml-provider",
    "accountId": "444455556666",
    "type": "AWS::IAM::SAMLProvider"
}
],
"requestID": "6EXAMPLE-e595-11e5-b2c7-c974fEXAMPLE",
"eventID": "dEXAMPLE-265a-41e0-9352-4401bEXAMPLE",
"eventType": "AwsApiCall",
"recipientAccountId": "444455556666"
}

```

Example web identity AWS STS API event in CloudTrail log file

The following example shows a CloudTrail log entry for a request made for the AWS STS AssumeRoleWithWebIdentity action. The request includes the attributes CostCenter and Project that are passed through the identity provider token as [session tags \(p. 293\)](#). Those tags are set as transitive so that they [persist in role chaining scenarios \(p. 299\)](#).

```
{  
    "eventVersion": "1.05",  
    "userIdentity": {  
        "type": "WebIdentityUser",  
        "principalId": "accounts.google.com:<id-of-application>.apps.googleusercontent.com:<id-of-user>",  
        "userName": "<id of user>",  
        "identityProvider": "accounts.google.com"  
    },  
    "eventTime": "2016-03-23T01:39:51Z",  
    "eventSource": "sts.amazonaws.com",  
    "eventName": "AssumeRoleWithWebIdentity",  
    "awsRegion": "us-east-2",  
    "sourceIPAddress": "192.0.2.101",  
    "userAgent": "aws-cli/1.3.23 Python/2.7.6 Linux/2.6.18-164.el5",  
    "requestParameters": {  
        "durationSeconds": 3600,  
        "roleArn": "arn:aws:iam::444455556666:role/FederatedWebIdentityRole",  
        "roleSessionName": "MyAssignedRoleSessionName"  
        "principalTags": {  
            "CostCenter": "24680",  
            "Project": "Pegasus"  
        },  
        "transitiveTagKeys": [  
            "CostCenter",  
            "Project"  
        ],  
    },  
    "responseElements": {  
        "provider": "accounts.google.com",  
        "subjectFromWebIdentityToken": "<id of user>",  
        "audience": "<id of application>.apps.googleusercontent.com",  
        "credentials": {  
            "accessKeyId": "ASIAQQRSTUVWRDAOEXAMPLE",  
            "expiration": "Mar 23, 2016 2:39:51 AM",  
            "sessionToken": "<encoded session token blob>"  
        },  
        "assumedRoleUser": {  
            "assumedRoleId": "AROACQRSTUVWRDAOEXAMPLE:MyAssignedRoleSessionName",  
            "arn": "arn:aws:sts::444455556666:assumed-role/FederatedWebIdentityRole/  
MyAssignedRoleSessionName"  
        }  
    },  
    "resources": [  
    {  
        "ARN": "arn:aws:iam::444455556666:role/FederatedWebIdentityRole",  
        "accountId": "444455556666",  
        "type": "AWS::IAM::Role"  
    }  
],  
    "requestID": "6EXAMPLE-e595-11e5-b2c7-c974fEXAMPLE",  
    "eventID": "bEXAMPLE-0b30-4246-b28c-e3da3EXAMPLE",  
    "eventType": "AwsApiCall",  
    "recipientAccountId": "444455556666"  
}
```

Example sign-in events in CloudTrail log

CloudTrail log files contain events that are formatted using JSON. A sign-in event represents a single sign-in request and includes information about the sign-in principal, the Region, and the date and time of the action.

Example sign-in success event in CloudTrail log file

The following example shows a CloudTrail log entry for a successful sign-in event.

```
{  
    "eventVersion": "1.05",  
    "userIdentity": {  
        "type": "IAMUser",  
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",  
        "arn": "arn:aws:iam::111122223333:user/JohnDoe",  
        "accountId": "111122223333",  
        "userName": "JohnDoe"  
    },  
    "eventTime": "2014-07-16T15:49:27Z",  
    "eventSource": "signin.amazonaws.com",  
    "eventName": "ConsoleLogin",  
    "awsRegion": "us-east-2",  
    "sourceIPAddress": "192.0.2.110",  
    "userAgent": "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:24.0) Gecko/20100101 Firefox/24.0",  
    "requestParameters": null,  
    "responseElements": {  
        "ConsoleLogin": "Success"  
    },  
    "additionalEventData": {  
        "MobileVersion": "No",  
        "LoginTo": "https://console.aws.amazon.com/s3/",  
        "MFAUsed": "No"  
    },  
    "eventID": "3fcfb182-98f8-4744-bd45-10a395ab61cb"  
}
```

For more details about the information contained in CloudTrail log files, see [CloudTrail Event Reference](#) in the [AWS CloudTrail User Guide](#).

Example sign-in failure event in CloudTrail log file

The following example shows a CloudTrail log entry for a failed sign-in event.

```
{  
    "eventVersion": "1.05",  
    "userIdentity": {  
        "type": "IAMUser",  
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",  
        "arn": "arn:aws:iam::111122223333:user/JaneDoe",  
        "accountId": "111122223333",  
        "userName": "JaneDoe"  
    },  
    "eventTime": "2014-07-08T17:35:27Z",  
    "eventSource": "signin.amazonaws.com",  
    "eventName": "ConsoleLogin",  
    "awsRegion": "us-east-2",  
    "sourceIPAddress": "192.0.2.100",  
    "userAgent": "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:24.0) Gecko/20100101 Firefox/24.0",  
    "errorMessage": "Failed authentication",  
    "requestParameters": null,  
}
```

```
"responseElements": {  
    "ConsoleLogin": "Failure"  
},  
"additionalEventData": {  
    "MobileVersion": "No",  
    "LoginTo": "https://console.aws.amazon.com/sns",  
    "MFAUsed": "No"  
},  
"eventID": "11ea990b-4678-4bcd-8fbe-62509088b7cf"  
}
```

From this information, you can determine that the sign-in attempt was made by an IAM user named JaneDoe, as shown in the `userIdentity` element. You can also see that the sign-in attempt failed, as shown in the `responseElements` element. You can see that JaneDoe tried to sign in to the Amazon SNS console at 5:35 PM (UTC) on July 8, 2014.

Example sign-in failure event caused by incorrect user name

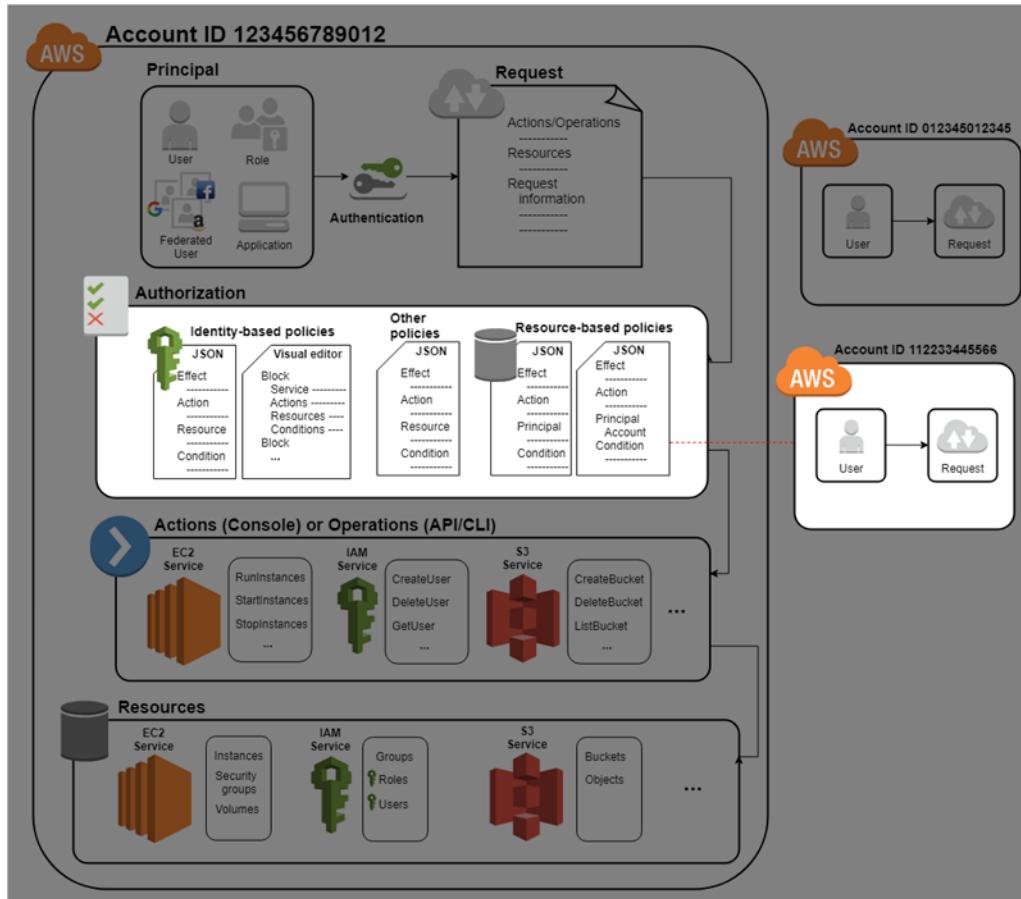
The following example shows a CloudTrail log entry for an unsuccessful sign-in event caused by the user entering an incorrect user name. AWS masks the `userName` text with `HIDDEN_DUE_TO_SECURITY_REASONs` to help prevent exposing potentially sensitive information.

```
{  
    "eventVersion": "1.05",  
    "userIdentity": {  
        "type": "IAMUser",  
        "accountId": "123456789012",  
        "accessKeyId": "",  
        "userName": "HIDDEN_DUE_TO_SECURITY_REASONs"  
    },  
    "eventTime": "2015-03-31T22:20:42Z",  
    "eventSource": "signin.amazonaws.com",  
    "eventName": "ConsoleLogin",  
    "awsRegion": "us-east-2",  
    "sourceIPAddress": "192.0.2.101",  
    "userAgent": "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:24.0) Gecko/20100101 Firefox/24.0",  
    "errorMessage": "No username found in supplied account",  
    "requestParameters": null,  
    "responseElements": {  
        "ConsoleLogin": "Failure"  
    },  
    "additionalEventData": {  
        "LoginTo": "https://console.aws.amazon.com/console/home?state=hashArgs%23&isauthcode=true",  
        "MobileVersion": "No",  
        "MFAUsed": "No"  
    },  
    "eventID": "a7654656-0417-45c6-9386-ea8231385051",  
    "eventType": "AwsConsoleSignin",  
    "recipientAccountId": "123456789012"  
}
```

Access management for AWS resources

AWS Identity and Access Management (IAM) is a web service that helps you securely control access to AWS resources. When a [principal \(p. 5\)](#) makes a request in AWS, the AWS enforcement code checks whether the principal is authenticated (signed in) and authorized (has permissions). You manage access in AWS by creating policies and attaching them to IAM identities or AWS resources. Policies are JSON documents in AWS that, when attached to an identity or resource, define their permissions. For more information about policy types and uses, see [Policies and permissions in IAM \(p. 351\)](#).

For details about the rest of the authentication and authorization process, see [Understanding how IAM works \(p. 3\)](#).



During authorization, the AWS enforcement code uses values from the [request context \(p. 5\)](#) to check for matching policies and determine whether to allow or deny the request.

AWS checks each policy that applies to the context of the request. If a single policy denies the request, AWS denies the entire request and stops evaluating policies. This is called an *explicit deny*. Because

requests are *denied by default*, IAM authorizes your request only if every part of your request is allowed by the applicable policies. The [evaluation logic \(p. 666\)](#) for a request within a single account follows these rules:

- By default, all requests are implicitly denied. (Alternatively, by default, the AWS account root user has full access.)
- An explicit allow in an identity-based or resource-based policy overrides this default.
- If a permissions boundary, Organizations SCP, or session policy is present, it might override the allow with an implicit deny.
- An explicit deny in any policy overrides any allows.

After your request has been authenticated and authorized, AWS approves the request. If you need to make a request in a different account, a policy in the other account must allow you to access the resource. In addition, the IAM entity that you use to make the request must have an identity-based policy that allows the request.

Access management resources

For more information about permissions and about creating policies, see the following resources:

The following entries in the AWS Security Blog cover common ways to write policies for access to Amazon S3 buckets and objects.

- [Writing IAM Policies: How to Grant Access to an Amazon S3 Bucket](#)
- [Writing IAM policies: Grant Access to User-Specific Folders in an Amazon S3 Bucket](#)
- [IAM Policies and Bucket Policies and ACLs! Oh My! \(Controlling Access to S3 Resources\)](#)
- [A Primer on RDS Resource-Level Permissions](#)
- [Demystifying EC2 Resource-Level Permissions](#)

Policies and permissions in IAM

You manage access in AWS by creating policies and attaching them to IAM identities (users, groups of users, or roles) or AWS resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when an IAM principal (user or role) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. AWS supports six types of policies: identity-based policies, resource-based policies, permissions boundaries, Organizations SCPs, ACLs, and session policies.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, if a policy allows the [GetUser](#) action, then a user with that policy can get user information from the AWS Management Console, the AWS CLI, or the AWS API. When you create an IAM user, you can choose to allow console or programmatic access. If console access is allowed, the IAM user can sign in to the console using a user name and password. Or if programmatic access is allowed, the user can use access keys to work with the CLI or API.

Policy types

The following policy types, listed in order of frequency, are available for use in AWS. For more details, see the sections below for each policy type.

- **Identity-based policies (p. 352)** – Attach managed and inline policies to IAM identities (users, groups to which users belong, or roles). Identity-based policies grant permissions to an identity.
- **Resource-based policies (p. 352)** – Attach inline policies to resources. The most common examples of resource-based policies are Amazon S3 bucket policies and IAM role trust policies. Resource-based policies grant permissions to the principal that is specified in the policy. Principals can be in the same account as the resource or in other accounts.
- **Permissions boundaries (p. 353)** – Use a managed policy as the permissions boundary for an IAM entity (user or role). That policy defines the maximum permissions that the identity-based policies can grant to an entity, but does not grant permissions. Permissions boundaries do not define the maximum permissions that a resource-based policy can grant to an entity.
- **Organizations SCPs (p. 353)** – Use an AWS Organizations service control policy (SCP) to define the maximum permissions for account members of an organization or organizational unit (OU). SCPs limit permissions that identity-based policies or resource-based policies grant to entities (users or roles) within the account, but do not grant permissions.
- **Access control lists (ACLs) (p. 353)** – Use ACLs to control which principals in other accounts can access the resource to which the ACL is attached. ACLs are similar to resource-based policies, although they are the only policy type that does not use the JSON policy document structure. ACLs are cross-account permissions policies that grant permissions to the specified principal. ACLs cannot grant permissions to entities within the same account.
- **Session policies (p. 353)** – Pass advanced session policies when you use the AWS CLI or AWS API to assume a role or a federated user. Session policies limit the permissions that the role or user's identity-based policies grant to the session. Session policies limit permissions for a created session, but do not grant permissions. For more information, see [Session Policies](#).

Identity-based policies

Identity-based policies are JSON permissions policy documents that control what actions an identity (users, groups of users, and roles) can perform, on which resources, and under what conditions. Identity-based policies can be further categorized:

- **Managed policies** – Standalone identity-based policies that you can attach to multiple users, groups, and roles in your AWS account. There are two types of managed policies:
 - **AWS managed policies** – Managed policies that are created and managed by AWS.
 - **Customer managed policies** – Managed policies that you create and manage in your AWS account. Customer managed policies provide more precise control over your policies than AWS managed policies.
- **Inline policies** – Policies that you add directly to a single user, group, or role. Inline policies maintain a strict one-to-one relationship between a policy and an identity. They are deleted when you delete the identity.

To learn how to choose between managed and inline policies, see [Choosing between managed policies and inline policies \(p. 363\)](#).

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource such as an Amazon S3 bucket. These policies grant the specified principal permission to perform specific actions on that resource and defines under what conditions this applies. Resource-based policies are inline policies. There are no managed resource-based policies.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in separate

AWS accounts, you must also use an identity-based policy to grant the principal access to the resource. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required.

The IAM service supports only one type of resource-based policy called a role *trust policy*, which is attached to an IAM role. An IAM role is both an identity and a resource that supports resource-based policies. For that reason, you must attach both a trust policy and an identity-based policy to an IAM role. Trust policies define which principal entities (accounts, users, roles, and federated users) can assume the role. To learn how IAM roles are different from other resource-based policies, see [How IAM roles differ from resource-based policies \(p. 286\)](#).

To see which other services support resource-based policies, see [AWS services that work with IAM \(p. 611\)](#). To learn more about resource-based policies, see [Identity-based policies and resource-based policies \(p. 374\)](#). To learn whether principals in accounts outside of your zone of trust (trusted organization or account) have access to assume your roles, see [What is IAM Access Analyzer?](#).

IAM permissions boundaries

A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity. When you set a permissions boundary for an entity, the entity can perform only the actions that are allowed by both its identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role as the principal are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see [Permissions boundaries for IAM entities \(p. 365\)](#).

Service control policies (SCPs)

AWS Organizations is a service for grouping and centrally managing the AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU). The SCP limits permissions for entities in member accounts, including each AWS account root user. An explicit deny in any of these policies overrides the allow.

For more information about Organizations and SCPs, see [How SCPs Work](#) in the *AWS Organizations User Guide*.

Access control lists (ACLs)

Access control lists (ACLs) are service policies that allow you to control which principals in another account can access a resource. ACLs cannot be used to control access for a principal within the same account. ACLs are similar to resource-based policies, although they are the only policy type that does not use the JSON policy document format. Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see [Access Control List \(ACL\) Overview](#) in the *Amazon Simple Storage Service Developer Guide*.

Session policies

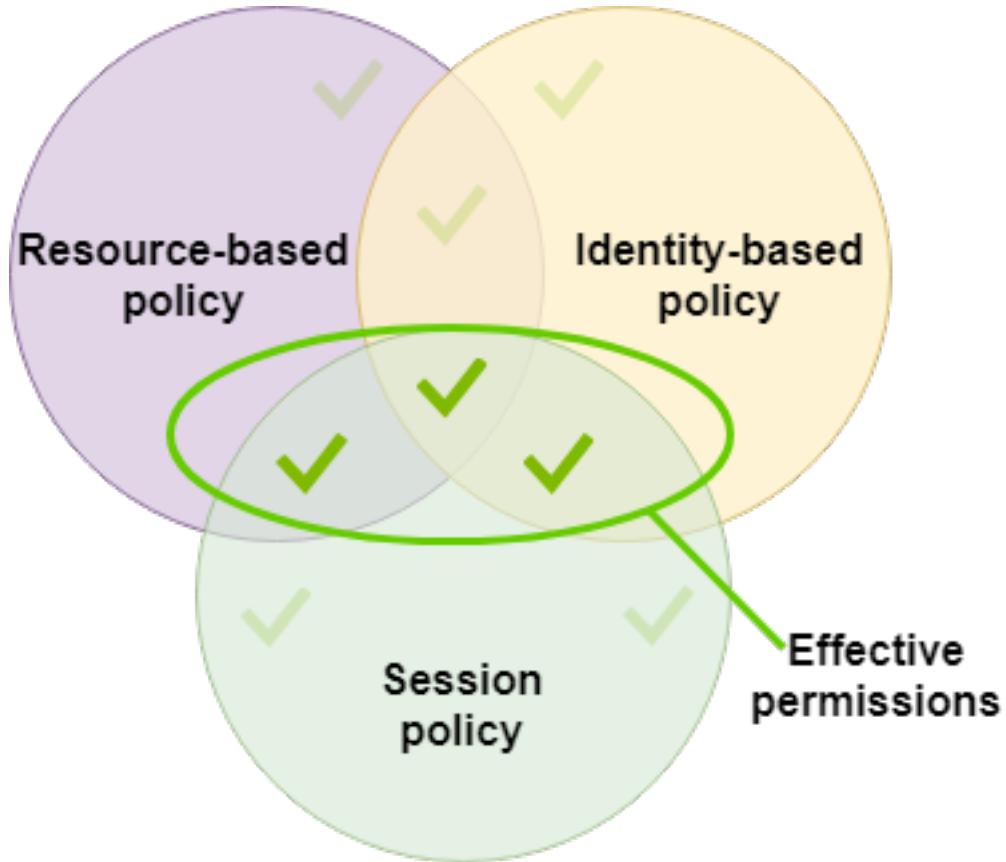
Session policies are advanced policies that you pass in a parameter when you programmatically create a temporary session for a role or federated user. The permissions for a session are the intersection of the identity-based policies for the IAM entity (user or role) used to create the session and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow.

You can create role session and pass session policies programmatically using the `AssumeRole`, `AssumeRoleWithSAML`, or `AssumeRoleWithWebIdentity` API operations. You can pass a single JSON

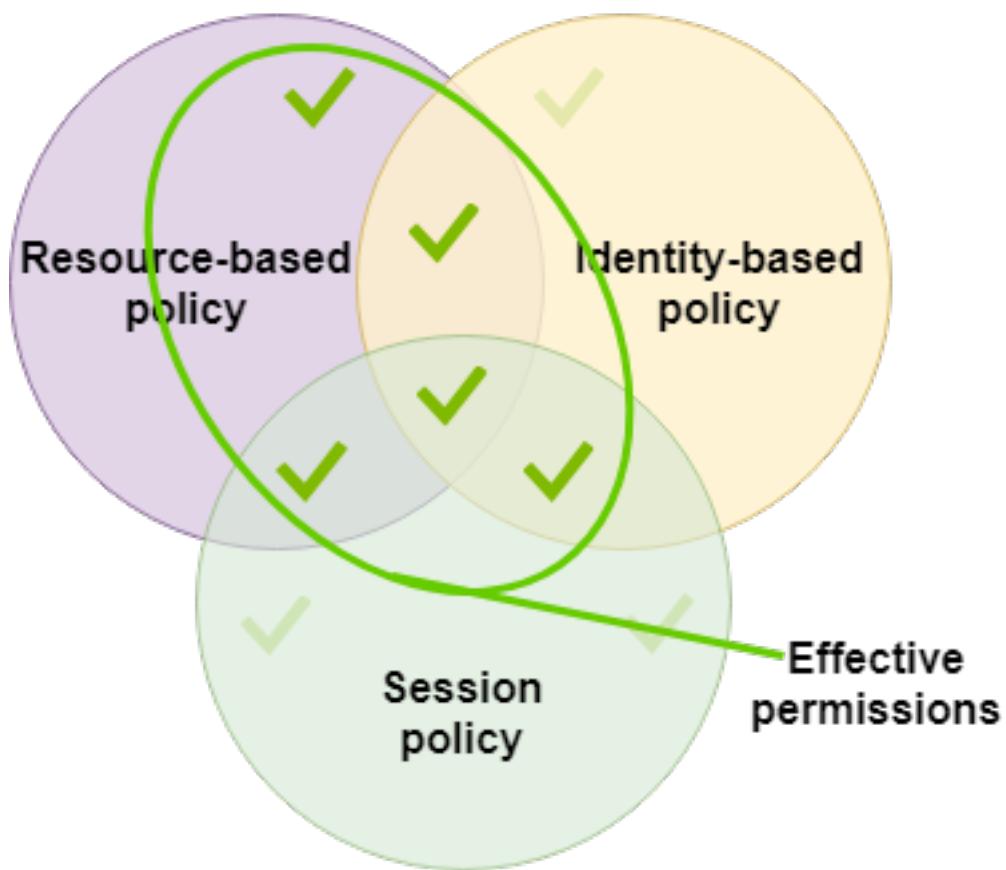
inline session policy document using the `Policy` parameter. You can use the `PolicyArns` parameter to specify up to 10 managed session policies. For more information about creating a role session, see [Requesting temporary security credentials \(p. 303\)](#).

When you create a federated user session, you use an IAM user's access keys to programmatically call the `GetFederationToken` API operation. You must also pass session policies. The resulting session's permissions are the intersection of the IAM user's identity-based policy and the session policy. For more information about creating a federated user session, see [GetFederationToken—federation through a custom identity broker \(p. 308\)](#).

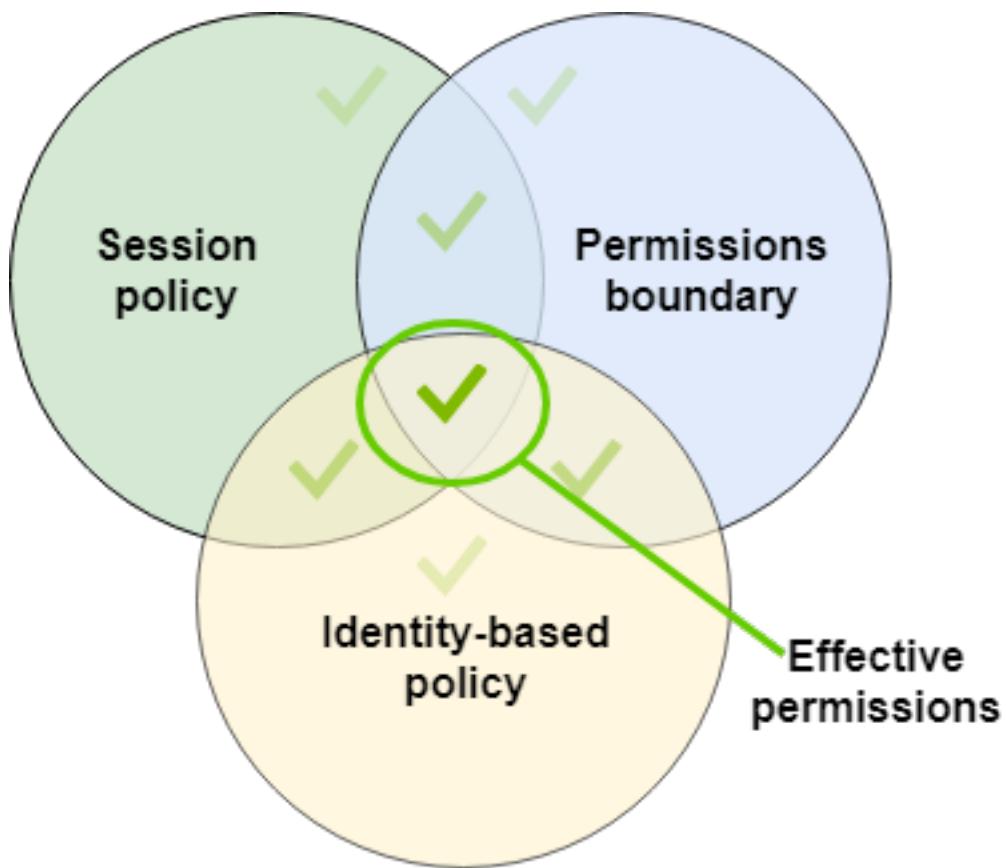
A resource-based policy can specify the ARN of the user or role as a principal. In that case, the permissions from the resource-based policy are added to the role or user's identity-based policy before the session is created. The session policy limits the total permissions granted by the resource-based policy and the identity-based policy. The resulting session's permissions are the intersection of the session policies and either the resource-based policy or the identity-based policy.



A resource-based policy can specify the ARN of the session as a principal. In that case, the permissions from the resource-based policy are added after the session is created. The resource-based policy permissions are not limited by the session policy. The resulting session has all the permissions of the resource-based policy *plus* the intersection of the identity-based policy and the session policy.



A permissions boundary can set the maximum permissions for a user or role that is used to create a session. In that case, the resulting session's permissions are the intersection of the session policy, the permissions boundary, and the identity-based policy. However, a permissions boundary does not limit permissions granted by a resource-based policy that specifies the ARN of the resulting session.



Policies and the root user

The AWS account root user is affected by some policy types but not others. You cannot attach identity-based policies to the root user, and you cannot set the permissions boundary for the root user. However, you can specify the root user as the principal in a resource-based policy or an ACL. As a member of an account, the root user is affected by any SCPs for the account.

Overview of JSON policies

Most policies are stored in AWS as JSON documents. Identity-based policies and policies used to set permissions boundaries are JSON policy documents that you attach to a user or role. Resource-based policies are JSON policy documents that you attach to a resource. SCPs are JSON policy documents with restricted syntax that you attach to an AWS Organizations organizational unit (OU). ACLs are also attached to a resource, but you must use a different syntax. Session policies are JSON policies that you provide when you assume a role or federated user session.

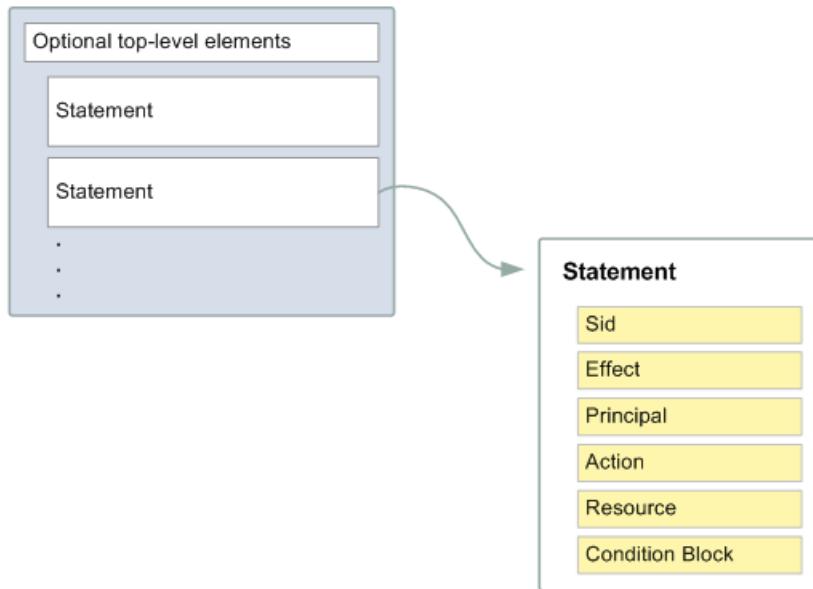
It is not necessary for you to understand the JSON syntax. You can use the visual editor in the AWS Management Console to create and edit customer managed policies without ever using JSON. However, if you use inline policies for groups or complex policies, you must still create and edit those policies in the JSON editor using the console. For more information about using the visual editor, see [Creating IAM policies \(p. 439\)](#) and [Editing IAM policies \(p. 465\)](#).

JSON policy document structure

As illustrated in the following figure, a JSON policy document includes these elements:

- Optional policy-wide information at the top of the document
- One or more individual statements

Each statement includes information about a single permission. If a policy includes multiple statements, AWS applies a logical OR across the statements when evaluating them. If multiple policies apply to a request, AWS applies a logical OR across all of those policies when evaluating them.



The information in a statement is contained within a series of elements.

- **Version** – Specify the version of the policy language that you want to use. As a best practice, use the latest 2012-10-17 version.
- **Statement** – Use this main policy element as a container for the following elements. You can include more than one statement in a policy.
- **Sid** (Optional) – Include an optional statement ID to differentiate between your statements.
- **Effect** – Use Allow or Deny to indicate whether the policy allows or denies access.
- **Principal** (Required in only some circumstances) – If you create a resource-based policy, you must indicate the account, user, role, or federated user to which you would like to allow or deny access. If you are creating an IAM permissions policy to attach to a user or role, you cannot include this element. The principal is implied as that user or role.
- **Action** – Include a list of actions that the policy allows or denies.
- **Resource** (Required in only some circumstances) – If you create an IAM permissions policy, you must specify a list of resources to which the actions apply. If you create a resource-based policy, this element is optional. If you do not include this element, then the resource to which the action applies is the resource to which the policy is attached.
- **Condition** (Optional) – Specify the circumstances under which the policy grants permission.

To learn about these and other more advanced policy elements, see [IAM JSON policy elements reference \(p. 628\)](#).

Multiple statements and multiple policies

If you want to define more than one permission for an entity (user or role), you can use multiple statements in a single policy. You can also attach multiple policies. If you try to define multiple permissions in a single statement, your policy might not grant the access that you expect. As a best practice, break up policies by resource type.

Because of the [limited size of policies \(p. 606\)](#), it might be necessary to use multiple policies for more complex permissions. It's also a good idea to create functional groupings of permissions in a separate customer managed policy. For example, Create one policy for IAM user management, one for self-management, and another policy for S3 bucket management. Regardless of the combination of multiple statements and multiple policies, AWS [evaluates \(p. 666\)](#) your policies the same way.

For example, the following policy has three statements, each of which defines a separate set of permissions within a single account. The statements define the following:

- The first statement, with an `Sid` (Statement ID) of `FirstStatement`, lets the user with the attached policy change their own password. The `Resource` element in this statement is `"*"` (which means "all resources"). But in practice, the `ChangePassword` API operation (or equivalent `change-password` CLI command) affects only the password for the user who makes the request.
- The second statement lets the user list all the Amazon S3 buckets in their AWS account. The `Resource` element in this statement is `"*"` (which means "all resources"). But because policies don't grant access to resources in other accounts, the user can list only the buckets in their own AWS account.
- The third statement lets the user list and retrieve any object that is in a bucket named `confidential-data`, but only when the user is authenticated with multi-factor authentication (MFA). The `Condition` element in the policy enforces the MFA authentication.

When a policy statement contains a `Condition` element, the statement is only in effect when the `Condition` element evaluates to true. In this case, the `Condition` evaluates to true when the user is MFA-authenticated. If the user is not MFA-authenticated, this `Condition` evaluates to false. In that case, the third statement in this policy does not apply and the user does not have access to the `confidential-data` bucket.

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Sid": "FirstStatement",  
      "Effect": "Allow",  
      "Action": ["iam:ChangePassword"],  
      "Resource": "*"  
    },  
    {  
      "Sid": "SecondStatement",  
      "Effect": "Allow",  
      "Action": "s3>ListAllMyBuckets",  
      "Resource": "*"  
    },  
    {  
      "Sid": "ThirdStatement",  
      "Effect": "Allow",  
      "Action": [  
        "s3>List*",  
        "s3:Get*"  
      ],  
      "Resource": [  
        "arn:aws:s3:::confidential-data",  
        "arn:aws:s3:::confidential-data/*"  
      ]  
    }  
  ]  
}
```

```
        ],
        "Condition": {"Bool": {"aws:MultiFactorAuthPresent": "true"}}
    }
}
```

Examples of JSON policy syntax

The following identity-based policy allows the implied principal to list a single Amazon S3 bucket named `example_bucket`:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3>ListBucket",
            "Resource": "arn:aws:s3:::example_bucket"
        }
    ]
}
```

The following resource-based policy can be attached to an Amazon S3 bucket. The policy allows members of a specific AWS account to perform any Amazon S3 actions in the bucket named `mybucket`. It allows any action that can be performed on a bucket or the objects within it. (Because the policy grants trust only to the account, individual users in the account must still be granted permissions for the specified Amazon S3 actions.)

```
{
    "Version": "2012-10-17",
    "Id": "S3-Account-Permissions",
    "Statement": [
        {
            "Sid": "1",
            "Effect": "Allow",
            "Principal": {"AWS": ["arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:root"]},
            "Action": "s3:*",
            "Resource": [
                "arn:aws:s3:::mybucket",
                "arn:aws:s3:::mybucket/*"
            ]
        }
    ]
}
```

To view example policies for common scenarios, see [Example IAM identity-based policies \(p. 389\)](#).

Managed policies and inline policies

When you need to set the permissions for an identity in IAM, you must decide whether to use an AWS managed policy, a customer managed policy, or an inline policy. The following sections provide more information about each of the types of identity-based policies and when to use them.

Topics

- [AWS managed policies \(p. 360\)](#)
- [Customer managed policies \(p. 361\)](#)
- [Inline policies \(p. 362\)](#)
- [Choosing between managed policies and inline policies \(p. 363\)](#)
- [Deprecated AWS managed policies \(p. 364\)](#)

AWS managed policies

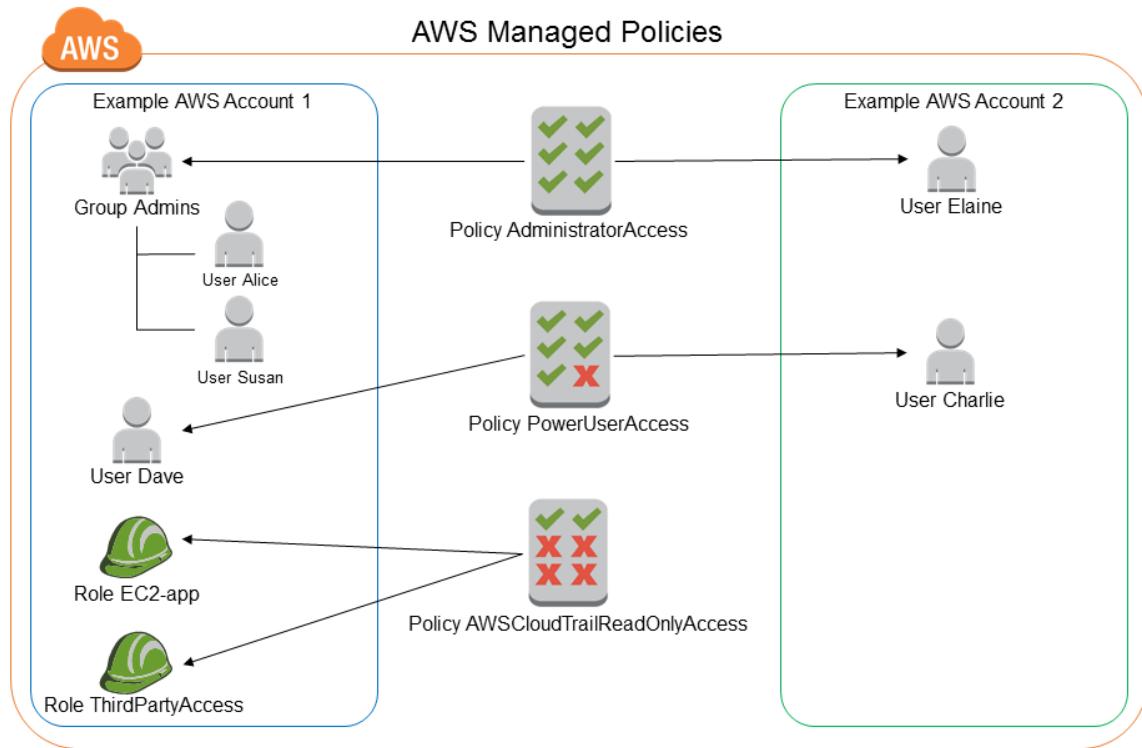
An *AWS managed policy* is a standalone policy that is created and administered by AWS. *Standalone policy* means that the policy has its own Amazon Resource Name (ARN) that includes the policy name. For example, `arn:aws:iam::aws:policy/IAMReadOnlyAccess` is an AWS managed policy. For more information about ARNs, see [IAM ARNs \(p. 601\)](#).

AWS managed policies are designed to provide permissions for many common use cases. Full access AWS managed policies such as [AmazonDynamoDBFullAccess](#) and [IAMFullAccess](#) define permissions for service administrators by granting full access to a service. Power-user AWS managed policies such as [AWSCodeCommitPowerUser](#) and [AWSKeyManagementServicePowerUser](#) are designed for power users. Partial-access AWS managed policies such as [AmazonMobileAnalyticsWriteOnlyAccess](#) and [AmazonEC2ReadOnlyAccess](#) provide specific levels of access to AWS services without allowing [permissions management \(p. 501\)](#) access level permissions. AWS managed policies make it easier for you to assign appropriate permissions to users, groups, and roles than if you had to write the policies yourself.

One particularly useful category of AWS managed policies are those designed for job functions. These policies align closely to commonly used job functions in the IT industry. The intent is to make granting permissions for these common job functions easy. One key advantage of using job function policies is that they are maintained and updated by AWS as new services and API operations are introduced. For example, the [AdministratorAccess](#) job function provides full access and permissions delegation to every service and resource in AWS. We recommend that this policy is used only for the account administrator. For power users that require full access to every service except limited access to IAM and Organizations, use the [PowerUserAccess](#) job function. For a list and descriptions of the job function policies, see [AWS managed policies for job functions \(p. 684\)](#).

You cannot change the permissions defined in AWS managed policies. AWS occasionally updates the permissions defined in an AWS managed policy. When AWS does this, the update affects all principal entities (users, groups, and roles) that the policy is attached to. AWS is most likely to update an AWS managed policy when a new AWS service is launched or new API calls become available for existing services. For example, the AWS managed policy called **ReadOnlyAccess** provides read-only access to all AWS services and resources. When AWS launches a new service, AWS updates the **ReadOnlyAccess** policy to add read-only permissions for the new service. The updated permissions are applied to all principal entities that the policy is attached to.

The following diagram illustrates AWS managed policies. The diagram shows three AWS managed policies: **AdministratorAccess**, **PowerUserAccess**, and **AWSCloudTrailReadOnlyAccess**. Notice that a single AWS managed policy can be attached to principal entities in different AWS accounts, and to different principal entities in a single AWS account.

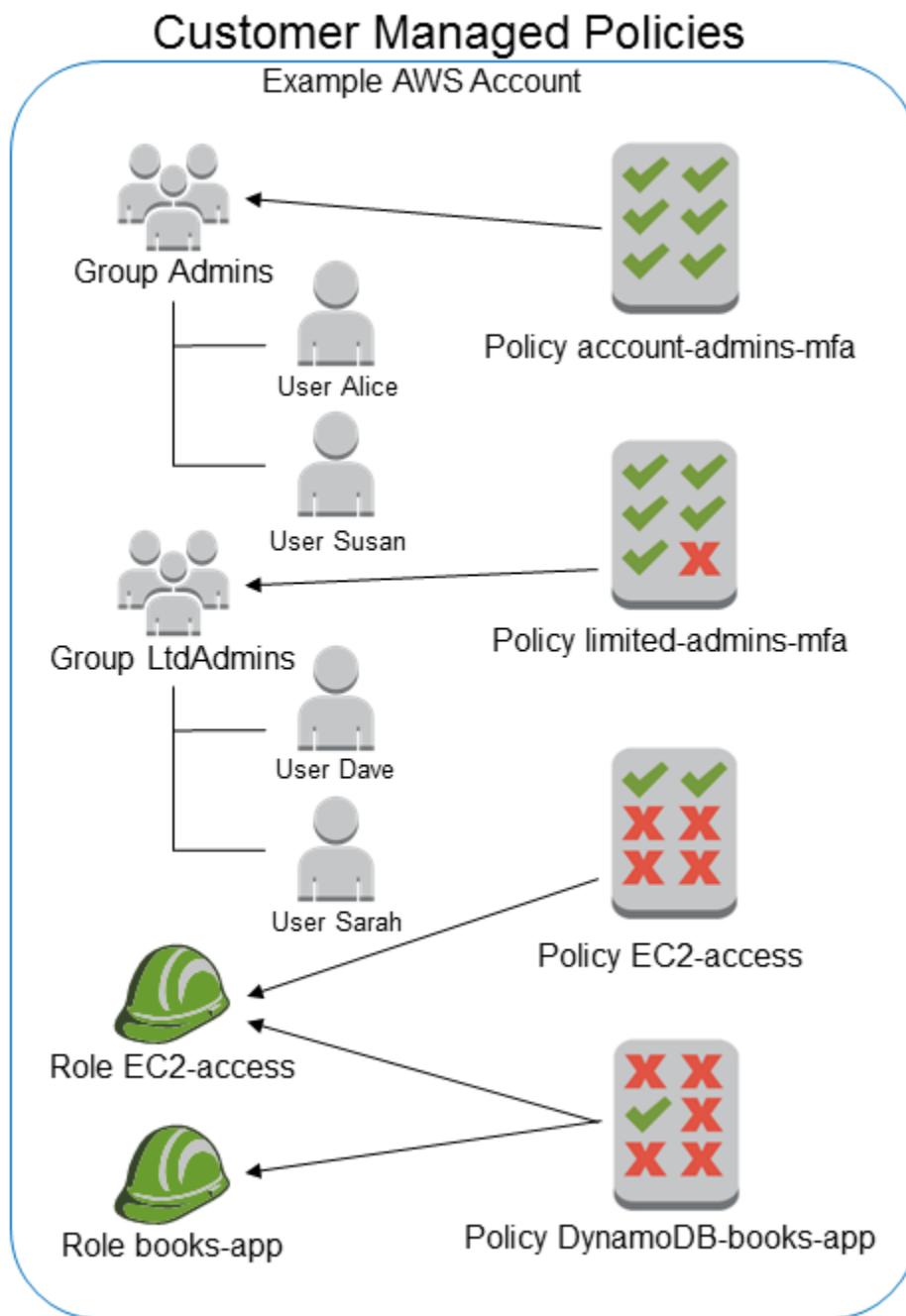


Customer managed policies

You can create standalone policies that you administer in your own AWS account, which we refer to as *customer managed policies*. You can then attach the policies to multiple principal entities in your AWS account. When you attach a policy to a principal entity, you give the entity the permissions that are defined in the policy.

A great way to create a customer managed policy is to start by copying an existing AWS managed policy. That way you know that the policy is correct at the beginning and all you need to do is customize it to your environment.

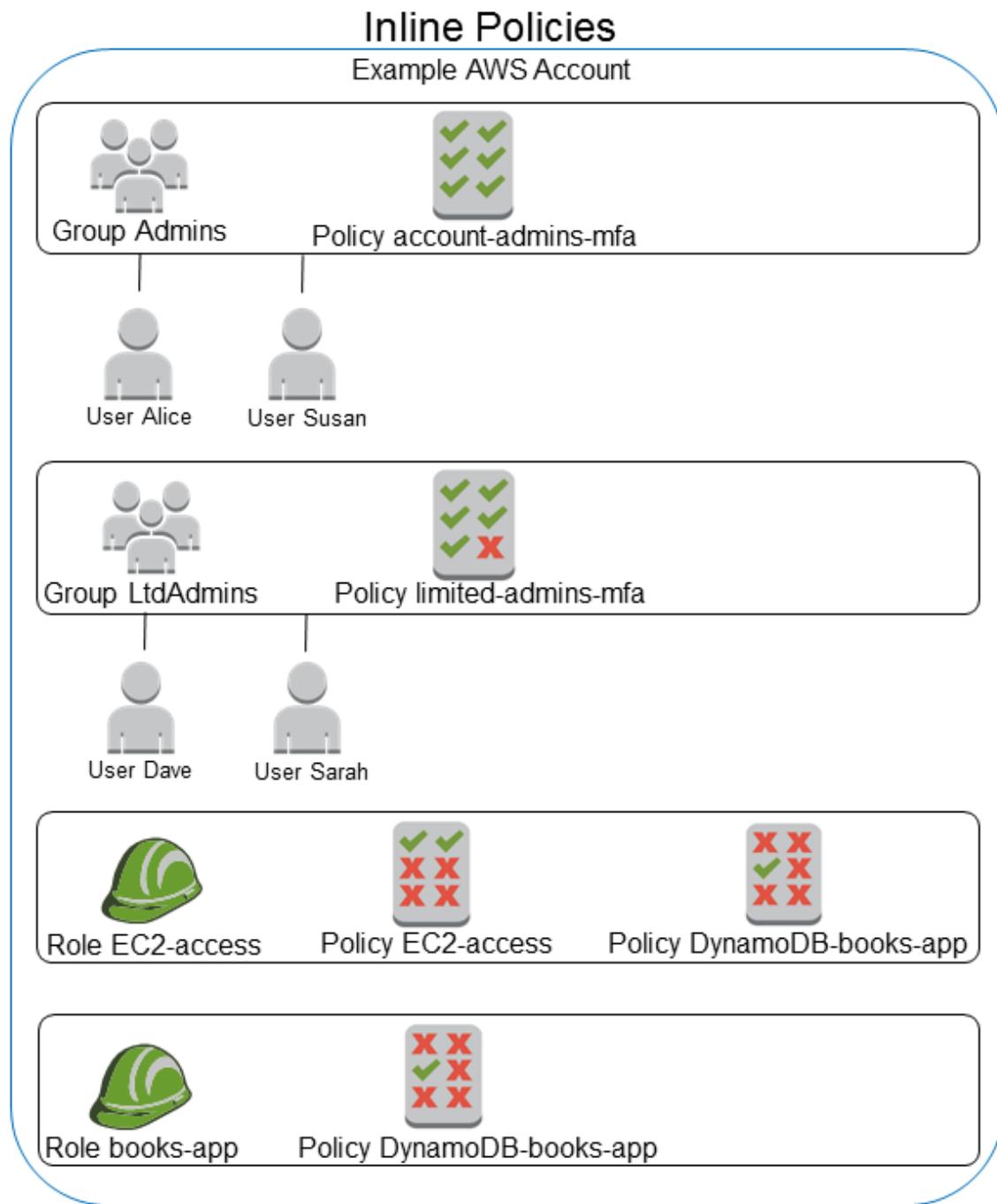
The following diagram illustrates customer managed policies. Each policy is an entity in IAM with its own [Amazon Resource Name \(ARN\)](#) (p. 601) that includes the policy name. Notice that the same policy can be attached to multiple principal entities—for example, the same **DynamoDB-books-app** policy is attached to two different IAM roles.



Inline policies

An inline policy is a policy that's embedded in an IAM identity (a user, group, or role). That is, the policy is an inherent part of the identity. You can create a policy and embed it in an identity, either when you create the identity or later.

The following diagram illustrates inline policies. Each policy is an inherent part of the user, group, or role. Notice that two roles include the same policy (the **DynamoDB-books-app** policy), but they are not sharing a single policy; each role has its own copy of the policy.



Choosing between managed policies and inline policies

The different types of policies are for different use cases. In most cases, we recommend that you use managed policies instead of inline policies.

Managed policies provide the following features:

Reusability

A single managed policy can be attached to multiple principal entities (users, groups, and roles). In effect, you can create a library of policies that define permissions that are useful for your AWS account, and then attach these policies to principal entities as needed.

Central change management

When you change a managed policy, the change is applied to all principal entities that the policy is attached to. For example, if you want to add permission for a new AWS API, you can update the managed policy to add the permission. (If you're using an AWS managed policy, AWS updates to the policy.) When the policy is updated, the changes are applied to all principal entities that the policy is attached to. In contrast, to change an inline policy you must individually edit each identity that contains the policy. For example, if a group and a role both contain the same inline policy, you must individually edit both principal entities in order to change that policy.

Versioning and rolling back

When you change a customer managed policy, the changed policy doesn't overwrite the existing policy. Instead, IAM creates a new version of the managed policy. IAM stores up to five versions of your customer managed policies. You can use policy versions to revert a policy to an earlier version if you need to.

A policy version is different from a `Version` policy element. The `Version` policy element is used within a policy and defines the version of the policy language. To learn more about policy versions, see [the section called "Versioning IAM policies" \(p. 462\)](#). To learn more about the `Version` policy element see [IAM JSON policy elements: Version \(p. 629\)](#).

Delegating permissions management

You can allow users in your AWS account to attach and detach policies while maintaining control over the permissions defined in those policies. In effect, you can designate some users as full administrators—that is, administrators that can create, update, and delete policies. You can then designate other users as limited administrators. That is, administrators that can attach policies to other principal entities, but only the policies that you have allowed them to attach.

For more information about delegating permissions management, see [Controlling access to policies \(p. 380\)](#).

Automatic updates for AWS managed policies

AWS maintains AWS managed policies and updates them when necessary (for example, to add permissions for new AWS services), without you having to make changes. The updates are automatically applied to the principal entities that you have attached the AWS managed policy to.

Using inline policies

Inline policies are useful if you want to maintain a strict one-to-one relationship between a policy and the identity that it's applied to. For example, you want to be sure that the permissions in a policy are not inadvertently assigned to an identity other than the one they're intended for. When you use an inline policy, the permissions in the policy cannot be inadvertently attached to the wrong identity. In addition, when you use the AWS Management Console to delete that identity, the policies embedded in the identity are deleted as well. That's because they are part of the principal entity.

Deprecated AWS managed policies

To simplify the assignment of permissions, AWS provides [managed policies \(p. 359\)](#)—predefined policies that are ready to be attached to your IAM users, groups, and roles.

Sometimes AWS needs to add a new permission to an existing policy, such as when a new service is introduced. Adding a new permission to an existing policy does not disrupt or remove any feature or ability.

However, AWS might choose to create a *new* policy when the needed changes could impact customers if they were applied to an existing policy. For example, removing permissions from an existing policy could break the permissions of any IAM entity or application that depended upon it, potentially disrupting a critical operation.

Therefore, when such a change is required, AWS creates a completely new policy with the required changes and makes it available to customers. The old policy is then marked *deprecated*. A deprecated managed policy appears with a warning icon next to it in the **Policies** list in the IAM console.

A deprecated policy has the following characteristics:

- It continues to work for all *currently* attached users, groups, and roles. Nothing breaks.
- It *cannot* be attached to any new users, groups, or roles. If you detach it from a current entity, you cannot reattach it.
- After you detach it from all current entities, it is no longer visible and can no longer be used in any way.

If any user, group, or role requires the policy, you must instead attach the new policy. When you receive notice that a policy is deprecated, we recommend that you immediately plan to attach all users, groups, and roles to the replacement policy and detach them from the deprecated policy. Continuing to use the deprecated policy can carry risks that are mitigated only by switching to the replacement policy.

Permissions boundaries for IAM entities

AWS supports *permissions boundaries* for IAM entities (users or roles). A permissions boundary is an advanced feature for using a managed policy to set the maximum permissions that an identity-based policy can grant to an IAM entity. An entity's permissions boundary allows it to perform only the actions that are allowed by both its identity-based policies and its permissions boundaries.

For more information about policy types, see [Policy types \(p. 351\)](#).

You can use an AWS managed policy or a customer managed policy to set the boundary for an IAM entity (user or role). That policy limits the maximum permissions for the user or role.

For example, assume that the IAM user named ShirleyRodriguez should be allowed to manage only Amazon S3, Amazon CloudWatch, and Amazon EC2. To enforce this rule, you can use the following policy to set the permissions boundary for the ShirleyRodriguez user:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:*",  
                "cloudwatch:*",  
                "ec2:/*"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

When you use a policy to set the permissions boundary for a user, it limits the user's permissions but does not provide permissions on its own. In this example, the policy sets the maximum permissions of ShirleyRodriguez as all operations in Amazon S3, CloudWatch, and Amazon EC2. Shirley can never perform operations in any other service, including IAM, even if she has a permissions policy that allows it. For example, you can add the following policy to the ShirleyRodriguez user:

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

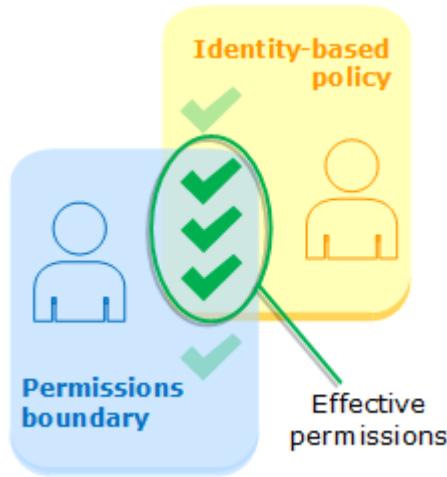
This policy allows creating a user in IAM. If you attach this permissions policy to the ShirleyRodriguez user, and Shirley tries to create a user, the operation fails. It fails because the permissions boundary does not allow the `iam:CreateUser` operation. Given these two policies, Shirley does not have permission to perform any operations in AWS. You must add a different permissions policy to allow actions in other services, such as Amazon S3. Alternatively, you could update the permissions boundary to allow her to create a user in IAM.

Evaluating effective permissions with boundaries

The permissions boundary for an IAM entity (user or role) sets the maximum permissions that the entity can have. This can change the effective permissions for that user or role. The effective permissions for an entity are the permissions that are granted by all the policies that affect the user or role. Within an account, the permissions for an entity can be affected by identity-based policies, resource-based policies, permissions boundaries, Organizations SCPs, or session policies. For more information about the different types of policies, see [Policies and permissions in IAM \(p. 351\)](#).

If any one of these policy types explicitly denies access for an operation, then the request is denied. The permissions granted to an entity by multiple permissions types are more complex. For more details about how AWS evaluates policies, see [Policy evaluation logic \(p. 666\)](#).

Identity-based policies with boundaries – Identity-based policies are inline or managed policies that are attached to a user, group of users, or role. Identity-based policies grant permission to the entity, and permissions boundaries limit those permissions. The effective permissions are the intersection of both policy types. An explicit deny in either of these policies overrides the allow.

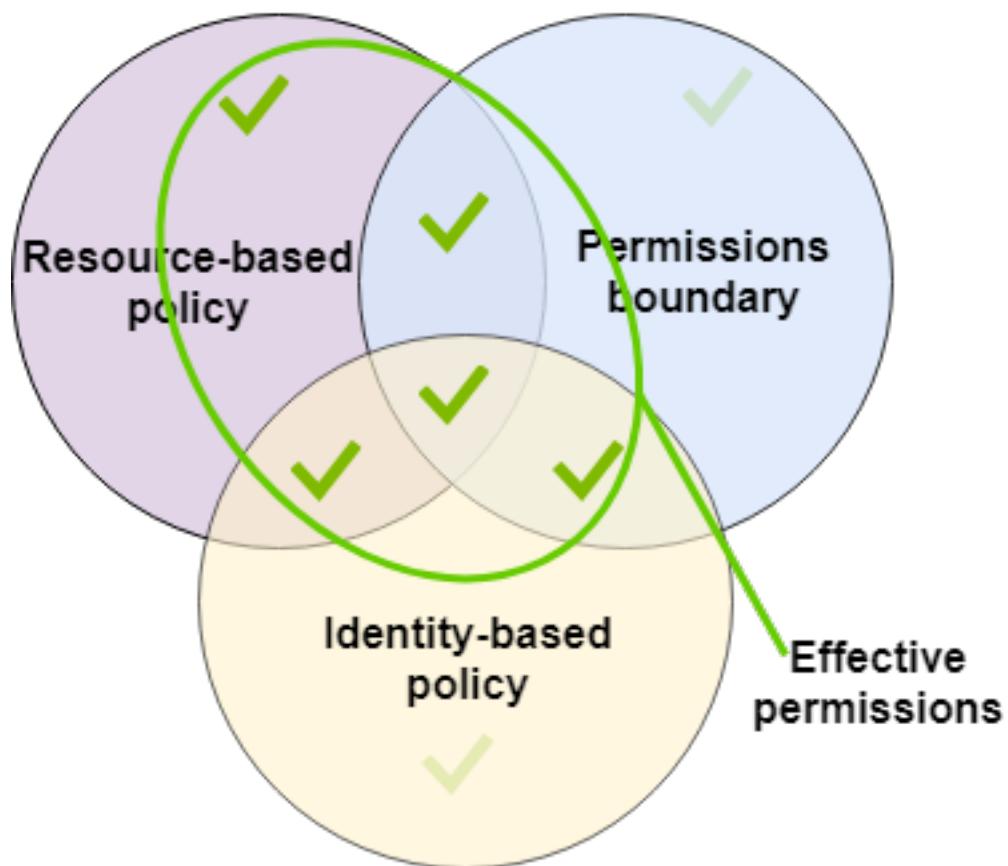


Resource-based policies – Resource-based policies control how the specified principal can access the resource to which the policy is attached.

Resource-based policies for IAM users

Within an account, an implicit deny in a permissions boundary *does not* limit the permissions granted to an IAM user by a resource-based policy. Permissions boundaries reduce permissions that

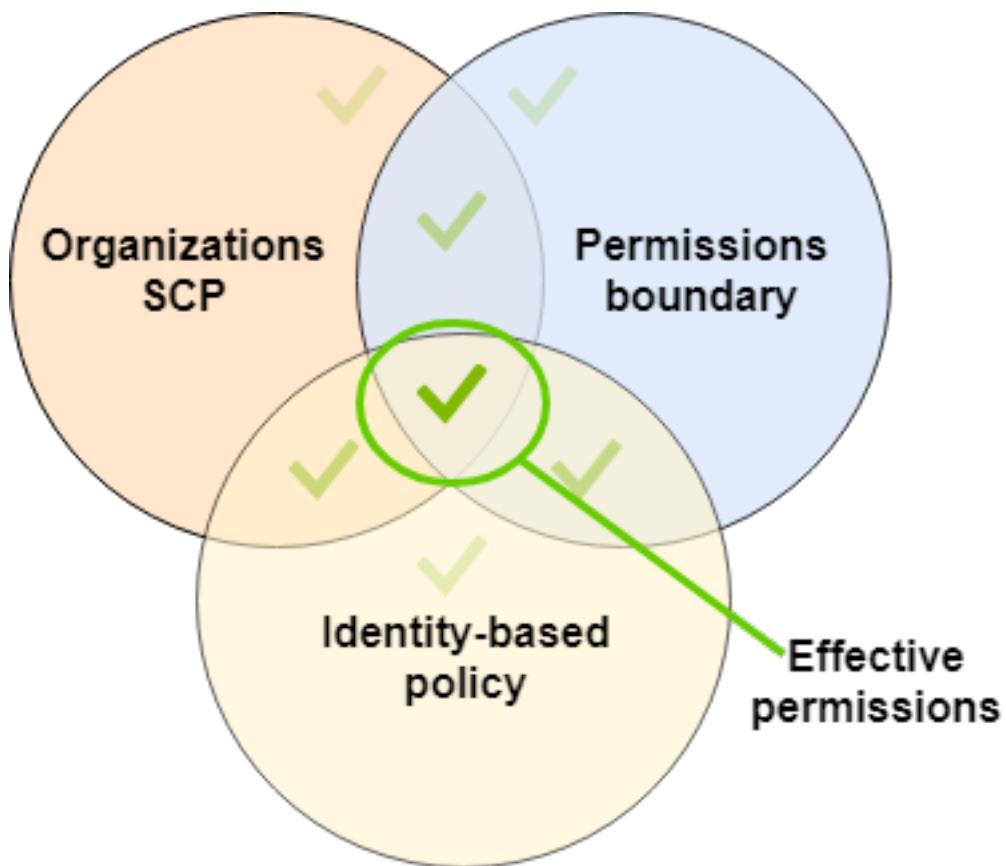
are granted to a user by identity-based policies. Resource-based policies can provide additional permissions to the user.



Resource-based policies for IAM roles and federated users

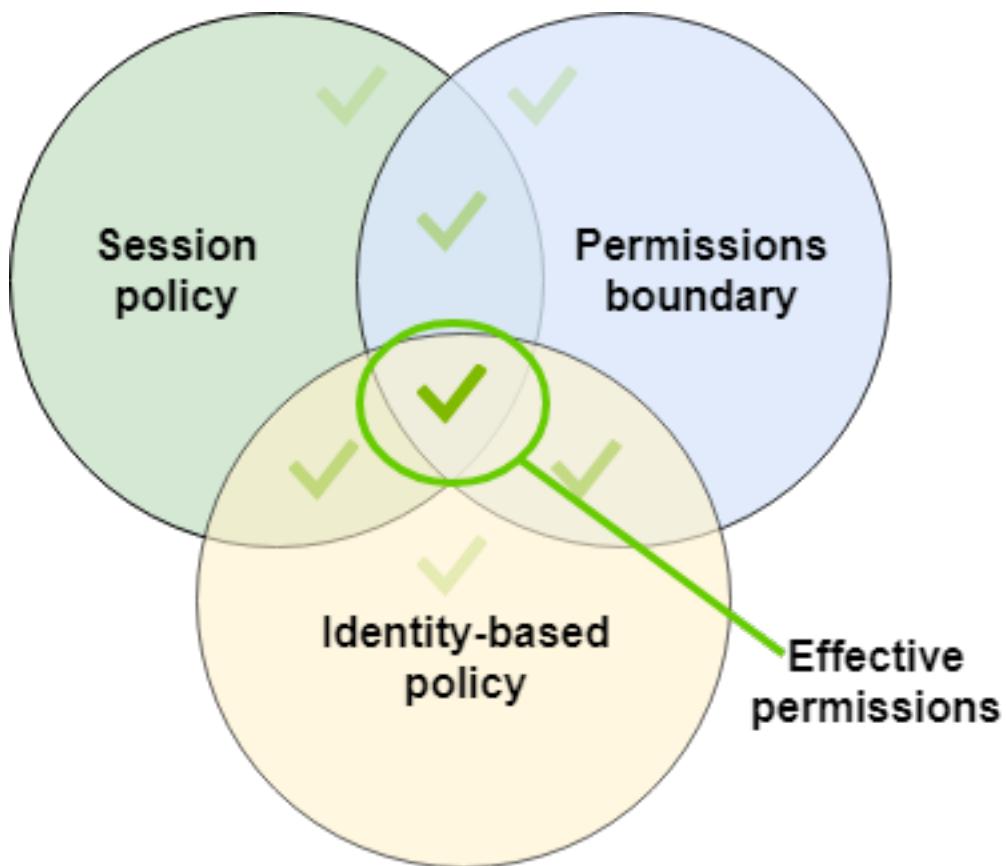
Within an account, an implicit deny in a permissions boundary *does* limit the permissions granted to the ARN of the underlying IAM role or IAM user by the resource-based policy. However, if the resource-based policy grants permissions directly to the session principal (the assumed-role ARN or federated user ARN), an implicit deny in the permissions boundary *does not* limit those permissions. For more information about effective permissions for assumed-role sessions or federated user sessions, see [Session policies](#).

Organizations SCPs – SCPs are applied to an entire AWS account. They limit permissions for every request made by a principal within the account. An IAM entity (user or role) can make a request that is affected by an SCP, a permissions boundary, and an identity-based policy. In this case, the request is allowed only if all three policy types allow it. The effective permissions are the intersection of all three policy types. An explicit deny in any of these policies overrides the allow.



You can learn [whether your account is a member of an organization](#) in AWS Organizations. Organization members might be affected by an SCP. To view this data using the AWS CLI command or AWS API operation, you must have permissions for the `organizations:DescribeOrganization` action for your Organizations entity. You must have additional permissions to perform the operation in the Organizations console. To learn whether an SCP is denying access to a specific request, or to change your effective permissions, contact your AWS Organizations administrator.

Session policies – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The permissions for a session come from the IAM entity (user or role) used to create the session and from the session policy. The entity's identity-based policy permissions are limited by the session policy and the permissions boundary. The effective permissions for this set of policy types are the intersection of all three policy types. An explicit deny in any of these policies overrides the allow. For more information about session policies, see [Session Policies](#).



Delegating responsibility to others using permissions boundaries

You can use permissions boundaries to delegate permissions management tasks, such as user creation, to IAM users in your account. This permits others to perform tasks on your behalf within a specific boundary of permissions.

For example, assume that María is the administrator of the X-Company AWS account. She wants to delegate user creation duties to Zhang. However, she must ensure that Zhang creates users that adhere to the following company rules:

- Users cannot use IAM to create or manage users, groups, roles, or policies.
- Users are denied access to the Amazon S3 logs bucket and cannot access the i-1234567890abcdef0 Amazon EC2 instance.
- Users cannot remove their own boundary policies.

To enforce these rules, María completes the following tasks, for which details are included below:

1. María creates the `xCompanyBoundaries` managed policy to use as a permissions boundary for all new users in the account.
2. María creates the `DelegatedUserBoundary` managed policy and assigns it as the permissions boundary for Zhang. María makes a note of her admin IAM user's ARN and uses it in the policy to prevent Zhang from accessing it.
3. María creates the `DelegatedUserPermissions` managed policy and attaches it as a permissions policy for Zhang.

4. María tells Zhang about his new responsibilities and limitations.

Task 1: María must first create a managed policy to define the boundary for the new users. María will allow Zhang to give users the permissions policies they need, but she wants those users to be restricted. To do this, she creates the following customer managed policy with the name `XCompanyBoundaries`. This policy does the following:

- Allows users full access to several services
- Allows limited self-managing access in the IAM console. This means they can change their password after signing into the console. They can't set their initial password. To allow this, add the `"*LoginProfile"` action to the `AllowManageOwnPasswordAndAccessKeys` statement.
- Denies users access to the Amazon S3 logs bucket or the `i-1234567890abcdef0` Amazon EC2 instance

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ServiceBoundaries",
            "Effect": "Allow",
            "Action": [
                "s3:*",
                "cloudwatch: *",
                "ec2: *",
                "dynamodb: *"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowIAMConsoleForCredentials",
            "Effect": "Allow",
            "Action": [
                "iam>ListUsers",
                "iam:GetAccountPasswordPolicy"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowManageOwnPasswordAndAccessKeys",
            "Effect": "Allow",
            "Action": [
                "iam>*AccessKey*",
                "iam:ChangePassword",
                "iam:GetUser",
                "iam>*ServiceSpecificCredential*",
                "iam>*SigningCertificate*"
            ],
            "Resource": ["arn:aws:iam:::user/${aws:username}"]
        },
        {
            "Sid": "DenyS3Logs",
            "Effect": "Deny",
            "Action": "s3: *",
            "Resource": [
                "arn:aws:s3:::logs",
                "arn:aws:s3:::logs/*"
            ]
        },
        {
            "Sid": "DenyEC2Production",
            "Effect": "Deny",
            "Action": "ec2: *",
            "Resource": [
                "arn:aws:ec2:::instance/i-1234567890abcdef0"
            ]
        }
    ]
}
```

```

        "Action": "ec2:*",
        "Resource": "arn:aws:ec2:***:instance/i-1234567890abcdef0"
    }
]
}

```

Each statement serves a different purpose:

1. The `ServiceBoundaries` statement of this policy allows full access to the specified AWS services. This means that a new user's actions in these services are limited only by the permissions policies that are attached to the user.
2. The `AllowIAMConsoleForCredentials` statement allows access to list all IAM users. This access is necessary to navigate the **Users** page in the AWS Management Console. It also allows viewing the password requirements for the account, which is necessary when changing your own password.
3. The `AllowManageOwnPasswordAndAccessKeys` statement allows the users manage only their own console password and programmatic access keys. This is important if Zhang or another administrator gives a new user a permissions policy with full IAM access. In that case, that user could then change their own or other users' permissions. This statement prevents that from happening.
4. The `DenyS3Logs` statement explicitly denies access to the logs bucket.
5. The `DenyEC2Production` statement explicitly denies access to the `i-1234567890abcdef0` instance.

Task 2: María wants to allow Zhang to create all X-Company users, but only with the `XCompanyBoundaries` permissions boundary. She creates the following customer managed policy named `DelegatedUserBoundary`. This policy defines the maximum permissions that Zhang can have.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CreateOrChangeOnlyWithBoundary",
            "Effect": "Allow",
            "Action": [
                "iam:CreateUser",
                "iam:DeleteUserPolicy",
                "iam:AttachUserPolicy",
                "iam:DetachUserPolicy",
                "iam:PutUserPermissionsBoundary",
                "iam:PutUserPolicy"
            ],
            "Resource": "*",
            "Condition": {"StringEquals":
                {"iam:PermissionsBoundary": "arn:aws:iam::123456789012:policy/
XCompanyBoundaries"}}
        },
        {
            "Sid": "CloudWatchAndOtherIAMTasks",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:*",
                "iam:GetUser",
                "iam>ListUsers",
                "iam>DeleteUser",
                "iam:UpdateUser",
                "iam>CreateAccessKey",
                "iam>CreateLoginProfile",
                "iam:GetAccountPasswordPolicy",
                "iam:GetLoginProfile",
                "iam>ListGroups",
                "iam>ListGroupsForUser",
                "iam:ListUsers"
            ]
        }
    ]
}

```

```

        "iam:CreateGroup",
        "iam:GetGroup",
        "iam:DeleteGroup",
        "iam:UpdateGroup",
        "iam:CreatePolicy",
        "iam:DeletePolicy",
        "iam:DeletePolicyVersion",
        "iam:GetPolicy",
        "iam:GetPolicyVersion",
        "iam:GetUserPolicy",
        "iam:GetRolePolicy",
        "iam>ListPolicies",
        "iam>ListPolicyVersions",
        "iam>ListEntitiesForPolicy",
        "iam>ListUserPolicies",
        "iam>ListAttachedUserPolicies",
        "iam>ListRolePolicies",
        "iam>ListAttachedRolePolicies",
        "iam:SetDefaultPolicyVersion",
        "iam:SimulatePrincipalPolicy",
        "iam:SimulateCustomPolicy"
    ],
    "NotResource": "arn:aws:iam::123456789012:user/Maria"
},
{
    "Sid": "NoBoundaryPolicyEdit",
    "Effect": "Deny",
    "Action": [
        "iam:CreatePolicyVersion",
        "iam:DeletePolicy",
        "iam:DeletePolicyVersion",
        "iam:SetDefaultPolicyVersion"
    ],
    "Resource": [
        "arn:aws:iam::123456789012:policy/XCompanyBoundaries",
        "arn:aws:iam::123456789012:policy/DelegatedUserBoundary"
    ]
},
{
    "Sid": "NoBoundaryUserDelete",
    "Effect": "Deny",
    "Action": "iam>DeleteUserPermissionsBoundary",
    "Resource": "*"
}
]
}

```

Each statement serves a different purpose:

1. The `CreateOrChangeOnlyWithBoundary` statement allows Zhang to create IAM users but only if he uses the `XCompanyBoundaries` policy to set the permissions boundary. This statement also allows him to set the permissions boundary for existing users but only using that same policy. Finally, this statement allows Zhang to manage permissions policies for users with this permissions boundary set.
2. The `CloudWatchAndOtherIAMTasks` statement allows Zhang to complete other user, group, and policy management tasks. He has permissions to reset passwords and create access keys for any IAM user not listed in the condition key. This allows him to help users with sign-in issues.
3. The `NoBoundaryPolicyEdit` statement denies Zhang access to update the `XCompanyBoundaries` policy. He is not allowed to change any policy that is used to set the permissions boundary for himself or other users.
4. The `NoBoundaryUserDelete` statement denies Zhang access to delete the permissions boundary for himself or other users.

María then assigns the `DelegatedUserBoundary` policy [as the permissions boundary \(p. 89\)](#) for the Zhang user.

Task 3: Because the permissions boundary limits the maximum permissions, but does not grant access on its own, María must create a permissions policy for Zhang. She creates the following policy named `DelegatedUserPermissions`. This policy defines the operations that Zhang can perform, within the defined boundary.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "IAM",  
            "Effect": "Allow",  
            "Action": "iam:*",  
            "Resource": "*"  
        },  
        {  
            "Sid": "CloudWatchLimited",  
            "Effect": "Allow",  
            "Action": [  
                "cloudwatch:GetDashboard",  
                "cloudwatch:GetMetricData",  
                "cloudwatch>ListDashboards",  
                "cloudwatch:GetMetricStatistics",  
                "cloudwatch>ListMetrics"  
            ],  
            "Resource": "*"  
        },  
        {  
            "Sid": "S3BucketContents",  
            "Effect": "Allow",  
            "Action": "s3>ListBucket",  
            "Resource": "arn:aws:s3:::ZhangBucket"  
        }  
    ]  
}
```

Each statement serves a different purpose:

1. The `IAM` statement of the policy allows Zhang full access to IAM. However, because his permissions boundary allows only some IAM operations, his effective IAM permissions are limited only by his permissions boundary.
2. The `CloudWatchLimited` statement allows Zhang to perform five actions in CloudWatch. His permissions boundary allows all actions in CloudWatch, so his effective CloudWatch permissions are limited only by his permissions policy.
3. The `S3BucketContents` statement allows Zhang to list the `ZhangBucket` Amazon S3 bucket. However, his permissions boundary does not allow any Amazon S3 action, so he cannot perform any S3 operations, regardless of his permissions policy.

Note

Zhang's policies allow him to create a user that can then access Amazon S3 resources that he can't access. By delegating these administrative actions, María effectively trusts Zhang with access to Amazon S3.

María then attaches the `DelegatedUserPermissions` policy as the permissions policy for the Zhang user.

Task 4: She gives Zhang instructions to create a new user. She tells him that he can create new users with any permissions that they need, but he must assign them the `XCompanyBoundaries` policy as a permissions boundary.

Zhang completes the following tasks:

1. Zhang [creates a user \(p. 77\)](#) with the AWS Management Console. He types the user name Nikhil and enables console access for the user. He clears the checkbox next to **Requires password reset**, because the policies above allow users to change their passwords only after they are signed in to the IAM console.
2. On the **Set permissions** page, Zhang chooses the `IAMFullAccess` and `AmazonS3ReadOnlyAccess` permissions policies that allow Nikhil to do his work.
3. Zhang skips the **Set permissions boundary** section, forgetting María's instructions.
4. Zhang reviews the user details and chooses **Create user**.

The operation fails and access is denied. Zhang's `DelegatedUserBoundary` permissions boundary requires that any user he creates have the `XCompanyBoundaries` policy used as a permissions boundary.

5. Zhang returns to the previous page. In the **Set permissions boundary** section, he chooses the `XCompanyBoundaries` policy.
6. Zhang reviews the user details and chooses **Create user**.

The user is created.

When Nikhil signs in, he has access to IAM and Amazon S3, except those operations that are denied by the permissions boundary. For example, he can change his own password in IAM but can't create another user or edit his policies. Nikhil has read-only access to Amazon S3.

If someone adds a resource-based policy to the `logs` bucket that allows Nikhil to put an object in the bucket, he still cannot access the bucket. The reason is that any actions on the `logs` bucket are explicitly denied by his permissions boundary. An explicit deny in any policy type results in a request being denied. However, if a resource-based policy attached to a Secrets Manager secret allows Nikhil to perform the `secretsmanager:GetSecretValue` action, then Nikhil can retrieve and decrypt the secret. The reason is that Secrets Manager operations are not explicitly denied by his permissions boundary, and implicit denies in permissions boundaries do not limit resource-based policies.

Identity-based policies and resource-based policies

A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. When you create a permissions policy to restrict access to a resource, you can choose an *identity-based policy* or a *resource-based policy*.

Identity-based policies are attached to an IAM user, group, or role. These policies let you specify what that identity can do (its permissions). For example, you can attach the policy to the IAM user named John, stating that he is allowed to perform the Amazon EC2 `RunInstances` action. The policy could further state that John is allowed to get items from an Amazon DynamoDB table named `MyCompany`. You can also allow John to manage his own IAM security credentials. Identity-based policies can be [managed or inline \(p. 359\)](#).

Resource-based policies are attached to a resource. For example, you can attach resource-based policies to Amazon S3 buckets, Amazon SQS queues, and AWS Key Management Service encryption keys. For a list of services that support resource-based policies, see [AWS services that work with IAM \(p. 611\)](#).

With resource-based policies, you can specify who has access to the resource and what actions they can perform on it. To learn whether principals in accounts outside of your zone of trust (trusted organization or account) have access to assume your roles, see [What is IAM Access Analyzer?](#). Resource-based policies are inline only, not managed.

Note

Resource-based policies differ from *resource-level* permissions. You can attach resource-based policies directly to a resource, as described in this topic. Resource-level permissions refer to the ability to use ARNs (p. 601) to specify individual resources in a policy. Resource-based policies are supported only by some AWS services. For a list of which services support resource-based policies and resource-level permissions, see [AWS services that work with IAM \(p. 611\)](#).

To better understand these concepts, view the following figure. The administrator of the 123456789012 account attached *identity-based policies* to the JohnSmith, CarlosSalazar, and MaryMajor users. Some of the actions in these policies can be performed on specific resources. For example, the user JohnSmith can perform some actions on Resource X. This is a *resource-level permission* in an identity-based policy. The administrator also added *resource-based policies* to Resource X, Resource Y, and Resource Z. Resource-based policies allow you to specify who can access that resource. For example, the resource-based policy on Resource X allows the JohnSmith and MaryMajor users list and read access to the resource.

Account ID: 123456789012

Identity-based policies

John Smith

Can List, Read
On Resource X

Carlos Salazar

Can List, Read
On Resource Y,Z

MaryMajor

Can List, Read, Write
On Resource X,Y,Z

ZhangWei

No policy

Resource-based policies

Resource X

JohnSmith: Can List, Read
MaryMajor: Can List, Read

Resource Y

CarlosSalazar: Can List, Write
ZhangWei: Can List, Read

Resource Z

CarlosSalazar: Denied access
ZhangWei: Allowed full access

The 123456789012 account example allows the following users to perform the listed actions:

- **JohnSmith** – John can perform list and read actions on Resource X. He is granted this permission by the identity-based policy on his user and the resource-based policy on Resource X.
- **CarlosSalazar** – Carlos can perform list, read, and write actions on Resource Y, but is denied access to Resource Z. The identity-based policy on Carlos allows him to perform list and read actions on Resource Y. The Resource Y resource-based policy also allows him write permissions. However,

although his identity-based policy allows him access to Resource Z, the Resource Z resource-based policy denies that access. An explicit Deny overrides an Allow and his access to Resource Z is denied. For more information, see [Policy evaluation logic \(p. 666\)](#).

- **MaryMajor** – Mary can perform list, read, and write operations on Resource X, Resource Y, and Resource Z. Her identity-based policy allows her more actions on more resources than the resource-based policies, but none of them deny access.
- **ZhangWei** – Zhang has full access to Resource Z. Zhang has no identity-based policies, but the Resource Z resource-based policy allows him full access to the resource. Zhang can also perform list and read actions on Resource Y.

Identity-based policies and resource-based policies are both permissions policies and are evaluated together. For a request to which only permissions policies apply, AWS first checks all policies for a Deny. If one exists, then the request is denied. Then AWS checks for each Allow. If at least one policy statement allows the action in the request, the request is allowed. It doesn't matter whether the Allow is in the identity-based policy or the resource-based policy.

Important

This logic applies only when the request is made within a single AWS account. For requests made from one account to another, the requester in Account A must have an identity-based policy that allows them to make a request to the resource in Account B. Also, the resource-based policy in Account B must allow the requester in Account A to access the resource. There must be policies in both accounts that allow the operation, otherwise the request fails. For more information about using resource-based policies for cross-account access, see [How IAM roles differ from resource-based policies \(p. 286\)](#).

A user who has specific permissions might request a resource that also has a permissions policy attached to it. In that case, AWS evaluates both sets of permissions when determining whether to grant access to the resource. For information about how policies are evaluated, see [Policy evaluation logic \(p. 666\)](#).

Note

Amazon S3 supports identity-based policies and resource-based policies (referred to as *bucket policies*). In addition, Amazon S3 supports a permission mechanism known as an *access control list (ACL)* that is independent of IAM policies and permissions. You can use IAM policies in combination with Amazon S3 ACLs. For more information, see [Access Control](#) in the *Amazon Simple Storage Service Developer Guide*.

Controlling access to AWS resources using policies

You can use a policy to control access to resources within IAM or all of AWS.

To use a [policy \(p. 351\)](#) to control access in AWS, you must understand how AWS grants access. AWS is composed of collections of *resources*. An IAM user is a resource. An Amazon S3 bucket is a resource. When you use the AWS API, the AWS CLI, or the AWS Management Console to perform an operation (such as creating a user), you send a *request* for that operation. Your request specifies an action, a resource, a *principal entity* (user or role), a *principal account*, and any necessary request information. All of this information provides *context*.

AWS then checks that you (the principal) are authenticated (signed in) and authorized (have permission) to perform the specified action on the specified resource. During authorization, AWS checks all the policies that apply to the context of your request. Most policies are stored in AWS as [JSON documents \(p. 356\)](#) and specify the permissions for principal entities. For more information about policy types and uses, see [Policies and permissions in IAM \(p. 351\)](#).

AWS authorizes the request only if each part of your request is allowed by the policies. To view a diagram of this process, see [Understanding how IAM works \(p. 3\)](#). For details about how AWS determines whether a request is allowed, see [Policy evaluation logic \(p. 666\)](#).

When you create an IAM policy, you can control access to the following:

- **Principals (p. 377)** – Control what the person making the request (the principal (p. 5)) is allowed to do.
- **IAM Identities (p. 378)** – Control which IAM identities (groups, users, and roles) can be accessed and how.
- **IAM Policies (p. 380)** – Control who can create, edit, and delete customer managed policies, and who can attach and detach all managed policies.
- **AWS Resources (p. 383)** – Control who has access to resources using an identity-based policy or a resource-based policy.
- **AWS Accounts (p. 384)** – Control whether a request is allowed only for members of a specific account.

Policies let you specify who has access to AWS resources, and what actions they can perform on those resources. Every IAM user starts with no permissions. In other words, by default, users can do nothing, not even view their own access keys. To give a user permission to do something, you can add the permission to the user (that is, attach a policy to the user). Or you can add the user to a group that has the intended 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 Administrators 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.

For information about how to delegate basic permissions to your users, groups, and roles, see [Permissions required to access IAM resources \(p. 516\)](#). For additional examples of policies that illustrate basic permissions, see [Example policies for administering IAM resources \(p. 520\)](#).

Controlling access for principals

You can use policies to control what the person making the request (the principal) is allowed to do. To do this, you must attach an identity-based policy to that person's identity (user, group of users, or role). You can also use a [permissions boundary \(p. 365\)](#) to set the maximum permissions that an entity (user or role) can have.

For example, assume that you want the user Zhang Wei to have full access to CloudWatch, Amazon DynamoDB, Amazon EC2, and Amazon S3. You can create two different policies so that you can later break them up if you need one set of permissions for a different user. Or you can put both the permissions together in a single policy, and then attach that policy to the IAM user that is named Zhang Wei. You could also attach a policy to a group to which Zhang belongs, or a role that Zhang can assume. As a result, when Zhang views the contents of an S3 bucket, his requests are allowed. If he tries to create a new IAM user, his request is denied because he doesn't have permission.

You can use a permissions boundary on Zhang to make sure that he is never given access to the CompanyConfidential S3 bucket. To do this, determine the *maximum* permissions that you want Zhang to have. In this case, you control what he does using his permissions policies. Here, you only care that he doesn't access the confidential bucket. So you use the following policy to define Zhang's boundary to allow all AWS actions for Amazon S3 and a few other services but deny access to the CompanyConfidential S3 bucket. Because the permissions boundary does not allow any IAM actions, it prevents Zhang from deleting his (or anyone's) boundary.

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

```
"Statement": [
    {
        "Sid": "SomeServices",
        "Effect": "Allow",
        "Action": [
            "cloudwatch:*",
            "dynamodb:*",
            "ec2:*",
            "s3:)"
        ],
        "Resource": "*"
    },
    {
        "Sid": "NoConfidentialBucket",
        "Effect": "Deny",
        "Action": "s3:*",
        "Resource": [
            "arn:aws:s3:::CompanyConfidential/*",
            "arn:aws:s3:::CompanyConfidential"
        ]
    }
]
```

When you assign a policy like this as a permissions boundary for a user, remember that it does not grant any permissions. It sets the maximum permissions that an identity-based policy can grant to an IAM entity. For more information about permissions boundaries, see [Permissions boundaries for IAM entities \(p. 365\)](#).

For detailed information about the procedures mentioned previously, refer to these resources:

- To learn more about creating an IAM policy that you can attach to a principal, see [Creating IAM policies \(p. 439\)](#).
- To learn how to attach an IAM policy to a principal, see [Adding and removing IAM identity permissions \(p. 454\)](#).
- To see an example policy for granting full access to EC2, see [Amazon EC2: Allows full EC2 access within a specific Region, programmatically and in the console \(p. 412\)](#).
- To allow read-only access to an S3 bucket, use the first two statements of the following example policy: [Amazon S3: Allows read and write access to objects in an S3 Bucket, programmatically and in the console \(p. 437\)](#).
- To see an example policy for allowing users to set or rotate their credentials, such as their console password, their programmatic access keys, and their MFA devices, see [AWS: Allows MFA-authenticated IAM users to manage their own credentials on the My Security Credentials page \(p. 393\)](#).

Controlling access to identities

You can use IAM policies to control what your users can do to an identity by creating a policy that you attach to all users through a group. To do this, create a policy that limits what can be done to an identity, or who can access it.

For example, you can create a group named **AllUsers**, and then attach that group to all users. When you create the group, you might give all your users access to rotate their credentials as described in the previous section. You can then create a policy that denies access to change the group unless the user name is included in the condition of the policy. But that part of the policy only denies access to anyone except those users listed. You also have to include permissions to allow all the group management actions for everyone in the group. Finally, you attach this policy to the group so that it is applied to all users. As a result, when a user not specified in the policy tries to make changes to the group, the request is denied.

To create this policy with the visual editor

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 on the left, choose **Policies**.

If this is your first time choosing **Policies**, the **Welcome to Managed Policies** page appears. Choose **Get Started**.

3. Choose **Create policy**.
4. On the **Visual editor** tab, choose **Choose a service** to get started. Then choose **IAM**.
5. Choose **Select actions** and then type **group** in the search box. The visual editor shows all the IAM actions that contain the word **group**. Select all of the check boxes.
6. Choose **Resources** to specify resources for your policy. Based on the actions you chose, you should see **group**, **group-path**, and **user** resource types.
 - **group** – Choose **Add ARN**. For **Resource**, select the check box next to **Any**. For **Group Name With Path**, type the group name **AllUsers**. Then choose **Add**.
 - **group-path** – Select the check box next to **Any**.
 - **user** – Select the check box next to **Any**.

One of the actions that you chose, **ListGroups**, does not support using specific resources. You do not have to choose **All resources** for that action. When you save your policy or view the policy on the **JSON** tab, you can see that IAM automatically creates a new permission block granting this action permission on all resources.

7. To add another permission block, choose **Add additional permissions**.
8. Choose **Choose a service** and then choose **IAM**.
9. Choose **Select actions** and then choose **Switch to deny permissions**. When you do that, the entire block is used to deny permissions.
10. Type **group** in the search box. The visual editor shows you all the IAM actions that contain the word **group**. Select the check boxes next to the following actions:
 - **CreateGroup**
 - **DeleteGroup**
 - **RemoveUserFromGroup**
 - **AttachGroupPolicy**
 - **DeleteGroupPolicy**
 - **DetachGroupPolicy**
 - **PutGroupPolicy**
 - **UpdateGroup**
11. Choose **Resources** to specify the resources for your policy. Based on the actions that you chose, you should see the **group** resource type. Choose **Add ARN**. For **Resource**, select the check box next to **Any**. For **Group Name With Path**, type the group name **AllUsers**. Then choose **Add**.
12. Choose **Specify request conditions (optional)** and then choose **Add condition**. Complete the form with the following values:
 - **Key** – Choose **aws:username**
 - **Qualifier** – Choose **Default**
 - **Operator** – Choose **StringNotEquals**
 - **Value** – Type **srodriguez** and then choose **Add another condition value**. Type **mjackson** and then choose **Add another condition value**. Type **adesai** and then choose **Add**.

This condition ensures that access will be denied to the specified group management actions when the user making the call is not included in the list. Because this explicitly denies permission, it overrides the previous block that allowed those users to call the actions. Users on the list are not denied access, and they are granted permission in the first permission block, so they can fully manage the group.

13. When you are finished, choose **Review policy**.

Note

You can switch between the **Visual editor** and **JSON** tabs any time. However, if you make changes or choose **Review policy** in the **Visual editor** tab, IAM might restructure your policy to optimize it for the visual editor. For more information, see [Policy restructuring \(p. 571\)](#).

14. On the **Review policy** page, for the **Name**, type **LimitAllUserGroupManagement**. For the **Description**, type **Allows all users Read-only access to a specific group, and allows only specific users access to make changes to the group**. Review the policy summary to make sure that you have granted the intended permissions. Then choose **Create policy** to save your new policy.
15. Attach the policy to your group. For more information, see [Adding and removing IAM identity permissions \(p. 454\)](#).

Alternatively, you can create the same policy using this example JSON policy document. To view this JSON policy, see [IAM: Allows specific IAM users to manage a group programmatically and in the console \(p. 423\)](#). For detailed instructions for creating a policy using a JSON document, see the section called “Creating policies on the JSON tab” (p. 440).

Controlling access to policies

You can control how your users can apply AWS managed policies. To do this, attach this policy to all your users. Ideally, you can do this using a group.

For example, you might create a policy that allows users to attach only the [IAMUserChangePassword](#) and [PowerUserAccess](#) AWS managed policies to a new IAM user, group, or role.

For customer managed policies, you can control who can create, update, and delete these policies. You can control who can attach and detach policies to and from principal entities (groups, users, and roles). You can also control which policies a user can attach or detach, and to and from which entities.

For example, you can give permissions to an account administrator to create, update, and delete policies. Then you give permissions to a team leader or other limited administrator to attach and detach these policies to and from principal entities that the limited administrator manages.

For more information, refer to these resources:

- To learn more about creating an IAM policy that you can attach to a principal, see [Creating IAM policies \(p. 439\)](#).
- To learn how to attach an IAM policy to a principal, see [Adding and removing IAM identity permissions \(p. 454\)](#).
- To see an example policy for limiting the use of managed policies, see [IAM: Limits managed policies that can be applied to an IAM user, group, or role \(p. 428\)](#).

Controlling permissions for creating, updating, and deleting customer managed policies

You can use [IAM policies \(p. 351\)](#) to control who is allowed to create, update, and delete customer managed policies in your AWS account. The following list contains API operations that pertain directly to creating, updating, and deleting policies or policy versions:

- [CreatePolicy](#)
- [CreatePolicyVersion](#)
- [DeletePolicy](#)
- [DeletePolicyVersion](#)
- [SetDefaultPolicyVersion](#)

The API operations in the preceding list correspond to actions that you can allow or deny—that is, permissions that you can grant—using an IAM policy.

Consider the following example policy. It allows a user to create, update (that is, create a new policy version), delete, and set a default version for all customer managed policies in the AWS account. The example policy also allows the user to list policies and get policies. To learn how to create a policy using this example JSON policy document, see [the section called “Creating policies on the JSON tab” \(p. 440\)](#).

Example Example policy that allows creating, updating, deleting, listing, getting, and setting the default version for all policies

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "iam:CreatePolicy",  
                "iam:CreatePolicyVersion",  
                "iam>DeletePolicy",  
                "iam>DeletePolicyVersion",  
                "iam:GetPolicy",  
                "iam:GetPolicyVersion",  
                "iam>ListPolicies",  
                "iam>ListPolicyVersions",  
                "iam:SetDefaultPolicyVersion"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

You can create policies that limit the use of these API operations to affect only the managed policies that you specify. For example, you might want to allow a user to set the default version and delete policy versions, but only for specific customer managed policies. You do this by specifying the policy ARN in the `Resource` element of the policy that grants these permissions.

The following example shows a policy that allows a user to delete policy versions and set the default version. But these actions are only allowed for the customer managed policies that include the path `/TEAM-A/`. The customer managed policy ARN is specified in the `Resource` element of the policy. (In this example the ARN includes a path and a wildcard and thus matches all customer managed policies that include the path `/TEAM-A/`). To learn how to create a policy using this example JSON policy document, see [the section called “Creating policies on the JSON tab” \(p. 440\)](#).

For more information about using paths in the names of customer managed policies, see [Friendly names and paths \(p. 600\)](#).

Example Example policy that allows deleting policy versions and setting the default version for only specific policies

```
{  
    "Version": "2012-10-17",  
    "Statement": {  
        "Effect": "Allow",  
        "Action": [  
            "iam:DeletePolicyVersion",  
            "iam:SetDefaultPolicyVersion"  
        ],  
        "Resource": "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:policy/TEAM-A/*"  
    }  
}
```

Controlling permissions for attaching and detaching managed policies

You can also use IAM policies to allow users to work with only specific managed policies. In effect, you can control which permissions a user is allowed to grant to other principal entities.

The following list shows API operations that pertain directly to attaching and detaching managed policies to and from principal entities:

- [AttachGroupPolicy](#)
- [AttachRolePolicy](#)
- [AttachUserPolicy](#)
- [DetachGroupPolicy](#)
- [DetachRolePolicy](#)
- [DetachUserPolicy](#)

You can create policies that limit the use of these API operations to affect only the specific managed policies and/or principal entities that you specify. For example, you might want to allow a user to attach managed policies, but only the managed policies that you specify. Or, you might want to allow a user to attach managed policies, but only to the principal entities that you specify.

The following example policy allows a user to attach managed policies to only the groups and roles that include the path /TEAM-A/. The group and role ARNs are specified in the Resource element of the policy. (In this example the ARNs include a path and a wildcard character and thus match all groups and roles that include the path /TEAM-A/). To learn how to create a policy using this example JSON policy document, see [the section called “Creating policies on the JSON tab” \(p. 440\)](#).

Example Example policy that allows attaching managed policies to only specific groups or roles

```
{  
    "Version": "2012-10-17",  
    "Statement": {  
        "Effect": "Allow",  
        "Action": [  
            "iam:AttachGroupPolicy",  
            "iam:AttachRolePolicy"  
        ],  
        "Resource": [  
            "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:group/TEAM-A/*",  
            "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:role/TEAM-A/*"  
        ]  
    }  
}
```

You can further limit the actions in the preceding example to affect only specific policies. That is, you can control which permissions a user is allowed to attach to other principal entities—by adding a condition to the policy.

In the following example, the condition ensures that the `AttachGroupPolicy` and `AttachRolePolicy` permissions are allowed only when the policy being attached matches one of the specified policies. The condition uses the `iam:PolicyARN` condition key (p. 641) to determine which policy or policies are allowed to be attached. The following example policy expands on the previous example. It allows a user to attach only the managed policies that include the path `/TEAM-A/` to only the groups and roles that include the path `/TEAM-A/`. To learn how to create a policy using this example JSON policy document, see [the section called “Creating policies on the JSON tab” \(p. 440\)](#).

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "iam:AttachGroupPolicy",
            "iam:AttachRolePolicy"
        ],
        "Resource": [
            "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:group/TEAM-A/*",
            "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:role/TEAM-A/*"
        ],
        "Condition": {"ArnLike":
            {"iam:PolicyARN": "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:policy/TEAM-A/*"}
        }
    }
}
```

This policy uses the `ArnLike` condition operator because the ARN includes a wildcard character. For a specific ARN, use the `ArnEquals` condition operator. For more information about `ArnLike` and `ArnEquals`, see [Amazon Resource Name \(ARN\) condition operators \(p. 649\)](#) in the *Condition Types* section of the *Policy Element Reference*.

For example, you can limit the use of actions to involve only the managed policies that you specify. You do this by specifying the policy ARN in the `Condition` element of the policy that grants these permissions. For example, to specify the ARN of a customer managed policy:

```
"Condition": {"ArnEquals":
    {"iam:PolicyARN": "arn:aws:iam::123456789012:policy/POLICY-NAME"}
}
```

You can also specify the ARN of an AWS managed policy in a policy's `Condition` element. The ARN of an AWS managed policy uses the special alias `aws` in the policy ARN instead of an account ID, as in this example:

```
"Condition": {"ArnEquals":
    {"iam:PolicyARN": "arn:aws:iam::aws:policy/AmazonEC2FullAccess"}
}
```

Controlling access to resources

You can control access to resources using an identity-based policy or a resource-based policy. In an identity-based policy, you attach the policy to an identity and specify what resources that identity can access. In a resource-based policy, you attach a policy to the resource that you want to control. In the policy, you specify which principals can access that resource. For more information about both types of policies, see [Identity-based policies and resource-based policies \(p. 374\)](#).

For more information, refer to these resources:

- To learn more about creating an IAM policy that you can attach to a principal, see [Creating IAM policies \(p. 439\)](#).
- To learn how to attach an IAM policy to a principal, see [Adding and removing IAM identity permissions \(p. 454\)](#).
- Amazon S3 supports using resource-based policies on their buckets. For more information, see [Bucket Policy Examples](#).

Resource Creators Do Not Automatically Have Permissions

If you sign in using the AWS account root user credentials, you have 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. This means that just because you create a resource, such as an IAM role, you do not automatically have permission to edit or delete that role. Additionally, your permission can be revoked at any time by the account owner or by another user who has been granted access to manage your permissions.

Controlling access to principals in a specific account

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. A role is an entity that includes permissions but 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 [Providing access to an IAM user in another AWS account that you own \(p. 171\)](#).

Note

Some services support resource-based policies as described in [Identity-based policies and resource-based policies \(p. 374\)](#) (such as Amazon S3, Amazon SNS, and Amazon SQS). For those services, an alternative to using roles is to attach a policy to the resource (bucket, topic, or queue) that you want to share. The resource-based policy can specify the AWS account that has permissions to access the resource.

Controlling access to and for IAM users and roles using IAM resource tags

Use the information in the following section to control who can access your IAM users and roles and what resources your users and roles can access. For more general information and examples for controlling access to other AWS resources, see [Controlling access to AWS resources using resource tags \(p. 387\)](#).

Tags can be attached to the IAM *resource*, passed in the *request*, or attached to the *principal* that is making the request. An IAM user or role can be both a resource and principal. For example, you can write a policy that allows a user to list the groups for a user. This operation is allowed only if the user making the request (the principal) has the same `project=blue` tag as the user they're trying to view. In this example, the user can view the group membership for any user, including themselves, as long as they are working on the same project.

To control access based on tags, you provide tag information in the [condition element \(p. 641\)](#) of a policy. When you create an IAM policy, you can use IAM tags and the associated tag condition key to control access to any of the following:

- [Resource \(p. 385\)](#) – Control access to user or role resources based on their tags. To do this, use the `iam:ResourceTag/key-name` condition key to specify which tag key-value pair must be attached

to the resource. A similar service-specific key, such as `ec2:ResourceTag`, is used for other AWS resources. For more information, see [Controlling access to AWS resources \(p. 388\)](#).

- **Request (p. 386)** – Control what tags can be passed in an IAM request. To do this, use the `aws:RequestTag/key-name` condition key to specify what tags can be added, changed, or removed from an IAM user or role. This key is used the same way for IAM resources and other AWS resources. For more information, see [Controlling access during AWS requests \(p. 388\)](#).
- **Principal (p. 386)** – Control what the person making the request (the principal) is allowed to do based on the tags that are attached to that person's IAM user or role. To do this, use the `aws:PrincipalTag/key-name` condition key to specify what tags must be attached to the IAM user or role before the request is allowed.
- **Any part of the authorization process (p. 386)** – Use the `aws:TagKeys` condition key to control whether specific tag keys can be used on a resource, in a request, or by a principal. In this case, the key value does not matter. This key behaves similarly for IAM resources and other AWS resources. However, when you tag a user in IAM, this also controls whether the principal can make the request to any service. For more information, see [Controlling access based on tag keys \(p. 389\)](#).

You can create an IAM policy using the visual editor, using JSON, or by importing an existing managed policy. For details, see [Creating IAM policies \(p. 439\)](#).

Controlling access to IAM resources

You can use tags in your IAM policies to control access to IAM user and role resources. However, because IAM does not support tags for groups, you cannot use tags to control access to groups.

This example shows how you might create a policy that allows deleting users with the `status=terminated` tag. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:DeleteUser",
            "Resource": "*",
            "Condition": {"StringLike": {"iam:ResourceTag/status": "terminated"}}
        }
    ]
}
```

This example shows how you might create a policy that allows editing tags for all users with the `jobFunction = employee` tag. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam>ListUserTags",
                "iam:TagUser",
                "iam:UntagUser"
            ],
            "Resource": "*",
            "Condition": {"StringLike": {"iam:ResourceTag/jobFunction": "employee"}}
        }
    ]
}
```

Controlling access during IAM requests

You can use tags in your IAM policies to control what tags can be passed in the IAM request. You can specify which tag key-value pairs can be added, changed, or removed from an IAM user or role.

This example shows how you might create a policy that allows tagging users only with a `department = HR` or `department = CS` tag. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#).

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {"Effect": "Allow",  
         "Action": "iam:TagUser",  
         "Resource": "*",  
         "Condition": {"StringLike": {"aws:RequestTag/department": [  
             "HR",  
             "CS"  
         ]}}  
    ]  
}
```

Controlling access for IAM principals

IAM tags enable you to control what the principal is allowed to do based on the tags attached to that person's identity.

This example shows how you might create a policy that allows a principal to start or stop an Amazon EC2 instance. This operation is allowed only when the instance's resource tag and the principal's tag have the same value for the tag key `cost-center`. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#).

```
{  
    "Version": "2012-10-17",  
    "Statement": {  
        "Effect": "Allow",  
        "Action": [  
            "ec2:startInstances",  
            "ec2:stopInstances"  
        ],  
        "Resource": "*",  
        "Condition": {"StringEquals":  
            {"ec2:ResourceTag/cost-center": "${aws:PrincipalTag/cost-center}"}}  
    }  
}
```

Controlling access based on tag keys

You can use tags in your IAM policies to control whether specific tag keys can be used on a resource, in a request, or by a principal.

This example shows how you might create a policy that allows removing only the tag with the `temporary` key from users. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#).

```
{
```

```
"Version": "2012-10-17",
"Statement": [
    {
        "Effect": "Allow",
        "Action": "iam:UntagUser",
        "Resource": "*",
        "Condition": {"ForAllValues:StringEquals": {"aws:TagKeys": ["temporary"]}}
    }
]
```

Controlling access to AWS resources using resource tags

You can use tags to control access to your AWS resources that support tagging. You can also tag IAM users and roles to control what they can access. To learn how to tag IAM users and roles, see [Tagging IAM users and roles \(p. 289\)](#). To view a tutorial for creating and testing a policy that allows IAM roles with principal tags to access resources with matching tags, see [IAM Tutorial: Define permissions to access AWS resources based on tags \(p. 45\)](#). Use the information in the following section to control access to other AWS services without tagging IAM users or roles.

Before you use tags to control access to your AWS resources, you must understand how AWS grants access. AWS is composed of collections of *resources*. An Amazon EC2 instance is a resource. An Amazon S3 bucket is a resource. You can use the AWS API, the AWS CLI, or the AWS Management Console to perform an operation, such as creating a bucket in Amazon S3. When you do, you send a *request* for that operation. Your request specifies an action, a resource, a *principal entity* (user or role), a *principal account*, and any necessary request information. All of this information provides *context*.

AWS then checks that you (the principal entity) are authenticated (signed in) and authorized (have permission) to perform the specified action on the specified resource. During authorization, AWS checks all the policies that apply to the context of your request. Most policies are stored in AWS as [JSON documents \(p. 356\)](#) and specify the permissions for principal entities. For more information about policy types and uses, see [Policies and permissions in IAM \(p. 351\)](#).

AWS authorizes the request only if each part of your request is allowed by the policies. To view a diagram and learn more about the IAM infrastructure, see [Understanding how IAM works \(p. 3\)](#). For details about how IAM determines whether a request is allowed, see [Policy evaluation logic \(p. 666\)](#).

Tags can complicate this process because tags can be attached to the *resource* or passed in the *request* to services that support tagging. To control access based on tags, you provide tag information in the [condition element \(p. 641\)](#) of a policy. To learn whether an AWS service supports controlling access using tags, see [AWS services that work with IAM \(p. 611\)](#) and look for the services that have **Yes** in the **Authorization based on tags** column. Choose the name of the service to view the authorization and access control documentation for that service.

You can then create an IAM policy that allows or denies access to a resource based on that resource's tag. In that policy, you can use tag condition keys to control access to any of the following:

- **Resource (p. 388)** – Control access to AWS service resources based on the tags on those resources. To do this, use the **ResourceTag/*key-name*** condition key to determine whether to allow access to the resource based on the tags that are attached to the resource.
- **Request (p. 388)** – Control what tags can be passed in a request. To do this, use the **aws:RequestTag/*key-name*** condition key to specify what tag key-value pairs can be passed in a request to tag or untag an AWS resource.
- **Any part of the authorization process (p. 389)** – Use the **aws:TagKeys** condition key to control whether specific tag keys can be used on a resource or in a request.

You can create an IAM policy visually, using JSON, or by importing an existing managed policy. For details, see [Creating IAM policies \(p. 439\)](#).

Controlling access to AWS resources

You can use conditions in your IAM policies to control access to AWS resources based on the tags on that resource. You can do this using the global `aws:ResourceTag/tag-key` condition key, or a service-specific key such as `iam:ResourceTag/tag-key`. Some services, such as IAM, support only the service-specific version of this key and not the global version.

Note

Do not use the `ResourceTag` condition key in a policy with the `iam:PassRole` action. You cannot use the tag on an IAM role to control access to who can pass that role. For more information about permissions required to pass a role to a service, see [Granting a user permissions to pass a role to an AWS service \(p. 251\)](#).

This example shows how you might create a policy that allows starting or stopping Amazon EC2 instances. These operations are allowed only if the instance tag `Owner` has the value of that user's user name. This policy also grants the necessary permissions to complete this action on the console.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:StartInstances",
                "ec2:StopInstances"
            ],
            "Resource": "arn:aws:ec2:*:*:instance/*",
            "Condition": {
                "StringEquals": {"ec2:ResourceTag/Owner": "${aws:username}"}
            }
        },
        {
            "Effect": "Allow",
            "Action": "ec2:DescribeInstances",
            "Resource": "*"
        }
    ]
}
```

You can attach this policy to the IAM users in your account. If a user named `richard-roe` attempts to start an Amazon EC2 instance, the instance must be tagged `Owner=richard-roe` or `owner=richard-roe`. Otherwise he will be denied access. The tag key `Owner` matches both `Owner` and `owner` because condition key names are not case-sensitive. For more information, see [IAM JSON policy elements: Condition \(p. 641\)](#).

Controlling access during AWS requests

You can use conditions in your IAM policies to control what tag key-value pairs can be passed in a request that tags an AWS resource.

This example shows how you might create a policy that allows using the Amazon EC2 `CreateTags` action to attach tags to an instance. You can attach tags only if the tag contains the `environment` key and the `preprod` or `production` values. If you want, you can use the `ForAllValues` modifier with the `aws:TagKeys` condition key to indicate that only the key `environment` is allowed in the request. This stops users from including other keys, such as accidentally using `Environment` instead of `environment`.

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

```

"Statement": {
    "Effect": "Allow",
    "Action": "ec2:CreateTags",
    "Resource": "arn:aws:ec2:*:instance/*",
    "Condition": {
        "StringEquals": {
            "aws:RequestTag/environmentpreprod",
                "production"
            ]
        },
        "ForAllValues:StringEquals": {"aws:TagKeys": "environment"}
    }
}

```

Controlling access based on tag keys

You can use a condition in your IAM policies to control whether specific tag keys can be used on a resource or in a request.

As a best practice, when you use policies to control access using tags, you should use the [aws:TagKeys condition key \(p. 705\)](#). AWS services that support tags might allow you to create multiple tag key names that differ only by case, such as tagging an Amazon EC2 instance with `stack=production` and `Stack=test`. Key names are not case sensitive in policy conditions. This means that if you specify `"ec2:ResourceTag/TagName": "Value1"` in the condition element of your policy, then the condition matches a resource tag key named either `TagName` or `tagname`, but not both. To prevent duplicate tags with a key that varies only by case, use the `aws:TagKeys` condition to define the tag keys that your users can apply.

This example shows how you might create a policy that allows creating and tagging a Secrets Manager secret, but only with the tag keys `environment` or `cost-center`.

```

{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "secretsmanager>CreateSecret",
            "secretsmanager:TagResource"
        ],
        "Resource": "*",
        "Condition": {
            "ForAllValues:StringEquals": {
                "aws:TagKeys": [
                    "environment",
                    "cost-center"
                ]
            }
        }
    }
}

```

Example IAM identity-based policies

A [policy \(p. 351\)](#) is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when an IAM principal (user or role) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents that are attached to an IAM identity (user, group of users, or role). Identity-based policies include AWS managed policies, customer managed policies, and inline policies.

To learn how to create an IAM policy using these example JSON policy documents, see [the section called "Creating policies on the JSON tab" \(p. 440\)](#).

By default all requests are denied, so you must provide access to the services, actions, and resources that you intend for the identity to access. If you also want to allow access to complete the specified actions in the IAM console, you need to provide additional permissions.

The following library of policies can help you define permissions for your IAM identities. After you find the policy that you need, choose **view this policy** to view the JSON for the policy. You can use the JSON policy document as a template for your own policies.

Note

If you would like to submit a policy to be included in this reference guide, use the **Feedback** button at the bottom of this page.

Example policies: AWS

- Allows access during a specific range of dates. ([View this policy \(p. 392\)](#).)
- Allows enabling and disabling AWS Regions. ([View this policy \(p. 393\)](#).)
- Allows MFA-authenticated users to manage their own credentials on the **My Security Credentials** page. ([View this policy \(p. 393\)](#).)
- Allows specific access when using MFA during a specific range of dates. ([View this policy \(p. 396\)](#).)
- Allows users to manage their own credentials on the **My Security Credentials** page. ([View this policy \(p. 397\)](#).)
- Allows users to manage their own MFA device on the **My Security Credentials** page. ([View this policy \(p. 399\)](#).)
- Allows users to manage their own password on the **My Security Credentials** page. ([View this policy \(p. 401\)](#).)
- Allows users to manage their own password, access keys, and SSH public keys on the **My Security Credentials** page. ([View this policy \(p. 402\)](#).)
- Denies access to AWS based on the requested Region. ([View this policy \(p. 403\)](#).)
- Denies access to AWS based on the source IP address. ([View this policy \(p. 404\)](#).)

Example policies: AWS Data Pipeline

- Denies access to pipelines that a user did not create ([View this policy \(p. 405\)](#).)

Example policies: Amazon DynamoDB

- Allows access to a specific Amazon DynamoDB table ([View this policy \(p. 405\)](#).)
- Allows access to specific Amazon DynamoDB attributes ([View this policy \(p. 406\)](#).)
- Allows item-level access to Amazon DynamoDB based on an Amazon Cognito ID ([View this policy \(p. 407\)](#).)

Example policies: Amazon EC2

- Allows an Amazon EC2 instance to attach or detach volumes ([View this policy \(p. 408\)](#).)
- Allows attaching or detaching Amazon EBS volumes to Amazon EC2 instances based on tags ([View this policy \(p. 408\)](#).)
- Allows launching Amazon EC2 instances in a specific subnet, programmatically and in the console ([View this policy \(p. 409\)](#).)

- Allows managing Amazon EC2 security groups associated with a specific VPC, programmatically and in the console ([View this policy \(p. 409\)](#).)
- Allows starting or stopping Amazon EC2 instances a user has tagged, programmatically and in the console ([View this policy \(p. 410\)](#).)
- Allows starting or stopping Amazon EC2 instances based on resource and principal tags, programmatically and in the console ([View this policy \(p. 411\)](#).)
- Allows starting or stopping Amazon EC2 instances when the resource and principal tags match ([View this policy \(p. 411\)](#).)
- Allows full Amazon EC2 access within a specific Region, programmatically and in the console. ([View this policy \(p. 412\)](#).)
- Allows starting or stopping a specific Amazon EC2 instance and modifying a specific security group, programmatically and in the console ([View this policy \(p. 412\)](#).)
- Denies access to specific Amazon EC2 operations without MFA ([View this policy \(p. 413\)](#).)
- Limits terminating Amazon EC2 instances to a specific IP address range ([View this policy \(p. 414\)](#).)

Example policies: AWS Identity and Access Management (IAM)

- Allows access to the policy simulator API ([View this policy \(p. 414\)](#).)
- Allows access to the policy simulator console ([View this policy \(p. 415\)](#).)
- Allows assuming any roles that have a specific tag, programmatically and in the console ([View this policy \(p. 416\)](#).)
- Allows and denies access to multiple services, programmatically and in the console ([View this policy \(p. 416\)](#).)
- Allows adding a specific tag to an IAM user with a different specific tag, programmatically and in the console ([View this policy \(p. 418\)](#).)
- Allows adding a specific tag to any IAM user or role, programmatically and in the console ([View this policy \(p. 418\)](#).)
- Allows creating a new user only with specific tags ([View this policy \(p. 419\)](#).)
- Allows generating and retrieving IAM credential reports ([View this policy \(p. 420\)](#).)
- Allows managing a group's membership, programmatically and in the console ([View this policy \(p. 421\)](#).)
- Allows managing a specific tag ([View this policy \(p. 421\)](#).)
- Allows passing an IAM role to a specific service ([View this policy \(p. 422\)](#).)
- Allows read-only access to the IAM console without reporting ([View this policy \(p. 422\)](#).)
- Allows read-only access to the IAM console ([View this policy \(p. 423\)](#).)
- Allows specific users to manage a group, programmatically and in the console ([View this policy \(p. 423\)](#).)
- Allows setting the account password requirements, programmatically and in the console ([View this policy \(p. 424\)](#).)
- Allows using the policy simulator API for users with a specific path ([View this policy \(p. 425\)](#).)
- Allows using the policy simulator console for users with a specific path ([View this policy \(p. 425\)](#).)
- Allows IAM users to self-manage an MFA device. ([View this policy \(p. 426\)](#).)
- Allows IAM users to rotate their own credentials, programmatically and in the console. ([View this policy \(p. 427\)](#).)
- Allows viewing service last accessed information for an AWS Organizations policy in the IAM console. ([View this policy \(p. 428\)](#).)
- Limits managed policies that can be applied to an IAM user, group, or role ([View this policy \(p. 428\)](#).)

Example policies: AWS Lambda

- Allows an AWS Lambda function to access an Amazon DynamoDB table ([View this policy \(p. 429\)](#).)

Example policies: Amazon RDS

- Allows full Amazon RDS database access within a specific Region. ([View this policy \(p. 430\)](#).)
- Allows restoring Amazon RDS databases, programmatically and in the console ([View this policy \(p. 430\)](#).)
- Allows tag owners full access to Amazon RDS resources that they have tagged ([View this policy \(p. 431\)](#).)

Example policies: Amazon S3

- Allows an Amazon Cognito user to access objects in their own Amazon S3 bucket ([View this policy \(p. 432\)](#).)
- Allows federated users to access their own home directory in Amazon S3, programmatically and in the console ([View this policy \(p. 433\)](#).)
- Allows full S3 access, but explicitly denies access to the Production bucket if the administrator has not signed in using MFA within the last thirty minutes ([View this policy \(p. 434\)](#).)
- Allows IAM users to access their own home directory in Amazon S3, programmatically and in the console ([View this policy \(p. 435\)](#).)
- Allows a user to manage a single Amazon S3 bucket and denies every other AWS action and resource ([View this policy \(p. 436\)](#).)
- Allows Read and Write access to a specific Amazon S3 bucket ([View this policy \(p. 437\)](#).)
- Allows Read and Write access to a specific Amazon S3 bucket, programmatically and in the console ([View this policy \(p. 437\)](#).)

AWS: Allows access based on date and time

This example shows how you might create a policy that allows access to actions based on date and time. This policy restricts access to actions that occur between April 1, 2020 and June 30, 2020 (UTC), inclusive. This policy grants the permissions necessary to complete this action from the AWS API or AWS CLI only. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#).

To learn about using multiple conditions within the Condition block of an IAM policy, see [Multiple values in a condition \(p. 643\)](#).

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "service-prefix:action-name",  
            "Resource": "*",  
            "Condition": {  
                "DateGreaterThan": {"aws:CurrentTime": "2020-04-01T00:00:00Z"},  
                "DateLessThan": {"aws:CurrentTime": "2020-06-30T23:59:59Z"}  
            }  
        }  
    ]  
}
```

}

AWS: Allows enabling and disabling AWS regions

This example shows how you might create a policy that allows an administrator to enable and disable the Asia Pacific (Hong Kong) Region (ap-east-1). This policy also grants the necessary permissions to complete this action on the console. This setting appears in the **Account settings** page in the AWS Management Console. This page includes sensitive account-level information that should be viewed and managed only by account administrators. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#).

Important

You cannot enable or disable regions that are enabled by default. You can only include regions that are *disabled* by default. For more information, see [Managing AWS Regions](#) in the AWS General Reference.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "EnableDisableHongKong",  
            "Effect": "Allow",  
            "Action": [  
                "account:EnableRegion",  
                "account:DisableRegion"  
            ],  
            "Resource": "*",  
            "Condition": {  
                "StringEquals": {"account:TargetRegion": "ap-east-1"}  
            }  
        },  
        {  
            "Sid": "ViewConsole",  
            "Effect": "Allow",  
            "Action": [  
                "aws-portal:ViewAccount",  
                "account>ListRegions"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

AWS: Allows MFA-authenticated IAM users to manage their own credentials on the My Security Credentials page

This example shows how you might create a policy that allows IAM users that are authenticated using [multi-factor authentication \(MFA\) \(p. 111\)](#) to manage their own credentials on the **My Security Credentials** page. This AWS Management Console page displays account information such as the account ID and canonical user ID. Users can also view and edit their own passwords, access keys, MFA devices, X.509 certificates, and SSH keys and Git credentials. This example policy includes the permissions required to view and edit all of the information on the page. It also requires the user to set up and authenticate using MFA before performing any other operations in AWS. To allow users to manage their own credentials without using MFA, see [AWS: Allows IAM users to manage their own credentials on the My Security Credentials page \(p. 397\)](#).

To learn how users can access the **My Security Credentials** page, see [How IAM users change their own password \(console\) \(p. 101\)](#).

Note

This example policy does not allow users to reset a password while signing in for the first time. AWS recommends that you do not grant permissions to new users until after they sign in. For more information, see [How do I securely create IAM users? \(p. 570\)](#). This also prevents users with an expired password from resetting their password before signing in. You can allow this by adding `iam:ChangePassword` and `iam:GetAccountPasswordPolicy` to the statement `DenyAllExceptListedIfNoMFA`. However, IAM does not recommend this. Allowing users to change their password without MFA can be a security risk.

What does this policy do?

- The `AllowViewAccountInfo` statement allows the user to view account-level information. These permissions must be in their own statement because they do not support or do not need to specify a resource ARN. Instead the permissions specify "Resource" : "*". This statement includes the following actions that allow the user to view specific information:
 - `GetAccountSummary` – View the account ID and the account [canonical user ID](#).
 - `GetAccountPasswordPolicy` – View the account password requirements while changing their own IAM user password.
 - `ListVirtualMFADevices` – View details about a virtual MFA device that is enabled for the user.
- The `AllowManageOwnPasswords` statement allows the user to change their own password. This statement also includes the `GetUser` action, which is required to view most of the information on the **My Security Credentials** page.
- The `AllowManageOwnAccessKeys` statement allows the user to create, update, and delete their own access keys.
- The `AllowManageOwnSigningCertificates` statement allows the user to upload, update, and delete their own signing certificates.
- The `AllowManageOwnSSHPublicKeys` statement allows the user to upload, update, and delete their own SSH public keys for CodeCommit.
- The `AllowManageOwnGitCredentials` statement allows the user to create, update, and delete their own Git credentials for CodeCommit.
- The `AllowManageOwnVirtualMFADevice` statement allows the user to create and delete their own virtual MFA device. The resource ARN in this statement allows access to only an MFA device that has the same name as the currently signed-in user. Users can't create or delete any virtual MFA device other than their own.
- The `AllowManageOwnUserMFA` statement allows the user to view or manage the virtual, U2F, or hardware MFA device for their own user. The resource ARN in this statement allows access to only the user's own IAM user. Users can't view or manage the MFA device for other users.
- The `DenyAllExceptListedIfNoMFA` statement denies access to every action in all AWS services, except a few listed actions, but **only if** the user is not signed in with MFA. The statement uses a combination of "Deny" and "NotAction" to explicitly deny access to every action that is not listed. The items listed are not denied or allowed by this statement. However, the actions are allowed by other statements in the policy. For more information about the logic for this statement, see [NotAction with Deny \(p. 638\)](#). If the user is signed in with MFA, then the Condition test fails and this statement does not deny any actions. In this case, other policies or statements for the user determine the user's permissions.

This statement ensures that when the user is not signed in with MFA that they can perform only the listed actions. In addition, they can perform the listed actions only if another statement or policy allows access to those actions. This does not allow a user to create a password at sign-in, because `iam:ChangePassword` action should not be allowed without MFA authorization.

The `...IfExists` version of the `Bool` operator ensures that if the `aws:MultiFactorAuthPresent` key is missing, the condition returns true. This means that a user accessing an API with long-term credentials, such as an access key, is denied access to the non-IAM API operations.

This policy does not allow users to view the **Users** page in the IAM console or use that page to access their own user information. To allow this, add the `iam>ListUsers` action to the `AllowViewAccountInfo` statement and the `DenyAllExceptListedIfNoMFA` statement. It also does not allow users to change their password on their own user page. To allow this, add the `iam>CreateLoginProfile`, `iam>DeleteLoginProfile`, `iamGetLoginProfile`, and `iamUpdateLoginProfile` actions to the `AllowManageOwnPasswords` statement. To also allow a user to change their password from their own user page without signing in using MFA, add the `iamCreateLoginProfile` action to the `DenyAllExceptListedIfNoMFA` statement.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowViewAccountInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetAccountPasswordPolicy",
                "iam:GetAccountSummary",
                "iam>ListVirtualMFADevices"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowManageOwnPasswords",
            "Effect": "Allow",
            "Action": [
                "iam:ChangePassword",
                "iam GetUser"
            ],
            "Resource": "arn:aws:iam::*:user/${aws:username}"
        },
        {
            "Sid": "AllowManageOwnAccessKeys",
            "Effect": "Allow",
            "Action": [
                "iam>CreateAccessKey",
                "iam>DeleteAccessKey",
                "iam>ListAccessKeys",
                "iam:UpdateAccessKey"
            ],
            "Resource": "arn:aws:iam::*:user/${aws:username}"
        },
        {
            "Sid": "AllowManageOwnSigningCertificates",
            "Effect": "Allow",
            "Action": [
                "iam>DeleteSigningCertificate",
                "iam>ListSigningCertificates",
                "iam:UpdateSigningCertificate",
                "iam:UploadSigningCertificate"
            ],
            "Resource": "arn:aws:iam::*:user/${aws:username}"
        },
        {
            "Sid": "AllowManageOwnSSHPublicKeys",
            "Effect": "Allow",
            "Action": [
                "iam>DeleteSSHPublicKey",
                "iam:GetSSHPublicKey",
                "iam>ListSSHPublicKeys",
                "iam:UpdateSSHPublicKey",
                "iam:UploadSSHPublicKey"
            ],
            "Resource": "arn:aws:iam::*:user/${aws:username}"
        }
    ]
}
```

```

},
{
    "Sid": "AllowManageOwnGitCredentials",
    "Effect": "Allow",
    "Action": [
        "iam>CreateServiceSpecificCredential",
        "iam>DeleteServiceSpecificCredential",
        "iam>ListServiceSpecificCredentials",
        "iam>ResetServiceSpecificCredential",
        "iam>UpdateServiceSpecificCredential"
    ],
    "Resource": "arn:aws:iam::*:user/${aws:username}"
},
{
    "Sid": "AllowManageOwnVirtualMFADevice",
    "Effect": "Allow",
    "Action": [
        "iam>CreateVirtualMFADevice",
        "iam>DeleteVirtualMFADevice"
    ],
    "Resource": "arn:aws:iam:::mfa/${aws:username}"
},
{
    "Sid": "AllowManageOwnUserMFA",
    "Effect": "Allow",
    "Action": [
        "iam>DeactivateMFADevice",
        "iam>EnableMFADevice",
        "iam>ListMFADevices",
        "iam>ResyncMFADevice"
    ],
    "Resource": "arn:aws:iam::*:user/${aws:username}"
},
{
    "Sid": "DenyAllExceptListedIfNoMFA",
    "Effect": "Deny",
    "NotAction": [
        "iam>CreateVirtualMFADevice",
        "iam>EnableMFADevice",
        "iam GetUser",
        "iam>ListMFADevices",
        "iam>ListVirtualMFADevices",
        "iam>ResyncMFADevice",
        "sts:GetSessionToken"
    ],
    "Resource": "*",
    "Condition": {
        "BoolIfExists": {
            "aws:MultiFactorAuthPresent": "false"
        }
    }
}
]
}

```

AWS: Allows specific access using MFA within specific dates

This example shows how you might create a policy that uses multiple conditions, which are evaluated using a logical AND. It allows full access to the service named SERVICE-NAME-1, and access to the ACTION-NAME-A and ACTION-NAME-B actions in the service named SERVICE-NAME-2. These actions are allowed only when the user is authenticated using [multifactor authentication \(MFA\)](#). Access is restricted to actions that occur between July 1, 2017 and December 31, 2017 (UTC), inclusive. This policy grants the permissions necessary to complete this action from the AWS API or AWS CLI only. To use this

policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#).

To learn about using multiple conditions within the Condition block of an IAM policy, see [Multiple values in a condition \(p. 643\)](#).

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "service-prefix-1:*",  
                "service-prefix-2:action-name-a",  
                "service-prefix-2:action-name-b"  
            ],  
            "Resource": "*",  
            "Condition": {  
                "Bool": {"aws:MultiFactorAuthPresent": true},  
                "DateGreaterThanOrEqualTo": {"aws:CurrentTime": "2017-07-01T00:00:00Z"},  
                "DateLessThanOrEqualTo": {"aws:CurrentTime": "2017-12-31T23:59:59Z"}  
            }  
        }  
    ]  
}
```

AWS: Allows IAM users to manage their own credentials on the My Security Credentials page

This example shows how you might create a policy that allows IAM users to manage all of their own credentials on the **My Security Credentials** page. This AWS Management Console page displays account information such as the account ID and canonical user ID. Users can also view and edit their own passwords, access keys, X.509 certificates, SSH keys, and Git credentials. This example policy includes the permissions required to view and edit all information on the page *except* the user's MFA device. To allow users to manage their own credentials with MFA, see [AWS: Allows MFA-authenticated IAM users to manage their own credentials on the My Security Credentials page \(p. 393\)](#).

To learn how users can access the **My Security Credentials** page, see [How IAM users change their own password \(console\) \(p. 101\)](#).

What does this policy do?

- The `AllowViewAccountInfo` statement allows the user to view account-level information. These permissions must be in their own statement because they do not support or do not need to specify a resource ARN. Instead the permissions specify `"Resource" : "*"`. This statement includes the following actions that allow the user to view specific information:
 - `GetAccountPasswordPolicy` – View the account password requirements while changing their own IAM user password.
 - `GetAccountSummary` – View the account ID and the account [canonical user ID](#).
- The `AllowManageOwnPasswords` statement allows the user to change their own password. This statement also includes the `GetUser` action, which is required to view most of the information on the **My Security Credentials** page.
- The `AllowManageOwnAccessKeys` statement allows the user to create, update, and delete their own access keys.
- The `AllowManageOwnSigningCertificates` statement allows the user to upload, update, and delete their own signing certificates.
- The `AllowManageOwnSSHPublicKeys` statement allows the user to upload, update, and delete their own SSH public keys for CodeCommit.

- The `AllowManageOwnGitCredentials` statement enables the user to create, update, and delete their own Git credentials for CodeCommit.

This policy does not allow users to view or manage their own MFA devices. They also cannot view the **Users** page in the IAM console or use that page to access their own user information. To allow this, add the `iam>ListUsers` action to the `AllowViewAccountInfo` statement. It also does not allow users to change their password on their own user page. To allow this, add the `iam>CreateLoginProfile`, `iam>DeleteLoginProfile`, `iamGetLoginProfile`, and `iamUpdateLoginProfile` actions to the `AllowManageOwnPasswords` statement.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowViewAccountInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetAccountPasswordPolicy",
                "iam:GetAccountSummary"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowManageOwnPasswords",
            "Effect": "Allow",
            "Action": [
                "iam:ChangePassword",
                "iam:GetUser"
            ],
            "Resource": "arn:aws:iam::*:user/${aws:username}"
        },
        {
            "Sid": "AllowManageOwnAccessKeys",
            "Effect": "Allow",
            "Action": [
                "iam>CreateAccessKey",
                "iam>DeleteAccessKey",
                "iam>ListAccessKeys",
                "iam:UpdateAccessKey"
            ],
            "Resource": "arn:aws:iam::*:user/${aws:username}"
        },
        {
            "Sid": "AllowManageOwnSigningCertificates",
            "Effect": "Allow",
            "Action": [
                "iam>DeleteSigningCertificate",
                "iam>ListSigningCertificates",
                "iam:UpdateSigningCertificate",
                "iam:UploadSigningCertificate"
            ],
            "Resource": "arn:aws:iam::*:user/${aws:username}"
        },
        {
            "Sid": "AllowManageOwnSSHPublicKeys",
            "Effect": "Allow",
            "Action": [
                "iam>DeleteSSHPublicKey",
                "iam:GetSSHPublicKey",
                "iam>ListSSHPublicKeys",
                "iam:UpdateSSHPublicKey",
                "iam:UploadSSHPublicKey"
            ],
        }
    ]
}
```

```
        "Resource": "arn:aws:iam::*:user/${aws:username}"  
    },  
    {  
        "Sid": "AllowManageOwnGitCredentials",  
        "Effect": "Allow",  
        "Action": [  
            "iam:CreateServiceSpecificCredential",  
            "iam>DeleteServiceSpecificCredential",  
            "iam>ListServiceSpecificCredentials",  
            "iam>ResetServiceSpecificCredential",  
            "iam>UpdateServiceSpecificCredential"  
        ],  
        "Resource": "arn:aws:iam::*:user/${aws:username}"  
    }  
}  
]
```

AWS: Allows MFA-authenticated IAM users to manage their own MFA device on the My Security Credentials page

This example shows how you might create a policy that allows IAM users that are authenticated through [multi-factor authentication \(MFA\) \(p. 111\)](#) to manage their own MFA device on the **My Security Credentials** page. This AWS Management Console page displays account and user information, but the user can only view and edit their own MFA device. To allow users to manage all of their own credentials with MFA, see [AWS: Allows MFA-authenticated IAM users to manage their own credentials on the My Security Credentials page \(p. 393\)](#).

Note

If an IAM user with this policy is not MFA-authenticated, this policy denies access to all AWS actions except those necessary to authenticate using MFA. To use the AWS CLI and AWS API, IAM users must first retrieve their MFA token using the AWS STS [GetSessionToken](#) operation and then use that token to authenticate the desired operation. Other policies, such as resource-based policies or other identity-based policies can allow actions in other services. This policy will deny that access if the IAM user is not MFA-authenticated.

To learn how users can access the **My Security Credentials** page, see [How IAM users change their own password \(console\) \(p. 101\)](#).

What does this policy do?

- The `AllowViewAccountInfo` statement allows the user to view details about a virtual MFA device that is enabled for the user. This permission must be in its own statement because it does not support specifying a resource ARN. Instead you must specify `"Resource" : "*"`.
- The `AllowManageOwnVirtualMFADevice` statement allows the user to create and delete their own virtual MFA device. The resource ARN in this statement allows access to only an MFA device that has the same name as the currently signed-in user. Users can't create or delete any virtual MFA device other than their own.
- The `AllowManageOwnUserMFA` statement allows the user to view or manage their own virtual, U2F, or hardware MFA device. The resource ARN in this statement allows access to only the user's own IAM user. Users can't view or manage the MFA device for other users.
- The `DenyAllExceptListedIfNoMFA` statement denies access to every action in all AWS services, except a few listed actions, but **only if** the user is not signed in with MFA. The statement uses a combination of `"Deny"` and `"NotAction"` to explicitly deny access to every action that is not listed. The items listed are not denied or allowed by this statement. However, the actions are allowed by other statements in the policy. For more information about the logic for this statement, see [NotAction with Deny \(p. 638\)](#). If the user is signed in with MFA, then the Condition test fails and this statement does not deny any actions. In this case, other policies or statements for the user determine the user's permissions.

This statement ensures that when the user is not signed in with MFA, they can perform only the listed actions. In addition, they can perform the listed actions only if another statement or policy allows access to those actions.

The `...IfExists` version of the `Bool` operator ensures that if the `aws:MultiFactorAuthPresent` key is missing, the condition returns true. This means that a user accessing an API operation with long-term credentials, such as an access key, is denied access to the non-IAM API operations.

This policy does not allow users to view the **Users** page in the IAM console or use that page to access their own user information. To allow this, add the `iam>ListUsers` action to the `AllowViewAccountInfo` statement and the `DenyAllExceptListedIfNoMFA` statement.

Warning

Do not add permission to delete an MFA device without MFA authentication. Users with this policy might attempt to assign themselves an MFA device and receive an error that they are not authorized to perform `iam>DeleteVirtualMFADevice`. If this happens, **do not** add that permission to the `DenyAllExceptListedIfNoMFA` statement. Users that are not authenticated using MFA should never be allowed to delete their MFA device. Users might see this error if they previously began assigning a virtual MFA device to their user and cancelled the process. To resolve this issue, you or another administrator must delete the user's existing MFA device using the AWS CLI or AWS API. For more information, see [I am not authorized to perform: iam>DeleteVirtualMFADevice \(p. 569\)](#).

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowViewAccountInfo",  
            "Effect": "Allow",  
            "Action": "iam>ListVirtualMFADevices",  
            "Resource": "*"  
        },  
        {  
            "Sid": "AllowManageOwnVirtualMFADevice",  
            "Effect": "Allow",  
            "Action": [  
                "iam>CreateVirtualMFADevice",  
                "iam>DeleteVirtualMFADevice"  
            ],  
            "Resource": "arn:aws:iam::*:mfa/${aws:username}"  
        },  
        {  
            "Sid": "AllowManageOwnUserMFA",  
            "Effect": "Allow",  
            "Action": [  
                "iam>DeactivateMFADevice",  
                "iam>EnableMFADevice",  
                "iam GetUser",  
                "iam>ListMFADevices",  
                "iam>ResyncMFADevice"  
            ],  
            "Resource": "arn:aws:iam::*:user/${aws:username}"  
        },  
        {  
            "Sid": "DenyAllExceptListedIfNoMFA",  
            "Effect": "Deny",  
            "NotAction": [  
                "iam>CreateVirtualMFADevice",  
                "iam>EnableMFADevice",  
                "iam GetUser",  
                "iam>ListMFADevices",  
                "iam>ListVirtualMFADevices"  
            ]  
        }  
    ]  
}
```

```

        "iam>ListVirtualMFADevices",
        "iam>ResyncMFADevice",
        "sts:GetSessionToken"
    ],
    "Resource": "*",
    "Condition": {
        "BoolIfExists": { "aws:MultiFactorAuthPresent": "false" }
    }
}
]
}

```

AWS: Allows IAM users to change their own console password on the My Security Credentials page

This example shows how you might create a policy that allows IAM users to change their own AWS Management Console password on the **My Security Credentials** page. This AWS Management Console page displays account and user information, but the user can only access their own password. To allow users to manage all of their own credentials with MFA, see [AWS: Allows MFA-authenticated IAM users to manage their own credentials on the My Security Credentials page \(p. 393\)](#). To allow users to manage their own credentials without using MFA, see [AWS: Allows IAM users to manage their own credentials on the My Security Credentials page \(p. 397\)](#).

To learn how users can access the **My Security Credentials** page, see [How IAM users change their own password \(console\) \(p. 101\)](#).

What does this policy do?

- The `ViewAccountPasswordRequirements` statement allows the user to view the account password requirements while changing their own IAM user password.
- The `ChangeOwnPassword` statement allows the user to change their own password. This statement also includes the `GetUser` action, which is required to view most of the information on the **My Security Credentials** page.

This policy does not allow users to view the **Users** page in the IAM console or use that page to access their own user information. To allow this, add the `iam>ListUsers` action to the `ViewAccountPasswordRequirements` statement.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewAccountPasswordRequirements",
            "Effect": "Allow",
            "Action": "iam:GetAccountPasswordPolicy",
            "Resource": "*"
        },
        {
            "Sid": "ChangeOwnPassword",
            "Effect": "Allow",
            "Action": [
                "iam:GetUser",
                "iam:ChangePassword"
            ],
            "Resource": "arn:aws:iam::*:user/${aws:username}"
        }
    ]
}

```

AWS: Allows IAM users to manage their own password, access keys, and SSH public keys on the My Security Credentials page

This example shows how you might create a policy that allows IAM users to manage their own password, access keys, and X.509 certificates on the **My Security Credentials** page. This AWS Management Console page displays account information such as the account ID and canonical user ID. Users can also view and edit their own passwords, access keys, MFA devices, X.509 certificates, SSH keys, and Git credentials. This example policy includes the permissions that are required to view and edit only their password, access keys, and X.509 certificate. To allow users to manage all of their own credentials with MFA, see [AWS: Allows MFA-authenticated IAM users to manage their own credentials on the My Security Credentials page \(p. 393\)](#). To allow users to manage their own credentials without using MFA, see [AWS: Allows IAM users to manage their own credentials on the My Security Credentials page \(p. 397\)](#).

To learn how users can access the **My Security Credentials** page, see [How IAM users change their own password \(console\) \(p. 101\)](#).

What does this policy do?

- The `AllowViewAccountInfo` statement allows the user to view account-level information. These permissions must be in their own statement because they do not support or do not need to specify a resource ARN. Instead the permissions specify "Resource" : "*". This statement includes the following actions that allow the user to view specific information:
 - `GetAccountPasswordPolicy` – View the account password requirements while changing their own IAM user password.
 - `GetAccountSummary` – View the account ID and the account [canonical user ID](#).
- The `AllowManageOwnPasswords` statement allows the user to change their own password. This statement also includes the `GetUser` action, which is required to view most of the information on the **My Security Credentials** page.
- The `AllowManageOwnAccessKeys` statement allows the user to create, update, and delete their own access keys.
- The `AllowManageOwnSSHPublicKeys` statement allows the user to upload, update, and delete their own SSH public keys for CodeCommit.

This policy does not allow users to view or manage their own MFA devices. They also cannot view the **Users** page in the IAM console or use that page to access their own user information. To allow this, add the `iam>ListUsers` action to the `AllowViewAccountInfo` statement. It also does not allow users to change their password on their own user page. To allow this, add the `iam>CreateLoginProfile`, `iam>DeleteLoginProfile`, `iam>GetLoginProfile`, and `iam>UpdateLoginProfile` actions to the `AllowManageOwnPasswords` statement.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowViewAccountInfo",  
            "Effect": "Allow",  
            "Action": [  
                "iam:GetAccountPasswordPolicy",  
                "iam:GetAccountSummary"  
            ],  
            "Resource": "*"  
        },  
        {  
            "Sid": "AllowManageOwnPasswords",  
            "Effect": "Allow",  
            "Action": [  
                "iam:CreateLoginProfile",  
                "iam>DeleteLoginProfile",  
                "iam:GetLoginProfile",  
                "iam:UpdateLoginProfile"  
            ]  
        }  
    ]  
}
```

```

        "iam:ChangePassword",
        "iam:GetUser"
    ],
    "Resource": "arn:aws:iam::*:user/${aws:username}"
},
{
    "Sid": "AllowManageOwnAccessKeys",
    "Effect": "Allow",
    "Action": [
        "iam:CreateAccessKey",
        "iam:DeleteAccessKey",
        "iam>ListAccessKeys",
        "iam:UpdateAccessKey"
    ],
    "Resource": "arn:aws:iam::*:user/${aws:username}"
},
{
    "Sid": "AllowManageOwnSSHPublicKeys",
    "Effect": "Allow",
    "Action": [
        "iam>DeleteSSHPublicKey",
        "iam:GetSSHPublicKey",
        "iam>ListSSHPublicKeys",
        "iam:UpdateSSHPublicKey",
        "iam:UploadSSHPublicKey"
    ],
    "Resource": "arn:aws:iam::*:user/${aws:username}"
}
]
}

```

AWS: Denies access to AWS based on the requested Region

This example shows how you might create a policy that denies access to any actions outside the Regions specified using `aws:RequestedRegion`, except for actions in the services specified using `NotAction`. This policy also grants the necessary permissions to complete this action on the console. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#).

This policy uses the `NotAction` element with the `Deny` effect, which explicitly denies access to all of the actions *not* listed in the statement. Actions in the CloudFront, IAM, Route 53, and AWS Support services should not be denied because these are popular AWS global services with a single endpoint that is physically located in the `us-east-1` Region. Because all requests to these services are made to the `us-east-1` Region, the requests would be denied without the `NotAction` element. Edit this element to include actions for other AWS global services that you use, such as budgets, globalaccelerator, importexport, organizations, or waf. To learn about all of the services that have a single global endpoint, see [AWS Regions and Endpoints](#) in the *AWS General Reference*. For more information about using the `NotAction` element with the `Deny` effect, see [IAM JSON policy elements: NotAction \(p. 638\)](#).

Important

This policy does not allow any actions. Use this policy in combination with other policies that allow specific actions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DenyAllOutsideRequestedRegions",
            "Effect": "Deny",
            "NotAction": [
                "cloudfront:*",
                "iam:*,"
            ]
        }
    ]
}
```

```

        "route53:*",
        "support:/*"
    ],
    "Resource": "*",
    "Condition": {
        "StringNotEquals": {
            "aws:RequestedRegion": [
                "eu-central-1",
                "eu-west-1",
                "eu-west-2",
                "eu-west-3"
            ]
        }
    }
}
]
}
}

```

AWS: Denies access to AWS based on the source IP

This example shows how you might create a policy that denies access to all AWS actions in the account when the request comes from *principals* outside the specified IP range. The policy is useful when the IP addresses for your company are within the specified ranges. The policy does not deny requests made by AWS services using the principal's credentials. This policy also grants the necessary permissions to complete this action on the console. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#).

Be careful using negative conditions in the same policy statement as "Effect": "Deny". When you do, the actions specified in the policy statement are explicitly denied in all conditions *except* for the ones specified.

Additionally, this policy includes [multiple condition keys \(p. 652\)](#) that result in a logical AND. In this policy, all AWS actions are denied when the source IP address is not in the specified range AND when an AWS service does not make the call.

Important

This policy does not allow any actions. Use this policy in combination with other policies that allow specific actions.

When other policies allow actions, principals can make requests from within the IP address range. An AWS service can also make requests using the principal's credentials. When a principal makes a request from outside the IP range, the request is denied. It is also denied if the service uses a [service role](#) or [service-linked role](#) to make a call on the principal's behalf.

For more information about using the `aws:SourceIp` and `aws:ViaAWSService` condition keys, see [AWS global condition context keys \(p. 692\)](#).

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

```
        },
        "Bool": {"aws:ViaAWSService": "false"}
    }
}
```

AWS Data Pipeline: Denies access to DataPipeline pipelines that a user did not create

This example shows how you might create a policy that denies access to pipelines that a user did not create. If the value of the `PipelineCreator` field matches the IAM user name, then the specified actions are not denied. This policy grants the permissions necessary to complete this action from the AWS API or AWS CLI only.

Important

This policy does not allow any actions. Use this policy in combination with other policies that allow specific actions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ExplicitDenyIfNotTheOwner",
            "Effect": "Deny",
            "Action": [
                "datapipeline:ActivatePipeline",
                "datapipeline:AddTags",
                "datapipeline:DeactivatePipeline",
                "datapipeline>DeletePipeline",
                "datapipeline:DescribeObjects",
                "datapipeline:EvaluateExpression",
                "datapipeline:GetPipelineDefinition",
                "datapipeline:PollForTask",
                "datapipeline:PutPipelineDefinition",
                "datapipeline:QueryObjects",
                "datapipeline:RemoveTags",
                "datapipeline:ReportTaskProgress",
                "datapipeline:ReportTaskRunnerHeartbeat",
                "datapipeline:SetStatus",
                "datapipeline:SetTaskStatus",
                "datapipeline:ValidatePipelineDefinition"
            ],
            "Resource": ["*"],
            "Condition": {
                "StringNotEquals": {"datapipeline:PipelineCreator": "${aws:userid}"}
            }
        }
    ]
}
```

Amazon DynamoDB: Allows access to a specific table

This example shows how you might create a policy that allows full access to the `MyTable` DynamoDB table. This policy grants the permissions necessary to complete this action from the AWS API or AWS CLI only. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#).

Important

This policy allows all actions that can be performed on a DynamoDB table. To review these actions, see [DynamoDB API Permissions: Actions, Resources, and Conditions Reference](#) in

the *Amazon DynamoDB Developer Guide*. You could provide the same permissions by listing each individual action. However, if you use the wildcard (*) in the Action element, such as "dynamodb>List*", then you don't have to update your policy if DynamoDB adds a new List action.

This policy allows actions only on DynamoDB tables that exist with the specified name. To allow your users Read access to everything in DynamoDB, you can also attach the [AmazonDynamoDBReadOnlyAccess](#) AWS managed policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ListAndDescribe",
            "Effect": "Allow",
            "Action": [
                "dynamodb>List*",
                "dynamodb>DescribeReservedCapacity*",
                "dynamodb>DescribeLimits",
                "dynamodb>DescribeTimeToLive"
            ],
            "Resource": "*"
        },
        {
            "Sid": "SpecificTable",
            "Effect": "Allow",
            "Action": [
                "dynamodb>BatchGet*",
                "dynamodb>DescribeStream",
                "dynamodb>DescribeTable",
                "dynamodb>Get*",
                "dynamodb>Query",
                "dynamodb>Scan",
                "dynamodb>BatchWrite*",
                "dynamodb>CreateTable",
                "dynamodb>Delete*",
                "dynamodb>Update*",
                "dynamodb>PutItem"
            ],
            "Resource": "arn:aws:dynamodb:*.*:table/MyTable"
        }
    ]
}
```

Amazon DynamoDB: Allows access to specific attributes

This example shows how you might create a policy that allows access to the specific DynamoDB attributes. This policy grants the permissions necessary to complete this action from the AWS API or AWS CLI only. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#).

The dynamodb>Select requirement prevents the API action from returning any attributes that aren't allowed, such as from an index projection. To learn more about DynamoDB condition keys, see [Specifying Conditions: Using Condition Keys](#) in the *Amazon DynamoDB Developer Guide*. To learn about using multiple conditions or multiple condition keys within the Condition block of an IAM policy, see [Multiple values in a condition \(p. 643\)](#).

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

```

```

    "Effect": "Allow",
    "Action": [
        "dynamodb:GetItem",
        "dynamodb:BatchGetItem",
        "dynamodb:Query",
        "dynamodb:PutItem",
        "dynamodb:UpdateItem",
        "dynamodb:DeleteItem",
        "dynamodb:BatchWriteItem"
    ],
    "Resource": ["arn:aws:dynamodb:*:*:table/table-name"],
    "Condition": {
        "ForAllValues:StringEquals": {
            "dynamodb:Attributes": [
                "column-name-1",
                "column-name-2",
                "column-name-3"
            ]
        },
        "StringEqualsIfExists": {"dynamodb:Select": "SPECIFIC_ATTRIBUTES"}
    }
}
]
}

```

Amazon DynamoDB: Allows item-level access to DynamoDB based on an Amazon Cognito ID

This example shows how you might create a policy that allows item-level access to the `MyTable` DynamoDB table based on an Amazon Cognito ID. This policy grants the permissions necessary to complete this action from the AWS API or AWS CLI only. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#).

To use this policy, you must structure your DynamoDB table so the Amazon Cognito user ID is the partition key. For more information, see [Creating a Table](#) in the *Amazon DynamoDB Developer Guide*.

To learn more about DynamoDB condition keys, see [Specifying Conditions: Using Condition Keys](#) in the *Amazon DynamoDB Developer Guide*.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:DeleteItem",
                "dynamodb:GetItem",
                "dynamodb:PutItem",
                "dynamodb:Query",
                "dynamodb:UpdateItem"
            ],
            "Resource": ["arn:aws:dynamodb:*:*:table/MyTable"],
            "Condition": {
                "ForAllValues:StringEquals": {
                    "dynamodb:LeadingKeys": ["${cognito-identity.amazonaws.com:sub}"]
                }
            }
        }
    ]
}

```

Amazon EC2: Attach or detach volumes to an EC2 instance

This example shows how you might create a policy that allows EBS volume owners to attach or detach volumes to the specified EC2 instance. The instance is specified with an ARN in the Condition element. This policy grants the permissions necessary to complete this action from the AWS API or AWS CLI only. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#).

Amazon EC2 instances can run AWS commands with permissions granted by an [AWS service role for an EC2 instance \(p. 168\)](#) that is attached to the instance profile. You can attach this policy to the role, or add this statement to an existing policy. Only the instance identified by *instance-id* can attach or detach volumes to instances in the account, including its own. Other statement elements that might exist in a larger policy are not impacted by the restriction of this one statement. For more information about creating IAM policies to control access to Amazon EC2 resources, see [Controlling Access to Amazon EC2 Resources](#) in the *Amazon EC2 User Guide for Linux Instances*.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AttachVolume",
                "ec2:DetachVolume"
            ],
            "Resource": [
                "arn:aws:ec2::::volume/*",
                "arn:aws:ec2::::instance/*"
            ],
            "Condition": {
                "ArnEquals": {"ec2:SourceInstanceARN": "arn:aws:ec2::::instance/instance-id"}
            }
        }
    ]
}
```

Amazon EC2: Attach or detach Amazon EBS volumes to EC2 instances based on tags

This example shows how you might create a policy that allows EBS volume owners to attach or detach their EBS volumes defined using the tag `VolumeUser` to EC2 instances that are tagged as development instances (`Department=Dev`). This policy grants the permissions necessary to complete this action from the AWS API or AWS CLI only. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AttachVolume",
                "ec2:DetachVolume"
            ],
            "Resource": "arn:aws:ec2::::instance/*",
            "Condition": {
                "StringEquals": {"ec2:ResourceTag/Department": "Development"}
            }
        }
    ]
}
```

```

        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AttachVolume",
                "ec2:DetachVolume"
            ],
            "Resource": "arn:aws:ec2:*::volume/*",
            "Condition": {
                "StringEquals": {"ec2:ResourceTag/VolumeUser": "${aws:username}"}
            }
        }
    ]
}

```

Amazon EC2: Allows launching EC2 instances in a specific subnet, programmatically and in the console

This example shows how you might create a policy that allows listing information for all EC2 objects and launching EC2 instances in a specific subnet. This policy also grants the necessary permissions to complete this action on the console. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#).

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:Describe*",
                "ec2:GetConsole*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "ec2:RunInstances",
            "Resource": [
                "arn:aws:ec2:::subnet/subnet-id",
                "arn:aws:ec2:::network-interface/*",
                "arn:aws:ec2:::instance/*",
                "arn:aws:ec2:::volume/*",
                "arn:aws:ec2:::image/ami-*",
                "arn:aws:ec2:::key-pair/*",
                "arn:aws:ec2:::security-group/*"
            ]
        }
    ]
}

```

Amazon EC2: Allows managing EC2 security groups associated with a specific VPC, programmatically and in the console

This example shows how you might create a policy that allows managing Amazon EC2 security groups associated with a specific virtual private cloud (VPC). This policy also grants the necessary permissions to complete this action on the console. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AuthorizeSecurityGroupEgress",
                "ec2:AuthorizeSecurityGroupIngress",
                "ec2>DeleteSecurityGroup",
                "ec2:RevokeSecurityGroupEgress",
                "ec2:RevokeSecurityGroupIngress"
            ],
            "Resource": "arn:aws:ec2:*:*:security-group/*",
            "Condition": {
                "ArnEquals": {
                    "ec2:Vpc": "arn:aws:ec2:*:*:vpc/vpc-vpc-id"
                }
            }
        },
        {
            "Action": [
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSecurityGroupReferences",
                "ec2:DescribeStaleSecurityGroups",
                "ec2:DescribeVpcs"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

Amazon EC2: Allows starting or stopping EC2 instances a user has tagged, programmatically and in the console

This example shows how you might create a policy that allows an IAM user to start or stop EC2 instances, but only if the instance tag `Owner` has the value of that user's user name. This policy also grants the necessary permissions to complete this action on the console.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:StartInstances",
                "ec2:StopInstances"
            ],
            "Resource": "arn:aws:ec2:*:*:instance/*",
            "Condition": {
                "StringEquals": {
                    "ec2:ResourceTag/Owner": "${aws:username}"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "ec2:DescribeInstances",
            "Resource": "*"
        }
    ]
}
```

}

EC2: Start or stop instances based on tags

This example shows how you might create a policy that allows starting or stopping instances with the tag key-value pair `Project = DataAnalytics`, but only by principals with the tag key-value pair `Department = Data`. This policy grants the permissions necessary to complete this action from the AWS API or AWS CLI only. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#).

The condition in the policy returns true if both parts of the condition are true. The instance must have the `Project=DataAnalytics` tag. In addition, the IAM principal (user or role) making the request must have the `Department=Data` tag.

Note

As a best practice, attach policies with the `aws:PrincipalTag` condition key to IAM groups, for the case where some users might have the specified tag and some might not.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "StartStopIfTags",  
            "Effect": "Allow",  
            "Action": [  
                "ec2:StartInstances",  
                "ec2:StopInstances",  
                "ec2:DescribeTags"  
            ],  
            "Resource": "arn:aws:ec2:region:account-id:instance/*",  
            "Condition": {  
                "StringEquals": {  
                    "ec2:ResourceTag/ProjectDataAnalytics",  
                    "aws:PrincipalTag/DepartmentData"  
                }  
            }  
        }  
    ]  
}
```

EC2: Start or stop instances based on matching principal and resource tags

This example shows how you might create a policy that allows a principal to start or stop an Amazon EC2 instance when the instance's resource tag and the principal's tag have the same value for the tag key `CostCenter`. This policy grants the permissions necessary to complete this action from the AWS API or AWS CLI only. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#).

Note

As a best practice, attach policies with the `aws:PrincipalTag` condition key to IAM groups, for the case where some users might have the specified tag and some might not.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "ec2:StartInstances",  
                "ec2:StopInstances",  
                "ec2:DescribeTags"  
            ],  
            "Resource": "arn:aws:ec2:region:account-id:instance/*",  
            "Condition": {  
                "StringEquals": {  
                    "ec2:ResourceTag/CostCenterCostCenter",  
                    "aws:PrincipalTag/CostCenterCostCenter"  
                }  
            }  
        }  
    ]  
}
```

```

        "ec2:startInstances",
        "ec2:stopInstances"
    ],
    "Resource": "*",
    "Condition": {"StringEquals":
        {"ec2:ResourceTag/CostCenter": "${aws:PrincipalTag/CostCenter}"}}
}
}

```

Amazon EC2: Allows full EC2 access within a specific Region, programmatically and in the console

This example shows how you might create a policy that allows full EC2 access within a specific Region. This policy also grants the necessary permissions to complete this action on the console. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#). For a list of Region codes, see [Available Regions](#) in the *Amazon EC2 User Guide*.

Alternatively, you can use the global condition key `aws:RequestedRegion`, which is supported by all Amazon EC2 API actions. For more information, see [Example: Restricting access to a specific Region](#) in the *Amazon EC2 User Guide*.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": "ec2:*",
            "Resource": "*",
            "Effect": "Allow",
            "Condition": {
                "StringEquals": {
                    "ec2:Region": "us-east-2"
                }
            }
        }
    ]
}
```

Amazon EC2: Allows starting or stopping an EC2 instance and modifying a security group, programmatically and in the console

This example shows how you might create a policy that allows starting or stopping a specific EC2 instance and modifying a specific security group. This policy also grants the necessary permissions to complete this action on the console. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "ec2:DescribeInstances",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSecurityGroupReferences",
                "ec2:DescribeStaleSecurityGroups"
            ],

```

```

        "Resource": "*",
        "Effect": "Allow"
    },
    {
        "Action": [
            "ec2:AuthorizeSecurityGroupEgress",
            "ec2:AuthorizeSecurityGroupIngress",
            "ec2:RevokeSecurityGroupEgress",
            "ec2:RevokeSecurityGroupIngress",
            "ec2:StartInstances",
            "ec2:StopInstances"
        ],
        "Resource": [
            "arn:aws:ec2:*.*:instance/i-instance-id",
            "arn:aws:ec2:*.*:security-group/sg-security-group-id"
        ],
        "Effect": "Allow"
    }
]
}

```

Amazon EC2: Requires MFA (GetSessionToken) for specific EC2 operations

This example shows how you might create a policy that allows full access to all AWS API operations in Amazon EC2. However, it explicitly denies access to `StopInstances` and `TerminateInstances` API operations if the user is not authenticated using [multi-factor authentication \(MFA\) \(p. 111\)](#). To do this programmatically, the user must include optional `SerialNumber` and `TokenCode` values while calling the `GetSessionToken` operation. This operation returns temporary credentials that were authenticated using MFA. To learn more about `GetSessionToken`, see [GetSessionToken—temporary credentials for users in untrusted environments \(p. 310\)](#).

What does this policy do?

- The `AllowAllActionsForEC2` statement allows all Amazon EC2 actions.
- The `DenyStopAndTerminateWhenMFAIsNotPresent` statement denies the `StopInstances` and `TerminateInstances` actions when the MFA context is missing. This means that the actions are denied when the multi-factor authentication context is missing (meaning MFA was not used). A deny overrides the allow.

Note

The condition check for `MultiFactorAuthPresent` in the `Deny` statement should not be a `{"Bool": {"aws:MultiFactorAuthPresent": false}}` because that key is not present and cannot be evaluated when MFA is not used. So instead, use the `BoolIfExists` check to see whether the key is present before checking the value. For more information, see [...IfExists condition operators \(p. 650\)](#).

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAllActionsForEC2",
            "Effect": "Allow",
            "Action": "ec2:*",
            "Resource": "*"
        },
        {
            "Sid": "DenyStopAndTerminateWhenMFAIsNotPresent",
            "Effect": "Deny",

```

```
        "Action": [
            "ec2:StopInstances",
            "ec2:TerminateInstances"
        ],
        "Resource": "*",
        "Condition": {
            "BoolIfExists": { "aws:MultiFactorAuthPresent": false}
        }
    }
]
```

Amazon EC2: Limits terminating EC2 instances to an IP address range

This example shows how you might create a policy that limits EC2 instances by allowing the action, but explicitly denying access when the request comes from outside the specified IP range. The policy is useful when the IP addresses for your company are within the specified ranges. This policy grants the permissions necessary to complete this action from the AWS API or AWS CLI only. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#).

If this policy is used in combination with other policies that allow the `ec2:TerminateInstances` action (such as the [AmazonEC2FullAccess](#) AWS managed policy), then access is denied. This is because an explicit deny statement takes precedence over allow statements. For more information, see [the section called “Determining whether a request is allowed or denied within an account” \(p. 669\)](#).

Important

The `aws:SourceIp` condition key denies access to an AWS service, such as AWS CloudFormation, that makes calls on your behalf. For more information about using the `aws:SourceIp` condition key, see [AWS global condition context keys \(p. 692\)](#).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["ec2:TerminateInstances"],
            "Resource": ["*"]
        },
        {
            "Effect": "Deny",
            "Action": ["ec2:TerminateInstances"],
            "Condition": {
                "NotIpAddress": {
                    "aws:SourceIp": [
                        "192.0.2.0/24",
                        "203.0.113.0/24"
                    ]
                }
            },
            "Resource": ["*"]
        }
    ]
}
```

IAM: Access the policy simulator API

This example shows how you might create a policy that allows using the policy simulator API for policies attached to a user, group, or role in the current AWS account. This policy also allows access to simulate

less sensitive policies passed to the API as strings. This policy grants the permissions necessary to complete this action from the AWS API or AWS CLI only.

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

Note

To allow a user to access the policy simulator console to simulate policies attached to a user, group, or role in the current AWS account, see [IAM: Access the policy simulator console \(p. 415\)](#).

IAM: Access the policy simulator console

This example shows how you might create a policy that allows using the policy simulator console for policies attached to a user, group, or role in the current AWS account. This policy grants the permissions necessary to complete this action from the AWS API or AWS CLI only.

You can access the IAM Policy Simulator console at: <https://policysim.aws.amazon.com/>

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Action": [  
                "iam:GetGroup",  
                "iam:GetGroupPolicy",  
                "iam:GetPolicy",  
                "iam:GetPolicyVersion",  
                "iam:GetRole",  
                "iam:GetRolePolicy",  
                "iam:GetUser",  
                "iam:GetUserPolicy",  
                "iam>ListAttachedGroupPolicies",  
                "iam>ListAttachedRolePolicies",  
                "iam>ListAttachedUserPolicies",  
                "iam>ListGroups",  
                "iam>ListGroupPolicies",  
                "iam>ListGroupsForUser",  
                "iam>ListRolePolicies",  
                "iam>ListRoles",  
                "iam>ListUserPolicies",  
                "iam>ListUsers"  
            ],  
            "Effect": "Allow",  
            "Resource": "*"  
        }  
    ]  
}
```

IAM: Assume roles that have a specific tag

This example shows how you might create a policy that allows an IAM user to assume roles with the tag key-value pair `Project = ExampleCorpABC`. This policy grants the permissions necessary to complete this action from the AWS API or AWS CLI only. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#).

If a role with this tag exists in the same account as the user, then the user can assume that role. If a role with this tag exists in an account other than the user's, it requires additional permissions. The cross-account role's trust policy must also allow the user or all members of the user's account to assume the role. For information about using roles for cross-account access, see [Providing access to an IAM user in another AWS account that you own \(p. 171\)](#).

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AssumeTaggedRole",  
            "Effect": "Allow",  
            "Action": "sts:AssumeRole",  
            "Resource": "*",  
            "Condition": {  
                "StringEquals": {"iam:ResourceTag/Project": "ExampleCorpABC"}  
            }  
        }  
    ]  
}
```

IAM: Allows and denies access to multiple services programmatically and in the console

This example shows how you might create a policy that allows full access to several services and limited self-managing access in IAM. It also denies access to the Amazon S3 logs bucket or the Amazon EC2 `i-1234567890abcdef0` instance. This policy also grants the necessary permissions to complete this action on the console. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#).

Warning

This policy allows full access to every action and resource in multiple services. This policy should be applied only to trusted administrators.

You can use this policy as a permissions boundary to define the maximum permissions that an identity-based policy can grant to an IAM user. For more information, see [Delegating responsibility to others using permissions boundaries \(p. 369\)](#). When the policy is used as a permissions boundary for a user, the statements define the following boundaries:

- The `AllowServices` statement allows full access to the specified AWS services. This means that the user's actions in these services are limited only by the permissions policies that are attached to the user.
- The `AllowIAMConsoleForCredentials` statement allows access to list all IAM users. This access is necessary to navigate the **Users** page in the AWS Management Console. It also allows viewing the password requirements for the account, which is necessary for the user to change their own password.
- The `AllowManageOwnPasswordAndAccessKeys` statement allows the users manage only their own console password and programmatic access keys. This is important because if another policy gives a

user full IAM access, that user could then change their own or other users' permissions. This statement prevents that from happening.

- The `DenyS3Logs` statement explicitly denies access to the `logs` bucket. This policy enforces company restrictions on the user.
- The `DenyEC2Production` statement explicitly denies access to the `i-1234567890abcdef0` instance.

This policy does not allow access to other services or actions. When the policy is used as a permissions boundary on a user, even if other policies attached to the user allow those actions, AWS denies the request.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowServices",
            "Effect": "Allow",
            "Action": [
                "s3:*",
                "cloudwatch:*",
                "ec2:)"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowIAMConsoleForCredentials",
            "Effect": "Allow",
            "Action": [
                "iam>ListUsers",
                "iam:GetAccountPasswordPolicy"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowManageOwnPasswordAndAccessKeys",
            "Effect": "Allow",
            "Action": [
                "iam>*AccessKey*",
                "iam:ChangePassword",
                "iam:GetUser",
                "iam>*LoginProfile*"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "DenyS3Logs",
            "Effect": "Deny",
            "Action": "s3:*",
            "Resource": [
                "arn:aws:s3:::logs",
                "arn:aws:s3:::logs/*"
            ]
        },
        {
            "Sid": "DenyEC2Production",
            "Effect": "Deny",
            "Action": "ec2:)",
            "Resource": "arn:aws:ec2::*:instance/i-1234567890abcdef0"
        }
    ]
}
```

IAM: Add a specific tag to a user with a specific tag

This example shows how you might create a policy that allows adding the tag key `Department` with the tag values `Marketing`, `Development`, or `QualityAssurance` to an IAM user. That user must already include the tag key-value pair `JobFunction = manager`. You can use this policy to require that a manager belong to only one of three departments. This policy also grants the necessary permissions to complete this action on the console. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#).

The `ListTagsForAllUsers` statement allows the viewing of tags for all users in your account.

The first condition in the `TagManagerWithSpecificDepartment` statement uses the `StringEquals` condition operator. The condition returns true if both parts of the condition are true. The user to be tagged must already have the `JobFunction=Manager` tag. The request must include the `Department` tag key with one of the listed tag values.

The second condition uses the `ForAllValues:StringEquals` condition operator. The condition returns true if all of the tag keys in the request match the key in the policy. This means that the only tag key in the request must be `Department`. For more information about using `ForAllValues`, see [Creating a condition with multiple keys or values \(p. 652\)](#).

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "ListTagsForAllUsers",  
            "Effect": "Allow",  
            "Action": [  
                "iam>ListUserTags",  
                "iam>ListUsers"  
            ],  
            "Resource": "*"  
        },  
        {  
            "Sid": "TagManagerWithSpecificDepartment",  
            "Effect": "Allow",  
            "Action": "iam:TagUser",  
            "Resource": "*",  
            "Condition": {"StringEquals": {  
                "iam:ResourceTag/JobFunction": "Manager",  
                "aws:RequestTag/Department": [  
                    "Marketing",  
                    "Development",  
                    "QualityAssurance"  
                ]  
            }},  
            "ForAllValues:StringEquals": {"aws:TagKeys": "Department"}  
        }  
    ]  
}
```

IAM: Add a specific tag with specific values

This example shows how you might create a policy that allows adding only the tag key `CostCenter` and either the tag value `A-123` or the tag value `B-456` to any IAM user or role. You can use this policy to limit tagging to a specific tag key and set of tag values. This policy also grants the necessary permissions to complete this action on the console. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#).

The `ConsoleDisplay` statement allows the viewing of tags for all users and roles in your account.

The first condition in the `AddTag` statement uses the `StringEquals` condition operator. The condition returns true if the request includes the `CostCenter` tag key with one of the listed tag values.

The second condition uses the `ForAllValues:StringEquals` condition operator. The condition returns true if all of the tag keys in the request match the key in the policy. This means that the only tag key in the request must be `CostCenter`. For more information about using `ForAllValues`, see [Creating a condition with multiple keys or values \(p. 652\)](#).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ConsoleDisplay",
            "Effect": "Allow",
            "Action": [
                "iam:GetRole",
                "iam:GetUser",
                "iam>ListRoles",
                "iam>ListRoleTags",
                "iam>ListUsers",
                "iam>ListUserTags"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AddTag",
            "Effect": "Allow",
            "Action": [
                "iam:TagUser",
                "iam:TagRole"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/CostCenterCostCenter"}
            }
        }
    ]
}
```

IAM: Create new users only with specific tags

This example shows how you might create a policy that allows the creation of IAM users but only with one or both of the `Department` and `JobFunction` tag keys. The `Department` tag key must have either the `Development` or `QualityAssurance` tag value. The `JobFunction` tag key must have the `Employee` tag value. You can use this policy to require that new users have a specific job function and department. This policy grants the permissions necessary to complete this action from the AWS API or AWS CLI only. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#).

The first condition in the statement uses the `StringEqualsIfExists` condition operator. If a tag with the `Department` or `JobFunction` key is present in the request, then the tag must have the specified value. If neither key is present, then this condition is evaluated as true. The only way that the condition evaluates as false is if one of the specified condition keys is present in the request, but has a

different value than those allowed. For more information about using `IfExists`, see [...IfExists condition operators \(p. 650\)](#).

The second condition uses the `ForAllValues:StringEquals` condition operator. The condition returns true if there's a match between all every one of the specified tag keys specified in the request, and at least one value in the policy. This means that all of the tags in the request must be in this list. However, the request can include only one of the tags in the list. For example, you can create an IAM user with only the `Department=QualityAssurance` tag. However, you cannot create an IAM user with the `JobFunction=employee` tag and the `Project=core` tag. For more information about using `ForAllValues`, see [Creating a condition with multiple keys or values \(p. 652\)](#).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "TagUsersWithOnlyTheseTags",
            "Effect": "Allow",
            "Action": [
                "iam:CreateUser",
                "iam:TagUser"
            ],
            "Resource": "*",
            "Condition": {
                "StringEqualsIfExists": {
                    "aws:RequestTag/DepartmentDevelopment",
                        "QualityAssurance"
                    ],
                    "aws:RequestTag/JobFunctionEmployee"
                },
                "ForAllValues:StringEquals": {
                    "aws:TagKeys": [
                        "Department",
                        "JobFunction"
                    ]
                }
            }
        }
    ]
}
```

IAM: Generate and retrieve IAM credential reports

This example shows how you might create a policy that allows users to generate and download a report that lists all IAM users in their AWS account. The report includes the status of the users' credentials, including passwords, access keys, MFA devices, and signing certificates. This policy grants the permissions necessary to complete this action from the AWS API or AWS CLI only.

For more information about credential reports, see [Getting credential reports for your AWS account \(p. 148\)](#).

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

```
}
```

IAM: Allows managing a group's membership programmatically and in the console

This example shows how you might create a policy that allows updating the membership of the group called `MarketingTeam`. This policy also grants the necessary permissions to complete this action on the console. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#).

What does this policy do?

- The `ViewGroups` statement allows the user to list all the users and groups in the AWS Management Console. It also allows the user to view basic information about the users in the account. These permissions must be in their own statement because they do not support or do not need to specify a resource ARN. Instead the permissions specify `"Resource" : "*"`.
- The `ViewEditThisGroup` statement allows the user to view information about the `MarketingTeam` group, and to add and remove users from that group.

This policy does not allow the user to view or edit the permissions of the users or the `MarketingTeam` group.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewGroups",
            "Effect": "Allow",
            "Action": [
                "iam>ListGroups",
                "iam>ListUsers",
                "iam GetUser",
                "iam>ListGroupsForUser"
            ],
            "Resource": "*"
        },
        {
            "Sid": "ViewEditThisGroup",
            "Effect": "Allow",
            "Action": [
                "iam>AddUserToGroup",
                "iam RemoveUserFromGroup",
                "iam GetGroup"
            ],
            "Resource": "arn:aws:iam::*:group/MarketingTeam"
        }
    ]
}
```

IAM: Manage a specific tag

This example shows how you might create a policy that allows adding and removing the IAM tag with the tag key `Department`. This policy does not limit the value of the `Department` tag. This policy grants the permissions necessary to complete this action from the AWS API or AWS CLI only. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#).

```
{
```

```
"Version": "2012-10-17",
"Statement": [
    "Effect": "Allow",
    "Action": [
        "iam:TagUser",
        "iam:TagRole",
        "iam:UntagUser",
        "iam:UntagRole"
    ],
    "Resource": "*",
    "Condition": {"ForAllValues:StringEquals": {"aws:TagKeys": "Department"}}
}
```

IAM: Pass an IAM role to a specific AWS service

This example shows how you might create a policy that allows passing any IAM service role to the Amazon CloudWatch service. This policy grants the permissions necessary to complete this action from the AWS API or AWS CLI only. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#).

A service role is an IAM role that specifies an AWS service as the principal that can assume the role. This allows the service to assume the role and access resources in other services on your behalf. To allow Amazon CloudWatch to assume the role that you pass, you must specify the `cloudwatch.amazonaws.com` service principal as the principal in the trust policy of your role. The service principal is defined by the service. To learn the service principal for a service, see the documentation for that service. For some services, see [AWS services that work with IAM \(p. 611\)](#) and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service. Search for `amazonaws.com` to view the service principal.

To learn more about passing a service role to a service, see [Granting a user permissions to pass a role to an AWS service \(p. 251\)](#).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "*",
            "Condition": {
                "StringEquals": {"iam:PassedToService": "cloudwatch.amazonaws.com"}
            }
        }
    ]
}
```

IAM: Allows read-only access to the IAM console without reporting

This example shows how you might create a policy that allows IAM users to perform any IAM action that begins with the string `Get`, `List`, or `Generate`. As users work with the console, the console makes requests to IAM to list groups, users, roles, and policies, and to generate reports about those resources.

The asterisk acts as a wildcard. When you use `iam:Get*` in a policy, the resulting permissions include all IAM actions that begin with `Get`, such as `GetUser` and `GetRole`. Wildcards are useful if new types of

entities are added to IAM in the future. In that case, the permissions granted by the policy automatically allow the user to list and get the details about those new entities.

This policy can't be used for reporting purposes.

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

IAM: Allows read-only access to the IAM console

This example shows how you might create a policy that allows IAM users to perform any IAM action that begins with the string Get, List, or Generate. As users work with the IAM console, the console makes requests to list groups, users, roles, and policies, and to generate reports about those resources.

The asterisk acts as a wildcard. When you use `iam:Get*` in a policy, the resulting permissions include all IAM actions that begin with Get, such as `GetUser` and `GetRole`. Using a wildcard is beneficial, especially if new types of entities are added to IAM in the future. In that case, the permissions granted by the policy automatically allow the user to list and get the details about those new entities.

Use this policy for console access or for reporting purposes.

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

IAM: Allows specific IAM users to manage a group programmatically and in the console

This example shows how you might create a policy that allows specific IAM users to manage the `AllUsers` group. This policy also grants the necessary permissions to complete this action on the console. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#).

What does this policy do?

- The `AllowAllUsersToListAllGroups` statement allows listing all groups. This is necessary for console access. This permission must be in its own statement because it does not support a resource ARN. Instead the permissions specify `"Resource" : "*"`.
- The `AllowAllUsersToViewAndManageThisGroup` statement allows all group actions that can be performed on the group resource type. It does not allow the `ListGroupsForUser` action, which can

be performed on a user resource type and not a group resource type. For more information about the resource types that you can specify for an IAM action, see [Actions, Resources, and Condition Keys for AWS Identity and Access Management](#).

- The `LimitGroupManagementAccessToSpecificUsers` statement denies users with the specified names access to write and permissions management group actions. When a user specified in the policy attempts to make changes to the group, this statement does not deny the request. That request is allowed by the `AllowAllUsersToViewAndManageThisGroup` statement. If other users attempt to perform these operations, the request is denied. You can view the IAM actions that are defined with the **Write or Permissions management** access levels while creating this policy in the IAM console. To do this, switch from the **JSON** tab to the **Visual editor** tab. For more information about access levels, see [Actions, Resources, and Condition Keys for AWS Identity and Access Management](#).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAllUsersToListAllGroups",
            "Effect": "Allow",
            "Action": "iam>ListGroups",
            "Resource": "*"
        },
        {
            "Sid": "AllowAllUsersToViewAndManageThisGroup",
            "Effect": "Allow",
            "Action": "iam:ListGroup*",
            "Resource": "arn:aws:iam::*:group/AllUsers"
        },
        {
            "Sid": "LimitGroupManagementAccessToSpecificUsers",
            "Effect": "Deny",
            "Action": [
                "iam>AddUserToGroup",
                "iam>CreateGroup",
                "iam>RemoveUserFromGroup",
                "iam>DeleteGroup",
                "iam>AttachGroupPolicy",
                "iam>UpdateGroup",
                "iam>DetachGroupPolicy",
                "iam>DeleteGroupPolicy",
                "iam>PutGroupPolicy"
            ],
            "Resource": "arn:aws:iam::*:group/AllUsers",
            "Condition": {
                "StringNotEquals": {
                    "aws:username": [
                        "srodriguez",
                        "mjackson",
                        "adesai"
                    ]
                }
            }
        }
    ]
}
```

IAM: Allows setting the account password requirements programmatically and in the console

This example shows how you might create a policy that allows a user to view and update their account's password requirements. The password requirements specify the complexity requirements and mandatory

rotation periods for the account members' passwords. This policy also grants the necessary permissions to complete this action on the console.

To learn how to set the account password requirements policy for your account, see [Setting an account password policy for IAM users \(p. 93\)](#).

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

IAM: Access the policy simulator API based on user path

This example shows how you might create a policy that allows using the policy simulator API only for those users that have the path `Department/Development`. This policy grants the permissions necessary to complete this action from the AWS API or AWS CLI only. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#).

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Action": [  
                "iam:GetContextKeysForPrincipalPolicy",  
                "iam:SimulatePrincipalPolicy"  
            ],  
            "Effect": "Allow",  
            "Resource": "arn:aws:iam::*:user/Department/Development/*"  
        }  
    ]  
}
```

Note

To create a policy that allows using the policy simulator console for those users that have the path `Department/Development`, see [IAM: Access the policy simulator console based on user path \(p. 425\)](#).

IAM: Access the policy simulator console based on user path

This example shows how you might create a policy that allows using the policy simulator console only for those users that have the path `Department/Development`. This policy grants the permissions necessary to complete this action from the AWS API or AWS CLI only. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#).

You can access the IAM Policy Simulator at: <https://policysim.aws.amazon.com/>

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Action": [  
                "iam:GetContextKeysForPrincipalPolicy",  
                "iam:SimulatePrincipalPolicy"  
            ],  
            "Effect": "Allow",  
            "Resource": "arn:aws:iam::*:user/Department/Development/*"  
        }  
    ]  
}
```

```

    "Action": [
        "iam:GetPolicy",
        "iam:GetUserPolicy"
    ],
    "Effect": "Allow",
    "Resource": "*"
},
{
    "Action": [
        "iam:GetUser",
        "iam>ListAttachedUserPolicies",
        "iam>ListGroupsForUser",
        "iam>ListUserPolicies",
        "iam>ListUsers"
    ],
    "Effect": "Allow",
    "Resource": "arn:aws:iam::*:user/Department/Development/*"
}
]
}

```

IAM: Allows IAM users to self-manage an MFA device

This example shows how you might create a policy that allows IAM users to self-manage their [multi-factor authentication \(MFA\)](#) (p. 111) device. This policy grants the permissions necessary to complete this action from the AWS API or AWS CLI only.

Note

If you add these permissions for a user that is signed in to AWS, they might need to sign out and back in to see these changes.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowListActions",
            "Effect": "Allow",
            "Action": [
                "iam>ListUsers",
                "iam>ListVirtualMFADevices"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowIndividualUserToListOnlyTheirOwnMFA",
            "Effect": "Allow",
            "Action": [
                "iam>ListMFADevices"
            ],
            "Resource": [
                "arn:aws:iam::*:mfa/*",
                "arn:aws:iam::*:user/${aws:username}"
            ]
        },
        {
            "Sid": "AllowIndividualUserToManageTheirOwnMFA",
            "Effect": "Allow",
            "Action": [
                "iam>CreateVirtualMFADevice",
                "iam>DeleteVirtualMFADevice",
                "iam>EnableMFADevice",
                "iam>ResyncMFADevice"
            ],
            "Resource": [

```

```

        "arn:aws:iam::*:mfa/${aws:username}",
        "arn:aws:iam::*:user/${aws:username}"
    ],
},
{
    "Sid": "AllowIndividualUserToDeactivateOnlyTheirOwnMFAOnlyWhenUsingMFA",
    "Effect": "Allow",
    "Action": [
        "iam:DeactivateMFADevice"
    ],
    "Resource": [
        "arn:aws:iam::*:mfa/${aws:username}",
        "arn:aws:iam::*:user/${aws:username}"
    ],
    "Condition": {
        "Bool": {
            "aws:MultiFactorAuthPresent": "true"
        }
    }
},
{
    "Sid": "BlockMostAccessUnlessSignedInWithMFA",
    "Effect": "Deny",
    "NotAction": [
        "iam>CreateVirtualMFADevice",
        "iam:EnableMFADevice",
        "iam>ListMFADevices",
        "iam>ListUsers",
        "iam>ListVirtualMFADevices",
        "iam:ResyncMFADevice"
    ],
    "Resource": "*",
    "Condition": {
        "BoolIfExists": {
            "aws:MultiFactorAuthPresent": "false"
        }
    }
}
]
}

```

IAM: Allows IAM users to rotate their own credentials programmatically and in the console

This example shows how you might create a policy that allows IAM users to rotate their own access keys, signing certificates, service specific credentials, and passwords. This policy also grants the necessary permissions to complete this action on the console.

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

```

```
        "iam:ChangePassword",
        "iam:GetUser",
        "iam:*ServiceSpecificCredential*",
        "iam:*SigningCertificate"
    ],
}
]
}
}
```

To learn how a user can change their own password in the console, see [the section called “How an IAM user changes their own password” \(p. 101\)](#).

IAM: View service last accessed information for an Organizations policy

This example shows how you might create a policy that allows viewing service last accessed information for a specific Organizations policy. This policy allows retrieving data for the service control policy (SCP) with the p-policy123 ID. The person who generates and views the report must be authenticated using AWS Organizations management account credentials. This policy allows the requester to retrieve the data for any Organizations entity in their organization. This policy also grants the necessary permissions to complete this action on the console. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#).

For important information about last accessed information, including permissions required, troubleshooting, and supported Regions, see [Refining permissions in AWS using last accessed information \(p. 472\)](#).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowOrgsReadOnlyAndIamGetReport",
            "Effect": "Allow",
            "Action": [
                "iam:GetOrganizationsAccessReport",
                "organizations:Describe*",
                "organizations>List*"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowGenerateReportOnlyForThePolicy",
            "Effect": "Allow",
            "Action": "iam:GenerateOrganizationsAccessReport",
            "Resource": "*",
            "Condition": {
                "StringEquals": {"iam:OrganizationsPolicyId": "p-policy123"}
            }
        }
    ]
}
```

IAM: Limits managed policies that can be applied to an IAM user, group, or role

This example shows how you might create a policy that limits customer managed and AWS managed policies that can be applied to an IAM user, group, or role. This policy grants the permissions necessary

to complete this action from the AWS API or AWS CLI only. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:AttachUserPolicy",
                "iam:DetachUserPolicy"
            ],
            "Resource": "*",
            "Condition": {
                "ArnEquals": {
                    "iam:PolicyARN": [
                        "arn:aws:iam::*:policy/policy-name-1",
                        "arn:aws:iam::*:policy/policy-name-2"
                    ]
                }
            }
        }
    }
}
```

AWS Lambda: Allows a Lambda function to access an Amazon DynamoDB table

This example shows how you might create a policy that allows read and write access to a specific Amazon DynamoDB table. The policy also allows writing log files to CloudWatch Logs. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#).

To use this policy, attach the policy to a Lambda [service role \(p. 229\)](#). A service role is a role that you create in your account to allow a service to perform actions on your behalf. That service role must include AWS Lambda as the principal in the trust policy. For details about how to use this policy, see [How to Create an AWS IAM Policy to Grant AWS Lambda Access to an Amazon DynamoDB Table](#) in the [AWS Security Blog](#).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadWriteTable",
            "Effect": "Allow",
            "Action": [
                "dynamodb:BatchGetItem",
                "dynamodb:GetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:BatchWriteItem",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/SampleTable"
        },
        {
            "Sid": "GetStreamRecords",
            "Effect": "Allow",
            "Action": "dynamodb:GetRecords",
            "Resource": "arn:aws:dynamodb:*:*:table/SampleTable/stream/*"
        }
    ]
}
```

```
        "Sid": "WriteLogStreamsAndGroups",
        "Effect": "Allow",
        "Action": [
            "logs>CreateLogStream",
            "logs:PutLogEvents"
        ],
        "Resource": "*"
    },
    {
        "Sid": "CreateLogGroup",
        "Effect": "Allow",
        "Action": "logs>CreateLogGroup",
        "Resource": "*"
    }
]
```

Amazon RDS: Allows full RDS database access within a specific Region

This example shows how you might create a policy that allows full RDS database access within a specific Region. This policy grants the permissions necessary to complete this action from the AWS API or AWS CLI only. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "rds:*",
            "Resource": ["arn:aws:rds:region::*"]
        },
        {
            "Effect": "Allow",
            "Action": ["rds:Describe*"],
            "Resource": ["*"]
        }
    ]
}
```

Amazon RDS: Allows restoring RDS databases, programmatically and in the console

This example shows how you might create a policy that allows restoring RDS databases. This policy also grants the necessary permissions to complete this action on the console.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "ec2:Describe*",  
                "rds>CreateDBParameterGroup",  
                "rds>CreateDBSnapshot",  
                "rds>DeleteDBSnapshot",  
                "rds:Describe*",  
                "rds:DownloadDBLogFilePortion".  
            ]  
        }  
    ]  
}
```

```

        "rds>List*",
        "rds:ModifyDBInstance",
        "rds:ModifyDBParameterGroup",
        "rds:ModifyOptionGroup",
        "rds:RebootDBInstance",
        "rds:RestoreDBInstanceFromDBSnapshot",
        "rds:RestoreDBInstanceToPointInTime"
    ],
    "Resource": "*"
}
]
}

```

Amazon RDS: Allows tag owners full access to RDS resources that they have tagged

This example shows how you might create a policy that allows tag owners full access to RDS resources that they have tagged. This policy grants the permissions necessary to complete this action from the AWS API or AWS CLI only.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "rds:Describe*",
                "rds>List*"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": [
                "rds>DeleteDBInstance",
                "rds:RebootDBInstance",
                "rds:ModifyDBInstance"
            ],
            "Effect": "Allow",
            "Resource": "*",
            "Condition": {
                "StringEqualsIgnoreCase": {"rds:db-tag/Owner": "${aws:username}"}
            }
        },
        {
            "Action": [
                "rds:ModifyOptionGroup",
                "rds>DeleteOptionGroup"
            ],
            "Effect": "Allow",
            "Resource": "*",
            "Condition": {
                "StringEqualsIgnoreCase": {"rds:og-tag/Owner": "${aws:username}"}
            }
        },
        {
            "Action": [
                "rds:ModifyDBParameterGroup",
                "rds:ResetDBParameterGroup"
            ],
            "Effect": "Allow",
            "Resource": "*",
            "Condition": {
                "StringEqualsIgnoreCase": {"rds:pg-tag/Owner": "${aws:username}"}
            }
        }
    ]
}

```

```

        }
    },
    {
        "Action": [
            "rds:AuthorizeDBSecurityGroupIngress",
            "rds:RevokeDBSecurityGroupIngress",
            "rds:DeleteDBSecurityGroup"
        ],
        "Effect": "Allow",
        "Resource": "*",
        "Condition": {
            "StringEqualsIgnoreCase": {"rds:secgrp-tag/Owner": "${aws:username}"}
        }
    },
    {
        "Action": [
            "rds:DeleteDBSnapshot",
            "rds:RestoreDBInstanceFromDBSnapshot"
        ],
        "Effect": "Allow",
        "Resource": "*",
        "Condition": {
            "StringEqualsIgnoreCase": {"rds:snapshot-tag/Owner": "${aws:username}"}
        }
    },
    {
        "Action": [
            "rds:ModifyDBSubnetGroup",
            "rds:DeleteDBSubnetGroup"
        ],
        "Effect": "Allow",
        "Resource": "*",
        "Condition": {
            "StringEqualsIgnoreCase": {"rds:subgrp-tag/Owner": "${aws:username}"}
        }
    },
    {
        "Action": [
            "rds:ModifyEventSubscription",
            "rds:AddSourceIdentifierToSubscription",
            "rds:RemoveSourceIdentifierFromSubscription",
            "rds:DeleteEventSubscription"
        ],
        "Effect": "Allow",
        "Resource": "*",
        "Condition": {
            "StringEqualsIgnoreCase": {"rds:es-tag/Owner": "${aws:username}"}
        }
    }
]
}

```

Amazon S3: Allows Amazon Cognito users to access objects in their bucket

This example shows how you might create a policy that allows Amazon Cognito users to access objects in a specific S3 bucket. This policy allows access only to objects with a name that includes cognito, the name of the application, and the federated user's ID, represented by the \${cognito-identity.amazonaws.com:sub} variable. This policy grants the permissions necessary to complete this action from the AWS API or AWS CLI only. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#).

Note

The 'sub' value used in the object key is not the user's sub value in the User Pool, it is the identity id associated with the user in the Identity Pool.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "ListYourObjects",  
            "Effect": "Allow",  
            "Action": "s3>ListBucket",  
            "Resource": ["arn:aws:s3:::bucket-name"],  
            "Condition": {  
                "StringLike": {  
                    "s3:prefix": ["cognito/application-name/${cognito-  
identity.amazonaws.com:sub}"]  
                }  
            }  
        },  
        {  
            "Sid": "ReadWriteDeleteYourObjects",  
            "Effect": "Allow",  
            "Action": [  
                "s3:GetObject",  
                "s3:PutObject",  
                "s3>DeleteObject"  
            ],  
            "Resource": [  
                "arn:aws:s3:::bucket-name/cognito/application-name/${cognito-  
identity.amazonaws.com:sub}",  
                "arn:aws:s3:::bucket-name/cognito/application-name/${cognito-  
identity.amazonaws.com:sub}/*"  
            ]  
        }  
    ]  
}
```

Amazon Cognito provides authentication, authorization, and user management for your web and mobile apps. Your users can sign in directly with a user name and password, or through a third party such as Facebook, Amazon, or Google.

The two main components of Amazon Cognito are user pools and identity pools. User pools are user directories that provide sign-up and sign-in options for your app users. Identity pools enable you to grant your users access to other AWS services. You can use identity pools and user pools separately or together.

For more information about Amazon Cognito, see the following:

- [Amazon Cognito Identity](#) in the *AWS Mobile SDK for Android Developer Guide*
- [Amazon Cognito Identity](#) in the *AWS Mobile SDK for iOS Developer Guide*

Amazon S3: Allows federated users access to their S3 home directory, programmatically and in the console

This example shows how you might create a policy that allows federated users to access their own home directory bucket object in S3. The home directory is a bucket that includes a home folder and folders for individual federated users. This policy also grants the necessary permissions to complete this action on the console. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy](#) (p. 439) or [edit a policy](#) (p. 465).

The `#{aws:userid}` variable in this policy resolves to `role-id:specified-name`. The `role-id` part of the federated user ID is a unique identifier assigned to the federated user's role during creation. For more information, see [Unique identifiers \(p. 604\)](#). The `specified-name` is the [RoleSessionName parameter](#) passed to the `AssumeRoleWithWebIdentity` request when the federated user assumed their role.

You can view the role ID using the AWS CLI command `aws iam get-role --role-name specified-name`. For example, imagine that you specify the friendly name `John` and the CLI returns the role ID `AROAXXT2NJT7D3SIQN7Z6`. In this case, the federated user ID is `AROAXXT2NJT7D3SIQN7Z6:John`. This policy then allows the federated user `John` to access the Amazon S3 bucket with prefix `AROAXXT2NJT7D3SIQN7Z6:John`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3>ListAllMyBuckets",
                "s3:GetBucketLocation"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "s3>ListBucket",
            "Resource": "arn:aws:s3:::bucket-name",
            "Condition": {
                "StringLike": {
                    "s3:prefix": [
                        "",
                        "home/",
                        "home/${aws:userid}/*"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "s3:*",
            "Resource": [
                "arn:aws:s3:::bucket-name/home/${aws:userid}",
                "arn:aws:s3:::bucket-name/home/${aws:userid}/*"
            ]
        }
    ]
}
```

Amazon S3: S3 Bucket access, but production bucket denied without recent MFA

This example shows how you might create a policy that allows an Amazon S3 administrator to access any bucket, including updating, adding, and deleting objects. However, it explicitly denies access to the `Production` bucket if the user has not signed in using [multi-factor authentication \(MFA\) \(p. 111\)](#) within the last thirty minutes. This policy grants the permissions necessary to perform this action in the console or programmatically using the AWS CLI or AWS API. To use this policy, replace the [*italicized placeholder text*](#) in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#).

This policy never allows programmatic access to the `Production` bucket using long-term user access keys. This is accomplished using the `aws:MultiFactorAuthAge` condition key with the

NumericGreaterThanIfExists condition operator. This policy condition returns true if MFA is not present or if the age of the MFA is greater than 30 minutes. In those situations, access is denied. To access the Production bucket programmatically, the S3 administrator must use temporary credentials that were generated in the last 30 minutes using the [GetSessionToken \(p. 310\)](#) API operation.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ListAllS3Buckets",
            "Effect": "Allow",
            "Action": ["s3>ListAllMyBuckets"],
            "Resource": "arn:aws:s3:::/*"
        },
        {
            "Sid": "AllowBucketLevelActions",
            "Effect": "Allow",
            "Action": [
                "s3>ListBucket",
                "s3:GetBucketLocation"
            ],
            "Resource": "arn:aws:s3:::/*"
        },
        {
            "Sid": "AllowBucketObjectActions",
            "Effect": "Allow",
            "Action": [
                "s3>PutObject",
                "s3>PutObjectAcl",
                "s3>GetObject",
                "s3>GetObjectAcl",
                "s3>DeleteObject"
            ],
            "Resource": "arn:aws:s3:::/*/*"
        },
        {
            "Sid": "RequireMFAForProductionBucket",
            "Effect": "Deny",
            "Action": "s3:*",
            "Resource": [
                "arn:aws:s3:::Production/*",
                "arn:aws:s3:::Production"
            ],
            "Condition": {
                "NumericGreaterThanIfExists": {"aws:MultiFactorAuthAge": "1800"}
            }
        }
    ]
}
```

Amazon S3: Allows IAM users access to their S3 home directory, programmatically and in the console

This example shows how you might create a policy that allows IAM users to access their own home directory bucket object in S3. The home directory is a bucket that includes a home folder and folders for individual users. This policy also grants the necessary permissions to complete this action on the console. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#).

This policy will not work when using IAM roles because the `aws:username` variable is not available when using IAM roles. For details about principal key values, see [Principal key values \(p. 663\)](#).

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

Amazon S3: Limits managing to a specific S3 Bucket

This example shows how you might create a policy that limits managing an S3 bucket by allowing all S3 actions on the specific bucket, but explicitly denying access to every AWS service except Amazon S3. This policy also denies access to actions that can't be performed on an S3 bucket, such as s3>ListAllMyBuckets or s3>GetObject. This policy grants the permissions necessary to complete this action from the AWS API or AWS CLI only. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#).

If this policy is used in combination with other policies (such as the [AmazonS3FullAccess](#) or [AmazonEC2FullAccess](#) AWS managed policies) that allow actions denied by this policy, then access is denied. This is because an explicit deny statement takes precedence over allow statements. For more information, see [the section called “Determining whether a request is allowed or denied within an account” \(p. 669\)](#).

Warning

[NotAction \(p. 638\)](#) and [NotResource \(p. 640\)](#) are advanced policy elements that must be used with care. This policy denies access to every AWS service except Amazon S3. If you attach this policy to a user, any other policies that grant permissions to other services are ignored and access is denied.

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

```

```
"Effect": "Allow",
"Action": "s3:*",
"Resource": [
    "arn:aws:s3:::bucket-name",
    "arn:aws:s3:::bucket-name/*"
]
},
{
    "Effect": "Deny",
    "NotAction": "s3:*",
    "NotResource": [
        "arn:aws:s3:::bucket-name",
        "arn:aws:s3:::bucket-name/*"
    ]
}
]
```

Amazon S3: Allows read and write access to objects in an S3 Bucket

This example shows how you might create a policy that allows Read and Write access to objects in a specific S3 bucket. This policy grants the permissions necessary to complete this action from the AWS API or AWS CLI only. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#).

The `s3:*Object` action uses a wildcard as part of the action name. The `AllObjectActions` statement allows the `GetObject`, `DeleteObject`, `PutObject`, and any other Amazon S3 action that ends with the word "Object".

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ListObjectsInBucket",
            "Effect": "Allow",
            "Action": ["s3>ListBucket"],
            "Resource": ["arn:aws:s3:::bucket-name"]
        },
        {
            "Sid": "AllObjectActions",
            "Effect": "Allow",
            "Action": "s3:*Object",
            "Resource": ["arn:aws:s3:::bucket-name/*"]
        }
    ]
}
```

Note

To allow Read and Write access to an object in an Amazon S3 bucket and also include additional permissions for console access, see [Amazon S3: Allows read and write access to objects in an S3 Bucket, programmatically and in the console \(p. 437\)](#).

Amazon S3: Allows read and write access to objects in an S3 Bucket, programmatically and in the console

This example shows how you might create a policy that allows Read and Write access to objects in a specific S3 bucket. This policy also grants the necessary permissions to complete this action on the

console. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#).

The `s3:*Object` action uses a wildcard as part of the action name. The `AllObjectActions` statement allows the `GetObject`, `DeleteObject`, `PutObject`, and any other Amazon S3 action that ends with the word "Object".

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "ConsoleAccess",  
            "Effect": "Allow",  
            "Action": [  
                "s3:GetAccountPublicAccessBlock",  
                "s3:GetBucketAcl",  
                "s3:GetBucketLocation",  
                "s3:GetBucketPolicyStatus",  
                "s3:GetBucketPublicAccessBlock",  
                "s3>ListAllMyBuckets"  
            ],  
            "Resource": "*"  
        },  
        {  
            "Sid": "ListObjectsInBucket",  
            "Effect": "Allow",  
            "Action": "s3>ListBucket",  
            "Resource": ["arn:aws:s3:::bucket-name"]  
        },  
        {  
            "Sid": "AllObjectActions",  
            "Effect": "Allow",  
            "Action": "s3:*Object",  
            "Resource": ["arn:aws:s3:::bucket-name/*"]  
        }  
    ]  
}
```

Managing IAM policies

IAM gives you the tools to create and manage all types of IAM policies (managed policies and inline policies). To add permissions to an IAM identity (IAM user, group, or role), you create a policy and then attach the policy to the identity. You can attach multiple policies to an identity, and each policy can contain multiple permissions.

Consult these resources for details:

- For more information about the different types of IAM policies, see [Policies and permissions in IAM \(p. 351\)](#).
- For general information about using policies within IAM, see [Access management for AWS resources \(p. 350\)](#).
- For information about how permissions are evaluated when multiple policies are in effect for a given IAM identity, see [Policy evaluation logic \(p. 666\)](#).
- The number and size of IAM resources in an AWS account are limited. For more information, see [IAM and STS quotas \(p. 606\)](#).

Topics

- [Creating IAM policies \(p. 439\)](#)
- [Validating IAM policy grammar \(p. 444\)](#)
- [Testing IAM policies with the IAM policy simulator \(p. 445\)](#)
- [Adding and removing IAM identity permissions \(p. 454\)](#)
- [Versioning IAM policies \(p. 462\)](#)
- [Editing IAM policies \(p. 465\)](#)
- [Deleting IAM policies \(p. 469\)](#)
- [Refining permissions in AWS using last accessed information \(p. 472\)](#)

Creating IAM policies

A [policy \(p. 351\)](#) is an entity that, when attached to an identity or resource, defines their permissions. You can use the AWS Management Console, AWS CLI, or AWS API to create *customer managed policies* in IAM. Customer managed policies are standalone policies that you administer in your own AWS account. You can then attach the policies to identities (users, groups, and roles) in your AWS account.

A policy that is attached to an identity in IAM is known as an *identity-based policy*. Identity-based policies can include AWS managed policies, customer managed policies, and inline policies. AWS managed policies are created and managed by AWS. You can use them, but you can't manage them. An inline policy is one that you create and embed directly to an IAM group, user, or role. Inline policies can't be reused on other identities or managed outside of the identity where it exists. For more information, see [Adding and removing IAM identity permissions \(p. 454\)](#).

As a best practice, [use customer managed policies instead of inline policies \(p. 530\)](#). It's also best to use customer managed policies instead of AWS managed policies. AWS managed policies usually provide broad administrative or read-only permissions. For greatest security, [grant least privilege \(p. 529\)](#), which is granting only the permissions required to perform specific job tasks.

You can use the AWS Management Console, AWS CLI, or AWS API to create customer managed policies in IAM.

Creating IAM policies (console)

A [policy \(p. 351\)](#) is an entity that, when attached to an identity or resource, defines their permissions. You can use the AWS Management Console to create *customer managed policies* in IAM. Customer managed policies are standalone policies that you administer in your own AWS account. You can then attach the policies to identities (users, groups, and roles) in your AWS account.

Topics

- [Creating IAM policies \(console\) \(p. 439\)](#)
- [Creating policies on the JSON tab \(p. 440\)](#)
- [Creating policies with the visual editor \(p. 440\)](#)
- [Importing existing managed policies \(p. 442\)](#)

Creating IAM policies (console)

You can create a customer managed policy in the AWS Management Console using one of the following methods:

- **JSON (p. 440)** — Paste and customize a published [example identity-based policy \(p. 389\)](#).
- **Visual editor (p. 440)** — Construct a new policy from scratch in the visual editor. If you use the visual editor, you do not have to understand JSON syntax.

- **Import (p. 442)** — Import and customize a managed policy from within your account. You can import an AWS managed policy or a customer managed policy that you previously created.

The number and size of IAM resources in an AWS account are limited. For more information, see [IAM and STS quotas \(p. 606\)](#).

Creating policies on the JSON tab

You can type or paste policies in JSON by choosing the **JSON** tab. This method is useful for copying an [example policy \(p. 389\)](#) to use in your account. Or, you can type your own JSON policy document in the JSON editor. You can also use the **JSON** tab to toggle between the visual editor and JSON to compare the views.

A JSON [policy \(p. 351\)](#) document consists of one or more statements. Each statement should contain all the actions that share the same effect (Allow or Deny) and support the same resources and conditions. If one action requires you to specify all resources ("*") and another action supports the Amazon Resource Name (ARN) of a specific resource, they must be in two separate JSON statements. For details about ARN formats, see [Amazon Resource Name \(ARN\)](#) in the *AWS General Reference Guide*. For general information about IAM policies, see [Policies and permissions in IAM \(p. 351\)](#). For information about the IAM policy language, see [IAM JSON policy reference \(p. 627\)](#).

To use the JSON policy editor to create a 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 on the left, choose **Policies**.
3. Choose **Create policy**.
4. Choose the **JSON** tab.
5. Type or paste a JSON policy document. For details about the IAM policy language, see [IAM JSON policy reference \(p. 627\)](#).
6. When you are finished, choose **Review policy**. The [Policy Validator \(p. 444\)](#) reports any syntax errors.

Note

You can switch between the **Visual editor** and **JSON** tabs anytime. However, if you make changes or choose **Review policy** in the **Visual editor** tab, IAM might restructure your policy to optimize it for the visual editor. For more information, see [Policy restructuring \(p. 571\)](#).

7. On the **Review policy** page, type a **Name** and a **Description** (optional) for the policy that you are creating. Review the policy **Summary** to see the permissions that are granted by your policy. Then choose **Create policy** to save your work.

After you create a policy, you can attach it to your groups, users, or roles. For more information, see [Adding and removing IAM identity permissions \(p. 454\)](#).

Creating policies with the visual editor

The visual editor in the IAM console guides you through creating a policy without having to write JSON syntax. To view an example of using the visual editor to create a policy, see [the section called "Controlling access to identities" \(p. 378\)](#).

To use the visual editor to create a 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 on the left, choose **Policies**.
3. Choose **Create policy**.
4. On the **Visual editor** tab, choose **Choose a service** and then choose an AWS service. You can use the search box at the top to limit the results in the list of services. You can choose only one service within a visual editor permission block. To grant access to more than one service, add multiple permission blocks by choosing **Add additional permissions**.
5. For **Actions**, choose the actions to add to the policy. You can choose actions in the following ways:
 - Select the check box for all actions.
 - Choose **add actions** to type the name of a specific action. You can use wildcards (*) to specify multiple actions.
 - Select one of the **Access level** groups to choose all actions for the access level (for example, **Read**, **Write**, or **List**).
 - Expand each of the **Access level** groups to choose individual actions.

By default, the policy that you are creating allows the actions that you choose. To deny the chosen actions instead, choose **Switch to deny permissions**. Because [IAM denies by default \(p. 666\)](#), we recommend as a security best practice that you allow permissions to only those actions and resources that a user needs. You should create a JSON statement to deny permissions only if you want to override a permission separately allowed by another statement or policy. We recommend that you limit the number of deny permissions to a minimum because they can increase the difficulty of troubleshooting permissions.

6. For **Resources**, if the service and actions that you selected in the previous steps do not support choosing [specific resources \(p. 383\)](#), all resources are allowed and you cannot edit this section.

If you chose one or more actions that support [resource-level permissions \(p. 383\)](#), then the visual editor lists those resources. You can then expand **Resources** to specify resources for your policy.

You can specify resources in the following ways:

- Choose **Add ARN** to specify resources by their Amazon Resource Names (ARN). You can use the visual ARN editor or list ARNs manually. For more information about ARN syntax, see [Amazon Resource Name \(ARN\)](#) in the *AWS General Reference Guide*. For information about using ARNs in the **Resource** element of a policy, see [IAM JSON policy elements: Resource \(p. 639\)](#).
 - Choose **Any** next to a resource to grant permissions to any resources of that type.
 - Choose **All resources** to choose all resources for the service.
7. (Optional) Choose **Specify request conditions (optional)** to add conditions to the policy that you are creating. Conditions limit a JSON policy statement's effect. For example, you can specify that a user is allowed to perform the actions on the resources only when that user's request happens within a certain time range. You can also use commonly used conditions to limit whether a user must be authenticated using a multi-factor authentication (MFA) device. Or you can require that the request originate from within a certain range of IP addresses. For lists of all of the context keys that you can use in a policy condition, see [Actions, Resources, and Condition Keys for AWS Services](#).

You can choose conditions in the following ways:

- Use check boxes to select commonly used conditions.
- Choose **Add condition** to specify other conditions. Choose the condition's **Condition Key**, **Qualifier**, and **Operator**, and then type a **Value**. To add more than one value, choose **Add new value**. You can consider the values as being connected by a logical "OR" operator. When you are finished, choose **Add**.

To add more than one condition, choose **Add condition** again. Repeat as needed. Each condition applies only to this one visual editor permission block. All the conditions must be true for the

permission block to be considered a match. In other words, consider the conditions to be connected by a logical "AND" operator.

For more information about the **Condition** element, see [IAM JSON policy elements: Condition \(p. 641\)](#) in the [IAM JSON policy reference \(p. 627\)](#).

8. To add more permission blocks, choose **Add additional permissions**. For each block, repeat steps 2 through 5.
9. When you are finished, choose **Review policy**.

Note

You can switch between the **Visual editor** and **JSON** tabs anytime. However, if you make changes or choose **Review policy** in the **Visual editor** tab, IAM might restructure your policy to optimize it for the visual editor. For more information, see [Policy restructuring \(p. 571\)](#).

10. On the **Review policy** page, type a **Name** and a **Description** (optional) for the policy that you are creating. Review the policy summary to make sure that you have granted the intended permissions, and then choose **Create policy** to save your new policy.

After you create a policy, you can attach it to your groups, users, or roles. For more information, see [Adding and removing IAM identity permissions \(p. 454\)](#).

Importing existing managed policies

An easy way to create a new policy is to import an existing managed policy within your account that has at least some of the permissions that you need. You can then customize the policy to match it to your new requirements.

You cannot import an inline policy. To learn about the difference between managed and inline policies, see [Managed policies and inline policies \(p. 359\)](#).

To import an existing managed policy in the visual editor

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 on the left, choose **Policies**.
3. Choose **Create policy**.
4. Choose the **Visual editor** tab, and then on the right side of the page, choose **Import managed policy**.
5. In the **Import managed policies** window, choose the managed policies that most closely match the policy that you want to include in your new policy. You can use the **Filter** menu or type in the search box at the top to limit the results in the list of policies.
6. Choose **Import**.

The imported policies are added in new permission blocks at the bottom of your policy.

7. Use the **Visual editor** or choose **JSON** to customize your policy. Then choose **Review policy**.

Note

You can switch between the **Visual editor** and **JSON** tabs anytime. However, if you make changes or choose **Review policy** in the **Visual editor** tab, IAM might restructure your policy to optimize it for the visual editor. For more information, see [Policy restructuring \(p. 571\)](#).

8. On the **Review** page, type a **Name** and a **Description** (optional) for the policy that you are creating. You cannot edit these settings later. Review the policy **Summary** and then choose **Create policy** to save your work.

To import an existing managed policy in the JSON tab

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 on the left, choose **Policies**.
3. Choose **Create policy**.
4. Choose the **JSON** tab, and then on the right side of the page, choose **Import managed policy**.
5. In the **Import managed policies** window, choose the managed policies that most closely match the policy that you want to include in your new policy. You can use the **Filter** menu or type in the search box at the top to limit the results in the list of policies.
6. Choose **Import**.

Statements from the imported policies are added to the bottom of your JSON policy.

7. Customize your policy in JSON, or choose the **Visual editor**. Then choose **Review policy**.

Note

You can switch between the **Visual editor** and **JSON** tabs anytime. However, if you make changes or choose **Review policy** in the **Visual editor** tab, IAM might restructure your policy to optimize it for the visual editor. For more information, see [Policy restructuring \(p. 571\)](#).

8. On the **Review policy** page, type a **Name** and a **Description** (optional) for the policy that you are creating. You cannot edit these later. Review the policy **Summary** and then choose **Create policy** to save your work.

After you create a policy, you can attach it to your groups, users, or roles. For more information, see [Adding and removing IAM identity permissions \(p. 454\)](#).

Creating IAM policies (AWS CLI)

A [policy \(p. 351\)](#) is an entity that, when attached to an identity or resource, defines their permissions. You can use the AWS CLI to create *customer managed policies* in IAM. Customer managed policies are standalone policies that you administer in your own AWS account. You can then attach the policies to identities (users, groups, and roles) in your AWS account.

The number and size of IAM resources in an AWS account are limited. For more information, see [IAM and STS quotas \(p. 606\)](#).

Creating IAM policies (AWS CLI)

You can create an IAM customer managed policy or an inline policy using the AWS Command Line Interface (AWS CLI).

To create a customer managed policy (AWS CLI)

Use the following command:

- [create-policy](#)

To create an inline policy for an IAM identity (group, user or role) (AWS CLI)

Use one of the following commands:

- [put-group-policy](#)
- [put-role-policy](#)

- [put-user-policy](#)

Note

You can't use IAM to embed an inline policy for a [service-linked role \(p. 168\)](#).

Creating IAM policies (AWS API)

A [policy \(p. 351\)](#) is an entity that, when attached to an identity or resource, defines their permissions. You can use the AWS API to create *customer managed policies* in IAM. Customer managed policies are standalone policies that you administer in your own AWS account. You can then attach the policies to identities (users, groups, and roles) in your AWS account.

The number and size of IAM resources in an AWS account are limited. For more information, see [IAM and STS quotas \(p. 606\)](#).

Creating IAM policies (AWS API)

You can create an IAM customer managed policy or an inline policy using the AWS API.

To create a customer managed policy (AWS API)

Call the following operation:

- [CreatePolicy](#)

To create an inline policy for an IAM identity (group, user, or role) (AWS API)

Call one of the following operations:

- [PutGroupPolicy](#)
- [PutRolePolicy](#)
- [PutUserPolicy](#)

Note

You can't use IAM to embed an inline policy for a [service-linked role \(p. 168\)](#).

Validating IAM policy grammar

A [policy](#) is a JSON document that uses the [IAM policy grammar](#). When you attach a policy to an IAM entity, such as a user, group, or role, it grants permissions to that entity.

When you create or edit IAM access control policies using the AWS Management Console, Policy Validator automatically examines them to ensure that they comply with the IAM policy grammar. If Policy Validator determines that a policy is not in compliance with the grammar, it prompts you to fix the policy.

Validation Scope

Policy Validator only checks JSON policy syntax and grammar. It does not verify that your ARNs, action names, or condition keys are correct.

Accessing Policy Validator

Policy Validator runs automatically when you do the following using the IAM console:

- **Create a JSON policy** – When you create a new JSON policy, Policy Validator runs automatically when you choose **Review policy**. If the policy is not valid, you receive a notification and must fix the problem before you can continue.
- **Edit a JSON policy** – When you edit an existing JSON policy, Policy Validator runs automatically when you choose **Review policy**. If the policy is not valid, you receive a notification and must fix the problem before you can continue.

Existing Policies

You might have existing policies that are not valid because they were created or last saved before the introduction of Policy Validator. You cannot edit and save an existing policy without fixing any policy syntax errors.

Testing IAM policies with the IAM policy simulator

For more information about how and why to use IAM policies, see [Policies and permissions in IAM \(p. 351\)](#).

You can access the IAM Policy Simulator Console at: <https://policysim.aws.amazon.com/>

Getting Started with the IAM Policy Simulator

With the IAM policy simulator, you can test and troubleshoot identity-based policies, IAM permissions boundaries, Organizations service control policies (SCPs), and resource-based policies. Here are some common things you can do with the policy simulator:

- Test policies that are attached to IAM users, groups, or roles in your AWS account. If more than one policy is attached to the user, group, or role, you can test all the policies, or select individual policies to test. You can test which actions are allowed or denied by the selected policies for specific resources.
- Test and troubleshoot the effect of [permissions boundaries \(p. 365\)](#) on IAM entities. Note: you can only simulate one permissions boundary at a time.
- Test policies that are attached to AWS resources, such as Amazon S3 buckets, Amazon SQS queues, Amazon SNS topics, or Amazon S3 Glacier vaults.
- If your AWS account is a member of an organization in [AWS Organizations](#), then you can test the impact of service control policies (SCPs) on your IAM policies and resource policies.
- Test new policies that are not yet attached to a user, group, or role by typing or copying them into the simulator. These are used only in the simulation and are not saved. Note: you cannot type or copy a resource-based policy into the simulator. To use a resource-based policy in the simulator, you must include the resource in the simulation. You must also select the check box to include that resource's policy in the simulation.
- Test the policies with selected services, actions, and resources. For example, you can test to ensure that your policy allows an entity to perform the `ListAllMyBuckets`, `CreateBucket`, and `DeleteBucket` actions in the Amazon S3 service on a specific bucket.
- Simulate real-world scenarios by providing context keys, such as an IP address or date, that are included in Condition elements in the policies being tested.
- Identify which specific statement in a policy results in allowing or denying access to a particular resource or action.

Topics

- [How the IAM policy simulator works \(p. 446\)](#)
- [Permissions required for using the IAM policy simulator \(p. 446\)](#)
- [Using the IAM policy simulator \(console\) \(p. 448\)](#)

- [Using the IAM policy simulator \(AWS CLI and AWS API\) \(p. 453\)](#)

How the IAM policy simulator works

The simulator evaluates the policies that you choose and determines the effective permissions for each of the actions that you specify. The simulator uses the same policy evaluation engine that is used during real requests to AWS services. But the simulator differs from the live AWS environment in the following ways:

- The simulator does not make an actual AWS service request, so you can safely test requests that might make unwanted changes to your live AWS environment.
- Because the simulator does not simulate running the selected actions, it cannot report any response to the simulated request. The only result returned is whether the requested action would be allowed or denied.
- If you edit a policy inside the simulator, these changes affect only the simulator. The corresponding policy in your AWS account remains unchanged.
- You can't test AWS Organizations service control policies (SCPs) with [global condition keys \(p. 692\)](#).

Permissions required for using the IAM policy simulator

You can use the policy simulator console or the policy simulator API to test policies. By default, console users can test policies that are not yet attached to a user, group, or role by typing or copying those policies into the simulator. These policies are used only in the simulation and do not disclose sensitive information. API users must have permissions to test unattached policies. You can allow console or API users to test policies that are attached to IAM users, groups, or roles in your AWS account. To do so, you must provide permission to retrieve those policies. In order to test resource-based policies, users must have permission to retrieve the resource's policy.

For examples of console and API policies that allow a user to simulate policies, see the section called ["Example policies: AWS Identity and Access Management \(IAM\)" \(p. 391\)](#).

Permissions required for using the policy simulator console

You can allow users to test policies that are attached to IAM users, groups, or roles in your AWS account. To do so, you must provide your users with permissions to retrieve those policies. In order to test resource-based policies, users must have permission to retrieve the resource's policy.

To view an example policy that allows using the policy simulator console for policies that are attached to a user, group, or role, see [IAM: Access the policy simulator console \(p. 415\)](#).

To view an example policy that allows using the policy simulator console only for those users with a specific path, see [IAM: Access the policy simulator console based on user path \(p. 425\)](#).

To create a policy to allow using the policy simulator console for only one type of entity, use the following procedures.

To allow console users to simulate policies for users

Include the following actions in your policy:

- `iam:GetGroupPolicy`
- `iam:GetPolicy`
- `iam:GetPolicyVersion`
- `iam:GetUser`

- iam:GetUserPolicy
- iam>ListAttachedUserPolicies
- iam>ListGroupsForUser
- iam>ListGroupPolicies
- iam>ListUserPolicies
- iam>ListUsers

To allow console users to simulate policies for groups

Include the following actions in your policy:

- iam:GetGroup
- iam:GetGroupPolicy
- iam:GetPolicy
- iam:GetPolicyVersion
- iam>ListAttachedGroupPolicies
- iam>ListGroupPolicies
- iam>ListGroups

To allow console users to simulate policies for roles

Include the following actions in your policy:

- iam:GetPolicy
- iam:GetPolicyVersion
- iam:GetRole
- iam:GetRolePolicy
- iam>ListAttachedRolePolicies
- iam>ListRolePolicies
- iam>ListRoles

To test resource-based policies, users must have permission to retrieve the resource's policy.

To allow console users to test resource-based policies in an Amazon S3 bucket

Include the following action in your policy:

- s3:GetBucketPolicy

For example, the following policy uses this action to allow console users to simulate a resource-based policy in a specific Amazon S3 bucket.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "s3:GetBucketPolicy",  
            "Resource": "arn:aws:s3:::bucket-name/*"  
        }  
    ]  
}
```

```
    ]  
}
```

Permissions required for using the policy simulator API

The policy simulator API operations [GetContextKeyForCustomPolicy](#) and [SimulateCustomPolicy](#) allow you to test policies that are not yet attached to a user, group, or role. To test such policies, you pass the policies as strings to the API. These policies are used only in the simulation and do not disclose sensitive information. You can also use the API to test policies that are attached to IAM users, groups, or roles in your AWS account. To do that, you must provide users with permissions to call [GetContextKeyForPrincipalPolicy](#) and [SimulatePrincipalPolicy](#).

To view an example policy that allows using the policy simulator API for attached and unattached policies in the current AWS account, see [IAM: Access the policy simulator API \(p. 414\)](#).

To create a policy to allow using the policy simulator API for only one type of policy, use the following procedures.

To allow API users to simulate policies passed directly to the API as strings

Include the following actions in your policy:

- `iam:GetContextKeysForCustomPolicy`
- `iam:SimulateCustomPolicy`

To allow API users to simulate policies attached to IAM users, groups, roles, or resources

Include the following actions in your policy:

- `iam:GetContextKeysForPrincipalPolicy`
- `iam:SimulatePrincipalPolicy`

For example, to give a user named Bob permission to simulate a policy that is assigned to a user named Alice, give Bob access to the following resource: `arn:aws:iam::777788899999:user/alice`.

To view an example policy that allows using the policy simulator API only for those users with a specific path, see [IAM: Access the policy simulator API based on user path \(p. 425\)](#).

Using the IAM policy simulator (console)

By default, users can test policies that are not yet attached to a user, group, or role by typing or copying those policies into the policy simulator console. These policies are used only in the simulation and do not disclose sensitive information.

To test a policy that is not attached to a user, group, or role (console)

1. Open the IAM policy simulator console at: <https://policysim.aws.amazon.com/>.
2. In the **Mode**: menu at the top of the page, choose **New Policy**.
3. In the **Policy Sandbox**, choose **Create New Policy**.
4. Type or copy a policy into the simulator, and use the simulator as described in the following steps.

After you have permission to use the IAM Policy Simulator Console, you can use the simulator to test an IAM user, group, role, or resource policy.

To test a policy that is attached to a user, group, or role (console)

1. Open the IAM policy simulator console at <https://policysim.aws.amazon.com/>.

Note

To sign in to the policy simulator as an IAM user, use your unique sign-in URL to sign in to the AWS Management Console. Then go to <https://policysim.aws.amazon.com/>. For more information about signing in as an IAM user, see [How IAM users sign in to AWS \(p. 81\)](#).

The simulator opens in **Existing Policies** mode and lists the IAM users in your account under **Users, Groups, and Roles**.

2. Choose the option that is appropriate to your task:

To test this:	Do this:
A policy attached to a user	Choose Users in the Users, Groups, and Roles list. Then choose the user.
A policy attached to a group	Choose Groups in the Users, Groups, and Roles list. Then choose the group.
A policy attached to a role	Choose Roles in the Users, Groups, and Roles list. Then choose the role.
A policy attached to a resource	See Step 9 .
A custom policy for a user, group, or role	Choose Create New Policy . In the new Policies pane, type or paste a policy and then choose Apply .

Tip

To test a policy that is attached to group, you can launch the IAM policy simulator directly from the [IAM console](#): In the navigation pane, choose **Groups**. Choose the name of the group that you want to test a policy on, and then choose the **Permissions** tab. In the **Inline Policies or Managed Policies** section, locate the policy that you want to test. In the **Actions** column for that policy, choose **Simulate Policy**.

To test a customer managed policy that is attached to a user: In the navigation pane, choose **Users**. Choose the name of the user that you want to test a policy on. Then choose the **Permissions** tab and expand the policy that you want to test. On the far right, choose **Simulate policy**. The **IAM Policy Simulator** opens in a new window and displays the selected policy in the **Policies** pane.

3. (Optional) If your account is a member of an organization in [AWS Organizations](#), then select the check box next to **AWS Organizations SCPs** to include SCPs in your simulated evaluation. SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU). The SCP limits permissions for entities in member accounts. If an SCP blocks a service or action, then no entity in that account can access that service nor perform that action. This is true even if an administrator explicitly grants permissions to that service or action through an IAM or resource policy.

If your account is not a member of an organization, then the check box does not appear.

4. (Optional) You can test a policy that is set as a [permissions boundary \(p. 365\)](#) for an IAM entity (user or role), but not for groups. If a permissions boundary policy is currently set for the entity, it appears in the **Policies** pane. You can set only one permissions boundary for an entity. To test a different permissions boundary, you can create a custom permissions boundary. To do this, choose **Create New Policy**. A new **Policies** pane opens. In the menu, choose **Custom IAM Permissions Boundary Policy**. Enter a name for the new policy and type or copy a policy into the space below.

- Choose **Apply** to save the policy. Next, choose **Back** to return to the original **Policies** pane. Then select the check box next to the permissions boundary you want to use for the simulation.
5. (Optional) You can test only a subset of policies attached to a user, group, or role. To do so, in the **Policies** pane clear the check box next to each policy that you want to exclude.
 6. Under **Policy Simulator**, choose **Select service** and then choose the service to test. Then choose **Select actions** and select one or more actions to test. Although the menus show the available selections for only one service at a time, all the services and actions that you have selected appear in **Action Settings and Results**.
 7. (Optional) If any of the policies that you choose in [Step 2](#) and [Step 5](#) include conditions with [AWS global condition keys \(p. 692\)](#), then supply values for those keys. You can do this by expanding the **Global Settings** section and typing values for the key names displayed there.

Warning

If you leave the value for a condition key empty, then that key is ignored during the simulation. In some cases, this results in an error, and the simulation fails to run. In other cases, the simulation runs, but the results might not be reliable. In those cases, the simulation does not match the real-world conditions that include a value for the condition key or variable.

8. (Optional) Each selected action appears in the **Action Settings and Results** list with **Not simulated** shown in the **Permission** column until you actually run the simulation. Before you run the simulation, you can configure each action with a resource. To configure individual actions for a specific scenario, choose the arrow to expand the action's row. If the action supports resource-level permissions, you can type the [Amazon Resource Name \(ARN\) \(p. 601\)](#) of the specific resource whose access you want to test. By default, each resource is set to a wildcard (*). You can also specify a value for any [condition context keys](#). As noted previously, keys with empty values are ignored, which can cause simulation failures or unreliable results.
 - a. Choose the arrow next to the action name to expand each row and configure any additional information required to accurately simulate the action in your scenario. If the action requires any resource-level permissions, you can type the [Amazon Resource Name \(ARN\) \(p. 601\)](#) of the specific resource that you want to simulate access to. By default, each resource is set to a wildcard (*).
 - b. If the action supports resource-level permissions but does not require them, then you can choose **Add Resource** to select the resource type that you want to add to the simulation.
 - c. If any of the selected policies include a **Condition** element that references a context key for this action's service, then that key name is displayed under the action. You can specify the value to be used during the simulation of that action for the specified resource.

Actions that require different groups of resource types

Some actions require different resource types under different circumstances. Each group of resource types is associated with a scenario. If one of these applies to your simulation, select it and the simulator requires the resource types appropriate for that scenario. The following list shows each of the supported scenario options and the resources that you must define to run the simulation.

Each of the following Amazon EC2 scenarios requires that you specify `instance`, `image`, and `security-group` resources. If your scenario includes an EBS volume, then you must specify that volume as a resource. If the Amazon EC2 scenario includes a virtual private cloud (VPC), then you must supply the `network-interface` resource. If it includes an IP subnet, then you must specify the `subnet` resource. For more information on the Amazon EC2 scenario options, see [Supported Platforms](#) in the [Amazon EC2 User Guide](#).

- **EC2-Classic-InstanceStore**

`instance`, `image`, `security-group`

- **EC2-Classic-EBS**

instance, image, security-group, volume

- **EC2-VPC-InstanceStore**

instance, image, security-group, network-interface

- **EC2-VPC-InstanceStore-Subnet**

instance, image, security-group, network-interface, subnet

- **EC2-VPC-EBS**

instance, image, security-group, network-interface, volume

- **EC2-VPC-EBS-Subnet**

instance, image, security-group, network-interface, subnet, volume

9. (Optional) If you want to include a resource-based policy in your simulation, then you must first select the actions that you want to simulate on that resource in [Step 6](#). Expand the rows for the selected actions, and type the ARN of the resource with a policy that you want to simulate. Then select **Include Resource Policy** next to the **ARN** text box. The IAM policy simulator currently supports resource-based policies from only the following services: Amazon S3 (resource-based policies only; ACLs are not currently supported), Amazon SQS, Amazon SNS, and unlocked S3 Glacier vaults (locked vaults are not currently supported).

10. Choose **Run Simulation** in the upper-right corner.

The **Permission** column in each row of **Action Settings and Results** displays the result of the simulation of that action on the specified resource.

11. Choose **Run Simulation** in the upper-right corner.

The **Permission** column in each row of **Action Settings and Results** displays the result of the simulation of that action on the specified resource.

12. To see which statement in a policy explicitly allowed or denied an action, choose the [**N matching statement\(s\)**](#) link in the **Permissions** column to expand the row. Then choose the [**Show statement**](#) link. The **Policies** pane shows the relevant policy with the statement that affected the simulation result highlighted.

Note

If an action is *implicitly* denied—that is, if the action is denied only because it is not explicitly allowed—the [**List**](#) and [**Show statement**](#) options are not displayed.

Troubleshooting IAM policy simulator console messages

The following table lists the informational and warning messages you might encounter when using the IAM policy simulator. The table also provides steps you can take to resolve them.

Message	Steps to resolve
This policy has been edited. Changes will not be saved to your account.	<p>No action required.</p> <p>This message is informational. If you edit an existing policy in the IAM policy simulator, your change does not affect your AWS account. The simulator allows you to make changes to policies for testing purposes only.</p>
Cannot get the resource policy. Reason: detailed error message	<p>The simulator is not able to access a requested resource-based policy. Ensure that the specified resource ARN is correct and that the user running</p>

Message	Steps to resolve
	the simulation has permission to read the resource's policy.
One or more policies require values in the simulation settings. The simulation might fail without these values.	<p>This message appears if the policy you are testing contains condition keys or variables but you have not provided any values for these keys or variables in Simulation Settings.</p> <p>To dismiss this message, choose Simulation Settings, Then enter a value for each condition key or variable.</p>
You have changed policies. These results are no longer valid.	<p>This message appears if you have changed the selected policy while results are displayed in the Results pane. Results shown in the Results pane are not updated dynamically.</p> <p>To dismiss this message, choose Run Simulation again to display new simulation results based on the changes made in the Policies pane.</p>
The resource you typed for this simulation does not match this service.	<p>This message appears if you have typed an Amazon Resource Name (ARN) in the Simulation Settings pane that does not match the service that you chose for the current simulation. For example, this message appears if you specify an ARN for an Amazon DynamoDB resource but you chose Amazon Redshift as the service to simulate.</p> <p>To dismiss this message, do one of the following:</p> <ul style="list-style-type: none"> Remove the ARN from the box in the Simulation Settings pane. Choose the service that matches the ARN that you specified in Simulation Settings.
This action belongs to a service that supports special access control mechanisms in addition to resource-based policies, such as Amazon S3 ACLs or S3 Glacier vault lock policies. The policy simulator does not support these mechanisms, so the results can differ from your production environment.	<p>No action required.</p> <p>This message is informational. In the current version, the simulator evaluates policies attached to users and groups, and can evaluate resource-based policies for Amazon S3, Amazon SQS, Amazon SNS, and S3 Glacier. The policy simulator does not support all access control mechanisms supported by other AWS services.</p>

Message	Steps to resolve
DynamoDB FGAC is currently not supported.	<p>No action required.</p> <p>This informational message refers to <i>fine-grained access control</i>. Fine-grained access control is the ability to use IAM policy conditions to determine who can access individual data items and attributes in DynamoDB tables and indexes. It also refers to the actions that can be performed on these tables and indexes. The current version of the IAM policy simulator does not support this type of policy condition. For more information on DynamoDB fine-grained access control, see Fine-Grained Access Control for DynamoDB.</p>
You have policies that do not comply with the policy syntax. You can use the Policy Validator to review and accept the recommended updates to your policies.	<p>This message appears at the top of the policy list if you have policies that do not comply with the IAM policy grammar. In order to simulate these policies, follow the instructions at Validating IAM policy grammar (p. 444) to identify and fix these policies.</p>
This policy must be updated to comply with the latest policy syntax rules.	<p>This message is displayed if you have policies that do not comply with the IAM policy grammar. In order to simulate these policies, follow the instructions at Validating IAM policy grammar (p. 444) to identify and fix these policies.</p>

Using the IAM policy simulator (AWS CLI and AWS API)

Policy simulator commands typically require calling API operations to do two things:

1. Evaluate the policies and return the list of context keys that they reference. You need to know what context keys are referenced so that you can supply values for them in the next step.
2. Simulate the policies, providing a list of actions, resources, and context keys that are used during the simulation.

For security reasons, the API operations have been broken into two groups:

- API operations that simulate only policies that are passed directly to the API as strings. This set includes [GetContextKeysForCustomPolicy](#) and [SimulateCustomPolicy](#).
- API operations that simulate the policies that are attached to a specified IAM user, group, role, or resource. Because these API operations can reveal details of permissions assigned to other IAM entities, you should consider restricting access to these API operations. This set includes [GetContextKeysForPrincipalPolicy](#) and [SimulatePrincipalPolicy](#). For more information about restricting access to API operations, see [Example policies: AWS Identity and Access Management \(IAM\) \(p. 391\)](#).

In both cases, the API operations simulate the effect of one or more policies on a list of actions and resources. Each action is paired with each resource and the simulation determines whether the policies allow or deny that action for that resource. You can also provide values for any context keys that your policies reference. You can get the list of context keys that the policies reference by first calling [GetContextKeysForCustomPolicy](#) or [GetContextKeysForPrincipalPolicy](#). If you don't provide

a value for a context key, the simulation still runs. But the results might not be reliable because the simulator cannot include that context key in the evaluation.

To get the list of context keys (AWS CLI, AWS API)

Use the following to evaluate a list of policies and return a list of context keys that are used in the policies.

- AWS CLI: `aws iam get-context-keys-for-custom-policy` and `aws iam get-context-keys-for-principal-policy`
- AWS API: `GetContextKeysForCustomPolicy` and `GetContextKeysForPrincipalPolicy`

To simulate IAM policies (AWS CLI, AWS API)

Use the following to simulate IAM policies to determine a user's effective permissions.

- AWS CLI: `aws iam simulate-custom-policy` and `aws iam simulate-principal-policy`
- AWS API: `SimulateCustomPolicy` and `SimulatePrincipalPolicy`

Adding and removing IAM identity permissions

You use policies to define the permissions for an identity (user, group, or role). You can add and remove permissions by attaching and detaching IAM policies for an identity using the AWS Management Console, the AWS Command Line Interface (AWS CLI), or the AWS API. You can also use policies to set [permissions boundaries \(p. 365\)](#) for only entities (users or roles) using the same methods. Permissions boundaries are an advanced AWS feature that control the maximum permissions that an entity can have.

Topics

- [Terminology \(p. 454\)](#)
- [View identity activity \(p. 455\)](#)
- [Adding IAM identity permissions \(console\) \(p. 455\)](#)
- [Removing IAM identity permissions \(console\) \(p. 457\)](#)
- [Adding IAM policies \(AWS CLI\) \(p. 458\)](#)
- [Removing IAM policies \(AWS CLI\) \(p. 458\)](#)
- [Adding IAM policies \(AWS API\) \(p. 460\)](#)
- [Removing IAM policies \(AWS API\) \(p. 461\)](#)

Terminology

When you associate permissions policies with identities (users, groups, and roles), terminology and procedures vary depending on whether you are working with a managed or inline policy:

- **Attach** – Used with managed policies. You attach a managed policy to an identity (a user, group, or role). Attaching a policy applies the permissions in the policy to the identity.
- **Detach** – Used with managed policies. You detach a managed policy from an IAM identity (a user, group, or role). Detaching a policy removes its permissions from the identity.
- **Embed** – Used with inline policies. You embed an inline policy in an identity (a user, group, or role). Embedding a policy applies the permissions in the policy to the identity. Because an inline policy is stored in the identity, it is embedded rather than attached, though the results are similar.

Note

You can embed an inline policy for a [service-linked role \(p. 168\)](#) only in the service that depends on the role. See the [AWS documentation](#) for your service to see whether it supports this feature.

- **Delete** – Used with inline policies. You delete an inline policy from an IAM identity (a user, group, or role). Deleting a policy removes its permissions from the identity.

Note

You can delete an inline policy for a [service-linked role \(p. 168\)](#) only in the service that depends on the role. See the [AWS documentation](#) for your service to see whether it supports this feature.

You can use the console, AWS CLI, or AWS API to perform any of these actions.

More information

- For more information about the difference between managed and inline policies, see [Managed policies and inline policies \(p. 359\)](#).
- For more information about permissions boundaries, see [Permissions boundaries for IAM entities \(p. 365\)](#).
- For general information about IAM policies, see [Policies and permissions in IAM \(p. 351\)](#).
- The number and size of IAM resources in an AWS account are limited. For more information, see [IAM and STS quotas \(p. 606\)](#).

View identity activity

Before you change the permissions for an identity (user, group, or role), you should review their recent service-level activity. This is important because you don't want to remove access from a principal (person or application) who is using it. For more information about viewing last accessed information, see [Refining permissions in AWS using last accessed information \(p. 472\)](#).

Adding IAM identity permissions (console)

You can use the AWS Management Console to add permissions to an identity (user, group, or role). To do this, attach managed policies that control permissions, or specify a policy that serves as a [permissions boundary \(p. 365\)](#). You can also embed an inline policy.

To use a managed policy as a permissions policy for an identity (console)

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, choose **Policies**.
3. In the list of policies, select the check box next to the name of the policy to attach. You can use the **Filter** menu and the search box to filter the list of policies.
4. Choose **Policy actions**, and then choose **Attach**.
5. Select one or more identities to attach the policy to. You can use the **Filter** menu and the search box to filter the list of principal entities. After selecting the identities, choose **Attach policy**.

To use a managed policy to set a permissions boundary (console)

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, choose **Policies**.
3. In the list of policies, choose the name of the policy to set. You can use the **Filter** menu and the search box to filter the list of policies.
4. On the policy summary page, choose the **Policy usage tab**, and then, if necessary, open the **Permissions boundaries** section and choose **Set boundary**.
5. Select one or more users or roles on which to use the policy for a permissions boundary. You can use the **Filter** menu and the search box to filter the list of principal entities. After selecting the principals, choose **Set boundaries**.

To embed an inline policy for a user or role (console)

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, choose **Users or Roles**.
3. In the list, choose the name of the user or role to embed a policy in.
4. Choose the **Permissions** tab.
5. Scroll to the bottom of the page and choose **Add inline policy**.

Note

You cannot embed an inline policy in a [service-linked role \(p. 168\)](#) in IAM. Because the linked service defines whether you can modify the permissions of the role, you might be able to add additional policies from the service console, API, or AWS CLI. To view the service-linked role documentation for a service, see [AWS services that work with IAM \(p. 611\)](#) and choose **Yes** in the **Service-Linked Role** column for your service.

6. Choose from the following methods to view the steps required to create your policy:
 - [Importing existing managed policies \(p. 442\)](#) – You can import a managed policy within your account and then edit the policy to customize it to your specific requirements. A managed policy can be an AWS managed policy or a customer managed policy that you created previously.
 - [Creating policies with the visual editor \(p. 440\)](#) – You can construct a new policy from scratch in the visual editor. If you use the visual editor, you do not have to understand JSON syntax.
 - [Creating policies on the JSON tab \(p. 440\)](#) – In the **JSON** tab, you can use JSON syntax to create a policy. You can type a new JSON policy document or paste an [example policy \(p. 389\)](#).
7. After you create an inline policy, it is automatically embedded in your user or role.

To embed an inline policy for a group (console)

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, choose **Groups**.
3. In the list, choose the name of the group to embed a policy in.
4. Choose the **Permissions** tab and expand the **Inline Policies** section if necessary.
5. Choose **Create Group Policy**. If there are no existing policies in **Groups**, instead choose [click here](#) to create your first inline policy.
6. Choose **Policy Generator** or **Custom Policy**, and then choose **Select**.
7. Do one of the following:
 - If you chose **Custom Policy**, specify a name for the policy and create your policy document. [Policy Validator \(p. 444\)](#) reports any syntax errors.

- If you are using the policy generator to create your policy, choose the appropriate **Effect**, **AWS Service**, and **Actions** options. Type the Amazon Resource Name (ARN) (if applicable), and add any conditions that you want to include. Then choose **Add Statement**. You can add as many statements as you want to the policy. When you are finished adding statements, choose **Next Step**.
8. When you are satisfied with the policy, choose **Apply Policy**.

To change the permissions boundary for one or more entities (console)

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, choose **Policies**.
3. In the list of policies, choose the name of the policy to set. You can use the **Filter** menu and the search box to filter the list of policies.
4. On the policy summary page, choose the **Policy usage tab**, and then, if necessary, open the **Permissions boundaries** section. Select the check box next to the users or roles whose boundaries you want to change and then choose **Change boundary**.
5. Select a new policy to use for a permissions boundary. You can use the **Filter** menu and the search box to filter the list of policies. After selecting the policy, choose **Change boundary**.

Removing IAM identity permissions (console)

You can use the AWS Management Console to remove permissions from an identity (user, group, or role). To do this, detach managed policies that control permissions, or remove a policy that serves as a [permissions boundary](#) (p. 365). You can also delete an inline policy.

To detach a managed policy used as a permissions policy (console)

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, choose **Policies**.
3. In the list of policies, select the check box next to the name of the policy to detach. You can use the **Filter** menu and the search box to filter the list of policies.
4. Choose **Policy actions**, and then choose **Detach**.
5. Select the identities to detach the policy from. You can use the **Filter** menu and the search box to filter the list of identities. After selecting the identities, choose **Detach policy**.

To remove a permissions boundary (console)

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, choose **Policies**.
3. In the list of policies, choose the name of the policy to set. You can use the **Filter** menu and the search box to filter the list of policies.
4. On the policy summary page, choose the **Policy usage tab**, and then, if necessary, open the **Permissions boundaries** section and choose **Remove boundary**.
5. Confirm that you want to remove the boundary and choose **Remove**.

To delete an inline policy (console)

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, choose **Groups**, **Users**, or **Roles**.
3. In the list, choose the name of the group, user, or role that has the policy you want to remove.
4. Choose the **Permissions** tab. If you chose **Groups**, expand the **Inline Policies** section if necessary.
5. If in **Groups**, choose **Remove Policy**. If in **Users** or **Roles**, choose X.

Adding IAM policies (AWS CLI)

You can use the AWS CLI to add permissions to an identity (user, group, or role). To do this, attach managed policies that control permissions, or specify a policy that serves as a [permissions boundary \(p. 365\)](#). You can also embed an inline policy.

To use a managed policy as a permissions policy for an entity (AWS CLI)

1. (Optional) To view information about a managed policy, run the following commands:
 - To list managed policies: `aws iam list-policies`
 - To retrieve detailed information about a managed policy: `get-policy`
2. To attach a managed policy to an identity (user, group, or role), use one of the following commands:
 - `aws iam attach-user-policy`
 - `aws iam attach-group-policy`
 - `aws iam attach-role-policy`

To use a managed policy to set a permissions boundary (AWS CLI)

1. (Optional) To view information about a managed policy, run the following commands:
 - To list managed policies: `aws iam list-policies`
 - To retrieve detailed information about a managed policy: `aws iam get-policy`
2. To use a managed policy to set the permissions boundary for an entity (user or role), use one of the following commands:
 - `aws iam put-user-permissions-boundary`
 - `aws iam put-role-permissions-boundary`

To embed an inline policy (AWS CLI)

To embed an inline policy to an identity (user, group, or role that is not a [service-linked role \(p. 168\)](#)), use one of the following commands:

- `aws iam put-user-policy`
- `aws iam put-group-policy`
- `aws iam put-role-policy`

Removing IAM policies (AWS CLI)

You can use the AWS CLI to detach managed policies that control permissions, or remove a policy that serves as a [permissions boundary \(p. 365\)](#). You can also delete an inline policy.

To detach a managed policy used as a permissions policy (AWS CLI)

1. (Optional) To view information about a policy, run the following commands:
 - To list managed policies: [aws iam list-policies](#)
 - To retrieve detailed information about a managed policy: [aws iam get-policy](#)
2. (Optional) To find out about the relationships between the policies and identities, run the following commands:
 - To list the identities (users, groups, and roles) to which a managed policy is attached:
 - [aws iam list-entities-for-policy](#)
 - To list the managed policies attached to an identity (a user, group, or role), use one of the following commands:
 - [aws iam list-attached-user-policies](#)
 - [aws iam list-attached-group-policies](#)
 - [aws iam list-attached-role-policies](#)
3. To detach a managed policy from an identity (user, group, or role), use one of the following commands:
 - [aws iam detach-user-policy](#)
 - [aws iam detach-group-policy](#)
 - [aws iam detach-role-policy](#)

To remove a permissions boundary (AWS CLI)

1. (Optional) To view which managed policy is currently used to set the permissions boundary for a user or role, run the following commands:
 - [aws iam get-user](#)
 - [aws iam get-role](#)
2. (Optional) To view the users or roles on which a managed policy is used for a permissions boundary, run the following command:
 - [aws iam list-entities-for-policy](#)
3. (Optional) To view information about a managed policy, run the following commands:
 - To list managed policies: [aws iam list-policies](#)
 - To retrieve detailed information about a managed policy: [aws iam get-policy](#)
4. To remove a permissions boundary from a user or role, use one of the following commands:
 - [aws iam delete-user-permissions-boundary](#)
 - [aws iam delete-role-permissions-boundary](#)

To delete an inline policy (AWS CLI)

1. (Optional) To list all inline policies that are attached to an identity (user, group, role), use one of the following commands:
 - [aws iam list-user-policies](#)
 - [aws iam list-group-policies](#)
 - [aws iam list-role-policies](#)

2. (Optional) To retrieve an inline policy document that is embedded in an identity (user, group, or role), use one of the following commands:
 - [aws iam get-user-policy](#)
 - [aws iam get-group-policy](#)
 - [aws iam get-role-policy](#)
3. To delete an inline policy from an identity (user, group, or role that is not a *service-linked role* (p. 168)), use one of the following commands:
 - [aws iam delete-user-policy](#)
 - [aws iam delete-group-policy](#)
 - [aws iam delete-role-policy](#)

Adding IAM policies (AWS API)

You can use the AWS API to attach managed policies that control permissions or specify a policy that serves as a [permissions boundary](#) (p. 365). You can also embed an inline policy.

To use a managed policy as a permissions policy for an entity (AWS API)

1. (Optional) To view information about a policy, call the following operations:
 - To list managed policies: [ListPolicies](#)
 - To retrieve detailed information about a managed policy: [GetPolicy](#)
2. To attach a managed policy to an identity (user, group, or role), call one of the following operations:
 - [AttachUserPolicy](#)
 - [AttachGroupPolicy](#)
 - [AttachRolePolicy](#)

To use a managed policy to set a permissions boundary (AWS API)

1. (Optional) To view information about a managed policy, call the following operations:
 - To list managed policies: [ListPolicies](#)
 - To retrieve detailed information about a managed policy: [GetPolicy](#)
2. To use a managed policy to set the permissions boundary for an entity (user or role), call one of the following operations:
 - [PutUserPermissionsBoundary](#)
 - [PutRolePermissionsBoundary](#)

To embed an inline policy (AWS API)

To embed an inline policy in an identity (user, group, or role that is not a *service-linked role* (p. 168)), call one of the following operations:

- [PutUserPolicy](#)
- [PutGroupPolicy](#)
- [PutRolePolicy](#)

Removing IAM policies (AWS API)

You can use the AWS API to detach managed policies that control permissions or remove a policy that serves as a [permissions boundary \(p. 365\)](#). You can also delete an inline policy.

To detach a managed policy used as a permissions policy (AWS API)

1. (Optional) To view information about a policy, call the following operations:
 - To list managed policies: [ListPolicies](#)
 - To retrieve detailed information about a managed policy: [GetPolicy](#)
2. (Optional) To find out about the relationships between the policies and identities, call the following operations:
 - To list the identities (users, groups, and roles) to which a managed policy is attached:
 - [ListEntitiesForPolicy](#)
 - To list the managed policies attached to an identity (a user, group, or role), call one of the following operations:
 - [ListAttachedUserPolicies](#)
 - [ListAttachedGroupPolicies](#)
 - [ListAttachedRolePolicies](#)
3. To detach a managed policy from an identity (user, group, or role), call one of the following operations:
 - [DetachUserPolicy](#)
 - [DetachGroupPolicy](#)
 - [DetachRolePolicy](#)

To remove a permissions boundary (AWS API)

1. (Optional) To view which managed policy is currently used to set the permissions boundary for a user or role, call the following operations:
 - [GetUser](#)
 - [GetRole](#)
2. (Optional) To view the users or roles on which a managed policy is used for a permissions boundary, call the following operation:
 - [ListEntitiesForPolicy](#)
3. (Optional) To view information about a managed policy, call the following operations:
 - To list managed policies: [ListPolicies](#)
 - To retrieve detailed information about a managed policy: [GetPolicy](#)
4. To remove a permissions boundary from a user or role, call one of the following operations:
 - [DeleteUserPermissionsBoundary](#)
 - [DeleteRolePermissionsBoundary](#)

To delete an inline policy (AWS API)

1. (Optional) To list all inline policies that are attached to an identity (user, group, role), call one of the following operations:

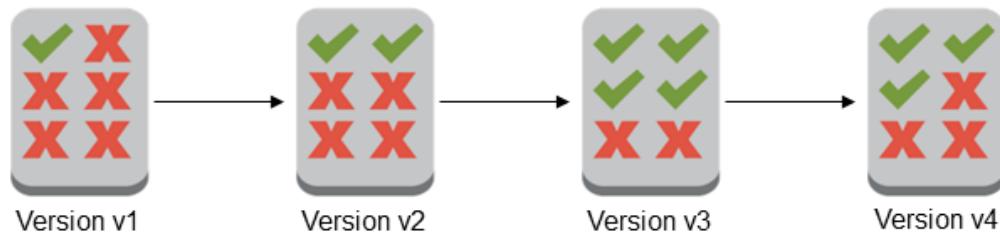
- [ListUserPolicies](#)
 - [ListGroupPolicies](#)
 - [ListRolePolicies](#)
2. (Optional) To retrieve an inline policy document that is embedded in an identity (user, group, or role), call one of the following operations:
 - [GetUserPolicy](#)
 - [GetGroupPolicy](#)
 - [GetRolePolicy](#)
 3. To delete an inline policy from an identity (user, group, or role that is not a *service-linked role* (p. 168)), call one of the following operations:
 - [DeleteUserPolicy](#)
 - [DeleteGroupPolicy](#)
 - [DeleteRolePolicy](#)

Versioning IAM policies

When you make changes to an IAM customer managed policy, and when AWS makes changes to an AWS managed policy, the changed policy doesn't overwrite the existing policy. Instead, IAM creates a new *version* of the managed policy. IAM stores up to five versions of your customer managed policies. IAM does not support versioning for inline policies.

The following diagram illustrates versioning for a customer managed policy. In this example, the versions are 1-4 are saved. You can have up to five managed policy versions saved to IAM. When you edit a policy that would create a sixth saved version, you can choose which older version should no longer be saved. You can revert to any of the other four saved versions at any time.

Multiple versions of a single managed policy



A policy version is different from a `Version` policy element. The `Version` policy element is used within a policy and defines the version of the policy language. To learn more about the `Version` policy element see [IAM JSON policy elements: Version \(p. 629\)](#).

You can use versions to track changes to a managed policy. For example, you might make a change to a managed policy and then discover that the change had unintended effects. In this case, you can roll back to a previous version of the managed policy by setting the previous version as the *default* version.

The following sections explain how you can use versioning for managed policies.

Topics

- [Permissions for setting the default version of a policy \(p. 463\)](#)
- [Setting the default version of customer managed policies \(p. 463\)](#)
- [Using versions to roll back changes \(p. 465\)](#)

- [Version limits \(p. 465\)](#)

Permissions for setting the default version of a policy

The permissions that are required to set the default version of a policy correspond to the AWS API operations for the task. You can use the `CreatePolicyVersion` or `SetDefaultPolicyVersion` API operations to set the default version of a policy. To allow someone to set the default policy version of an existing policy, you can allow access to either the `iam:CreatePolicyVersion` action or the `iam:SetDefaultPolicyVersion` action. The `iam:CreatePolicyVersion` action allows them to create a new version of the policy and to set that version as the default. The `iam:SetDefaultPolicyVersion` action allows them to set any existing version of the policy as the default.

Important

Denying the `iam:SetDefaultPolicyVersion` action in a user's policy does not stop the user from creating a new policy version and setting it as the default.

You can use the following policy to deny a user access to change an existing customer managed policy:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Deny",  
            "Action": [  
                "iam:CreatePolicyVersion",  
                "iam:SetDefaultPolicyVersion"  
            ],  
            "Resource": "arn:aws:iam::*:policy/POLICY-NAME"  
        }  
    ]  
}
```

Setting the default version of customer managed policies

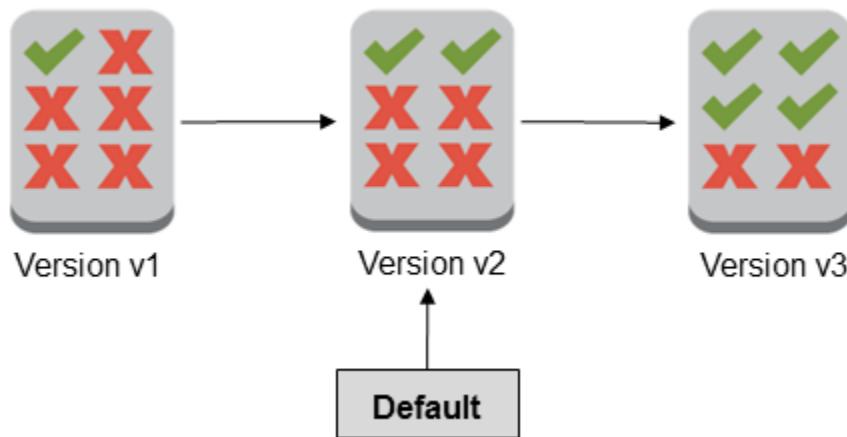
One of the versions of a managed policy is set as the *default* version. The policy's default version is the operative version—that is, it's the version that is in effect for all of the principal entities (users, groups, and roles) that the managed policy is attached to.

When you create a customer managed policy, the policy begins with a single version identified as v1. For managed policies with only a single version, that version is automatically set as the default. For customer managed policies with more than one version, you choose which version to set as the default. For AWS managed policies, the default version is set by AWS. The following diagrams illustrate this concept.

Managed policy with one version



Managed policy with multiple versions



You can set the default version of a customer managed policy to apply that version to every IAM identity (user, group, and role) where the policy is attached. You cannot set the default version for an AWS managed policy or an inline policy.

To set the default version of a customer managed policy (console)

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, choose **Policies**.
3. In the list of policies, choose the policy name of the policy to set the default version of. You can use the **Filter** menu and the search box to filter the list of policies.
4. Choose the **Policy versions** tab. Select the check box next to the version that you want to set as the default version, and then choose **Set as default**.

To learn how to set the default version of a customer managed policy from the AWS Command Line Interface or the AWS API, see [Editing customer managed policies \(AWS CLI\) \(p. 468\)](#).

Using versions to roll back changes

You can set the default version of a customer managed policy to roll back your changes. For example, consider the following scenario:

You create a customer managed policy that allows users to administer a particular Amazon S3 bucket using the AWS Management Console. Upon creation, your customer managed policy has only one version, identified as v1, so that version is automatically set as the default. The policy works as intended.

Later, you update the policy to add permission to administer a second Amazon S3 bucket. IAM creates a new version of the policy, identified as v2, that contains your changes. You set version v2 as the default, and a short time later your users report that they lack permission to use the Amazon S3 console. In this case, you can roll back to version v1 of the policy, which you know works as intended. To do this, you set version v1 as the default version. Your users are now able to use the Amazon S3 console to administer the original bucket.

Later, after you determine the error in version v2 of the policy, you update the policy again to add permission to administer the second Amazon S3 bucket. IAM creates another new version of the policy, identified as v3. You set version v3 as the default, and this version works as intended. At this point, you delete version v2 of the policy.

Version limits

A managed policy can have up to five versions. If you need to make changes to a managed policy beyond five versions from the AWS Command Line Interface, or the AWS API, you must first delete one or more existing versions. If you use the AWS Management Console, you do not have to delete a version before editing your policy. When you save a sixth version, a dialog box appears that prompts you to delete one or more nondefault versions of your policy. You can view the JSON policy document for each version to help you decide. For details about this dialog box, see [the section called "Editing IAM policies" \(p. 465\)](#).

You can delete any version of the managed policy that you want, except for the default version. When you delete a version, the version identifiers for the remaining versions do not change. As a result, version identifiers might not be sequential. For example, if you delete versions v2 and v4 of a managed policy and add two new versions, the remaining version identifiers might be v1, v3, v5, v6, and v7.

Editing IAM policies

A [policy \(p. 351\)](#) is an entity that, when attached to an identity or resource, defines their permissions. Policies are stored in AWS as JSON documents and are attached to principals as *identity-based policies* in IAM. You can attach an identity-based policy to a principal (or identity), such as an IAM group, user, or role. Identity-based policies include AWS managed policies, customer managed policies, and [inline policies \(p. 359\)](#). You can edit customer managed policies and inline policies in IAM. AWS managed policies cannot be edited. The number and size of IAM resources in an AWS account are limited. For more information, see [IAM and STS quotas \(p. 606\)](#).

Topics

- [View policy access \(p. 465\)](#)
- [Editing customer managed policies \(console\) \(p. 466\)](#)
- [Editing inline policies \(console\) \(p. 467\)](#)
- [Editing customer managed policies \(AWS CLI\) \(p. 468\)](#)
- [Editing customer managed policies \(AWS API\) \(p. 468\)](#)

View policy access

Before you change the permissions for a policy, you should review its recent service-level activity. This is important because you don't want to remove access from a principal (person or application) who is using

it. For more information about viewing last accessed information, see [Refining permissions in AWS using last accessed information \(p. 472\)](#).

Editing customer managed policies (console)

You can edit customer managed policies to change the permissions that are defined in the policy. A customer managed policy can have up to five versions. This is important because if you make changes to a managed policy beyond five versions, the AWS Management Console prompts you to decide which version to delete. You can also change the default version or delete a version of a policy before you edit it to avoid being prompted. To learn more about versions, see [Versioning IAM policies \(p. 462\)](#).

To edit a customer managed policy (console)

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, choose **Policies**.
3. In the list of policies, choose the policy name of the policy to edit. You can use the **Filter** menu and the search box to filter the list of policies.
4. Choose the **Permissions** tab, and then choose **Edit policy**.
5. Do one of the following:
 - Choose the **Visual editor** tab to change your policy without understanding JSON syntax. You can make changes to the service, actions, resources, or optional conditions for each permission block in your policy. You can also import a policy to add additional permissions to the bottom of your policy. When you are finished making changes, choose **Review policy** to continue.
 - Choose the **JSON** tab to modify your policy by typing or pasting text in the JSON text box. You can also import a policy to add additional permissions to the bottom of your policy. When you are finished making changes, choose **Review policy** to continue. [Policy Validator \(p. 444\)](#) reports any syntax errors.

Note

You can switch between the **Visual editor** and **JSON** tabs any time. However, if you make changes or choose **Review policy** in the **Visual editor** tab, IAM might restructure your policy to optimize it for the visual editor. For more information, see [Policy restructuring \(p. 571\)](#).

6. On the **Review** page, review the policy **Summary** and then choose **Save changes** to save your work.
7. If the managed policy already has the maximum of five versions, choosing **Save** displays a dialog box. To save your new version, you must remove at least one earlier version. You cannot delete the default version. Choose from the following options:
 - **Remove oldest non-default policy version (version v# - created # days ago)** – Use this option to see which version will be deleted and when it was created. You can view the JSON policy document for all nondefault versions by choosing the second option, **Select versions to remove**.
 - **Select versions to remove** – Use this option to view the JSON policy document and choose one or more versions to delete.

After choosing the versions to remove, choose **Delete version and save** to save your new policy version.

To set the default version of a customer managed policy (console)

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, choose **Policies**.

3. In the list of policies, choose the policy name of the policy to set the default version of. You can use the **Filter** menu and the search box to filter the list of policies.
4. Choose the **Policy versions** tab. Select the check box next to the version that you want to set as the default version, and then choose **Set as default**.

To delete a version of a customer managed policy (console)

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, choose **Policies**.
3. Choose the name of the customer managed policy that has a version you want to delete. You can use the **Filter** menu and the search box to filter the list of policies.
4. Choose the **Policy versions** tab. Select the check box next to the version that you want to delete. Then choose **Delete**.
5. Confirm that you want to delete the version, and then choose **Delete**.

Editing inline policies (console)

You can edit an inline policy from the AWS Management Console.

To edit an inline policy for a user or role (console)

1. In the navigation pane, choose **Users or Roles**.
2. Choose the name of the user or role with the policy that you want to modify. Then choose the **Permissions** tab and expand the policy.
3. To edit an inline policy, choose **Edit Policy**.
4. Do one of the following:
 - Choose the **Visual editor** tab to change your policy without understanding JSON syntax. You can make changes to the service, actions, resources, or optional conditions for each permission block in your policy. You can also import a policy to add additional permissions to the bottom of your policy. When you are finished making changes, choose **Review policy** to continue.
 - Choose the **JSON** tab to modify your policy by typing or pasting text in the JSON text box. You can also import a policy to add additional permissions to the bottom of your policy. When you are finished making changes, choose **Review policy** to continue. [Policy Validator \(p. 444\)](#) reports any syntax errors. To save your changes without affecting the currently attached entities, clear the check box for **Save as default version**.

Note

You can switch between the **Visual editor** and **JSON** tabs any time. However, if you make changes or choose **Review policy** in the **Visual editor** tab, IAM might restructure your policy to optimize it for the visual editor. For more information, see [Policy restructuring \(p. 571\)](#).

5. On the **Review** page, review the policy **Summary** and then choose **Save changes** to save your work.

To edit an inline policy for a group (console)

1. In the navigation pane, choose **Groups**.
2. Choose the name of the group with the policy that you want to modify. Then choose the **Permissions** tab.
3. To edit an inline policy, choose **Edit Policy**.

4. After you have modified your JSON policy, choose **Save** to save your changes.

Editing customer managed policies (AWS CLI)

You can edit a customer managed policy from the AWS Command Line Interface (AWS CLI).

Note

A managed policy can have up to five versions. If you need to make changes to a customer managed policy beyond five versions, you must first delete one or more existing versions.

To edit a customer managed policy (AWS CLI)

1. (Optional) To view information about a policy, run the following commands:
 - To list managed policies: [list-policies](#)
 - To retrieve detailed information about a managed policy: [get-policy](#)
2. (Optional) To find out about the relationships between the policies and identities, run the following commands:
 - To list the identities (users, groups, and roles) to which a managed policy is attached:
 - [list-entities-for-policy](#)
 - To list the managed policies attached to an identity (a user, group, or role):
 - [list-attached-user-policies](#)
 - [list-attached-group-policies](#)
 - [list-attached-role-policies](#)
3. To edit a customer managed policy, run the following command:
 - [create-policy-version](#)

To set the default version of a customer managed policy (AWS CLI)

1. (Optional) To list managed policies, run the following command:
 - [list-policies](#)
2. To set the default version of a customer managed policy, run the following command:
 - [set-default-policy-version](#)

To delete a version of a customer managed policy (AWS CLI)

1. (Optional) To list managed policies, run the following command:
 - [list-policies](#)
2. To delete a customer managed policy, run the following command:
 - [delete-policy-version](#)

Editing customer managed policies (AWS API)

You can edit a customer managed policy using the AWS API.

Note

A managed policy can have up to five versions. If you need to make changes to a customer managed policy beyond five versions, you must first delete one or more existing versions.

To edit a customer managed policy (AWS API)

1. (Optional) To view information about a policy, call the following operations:
 - To list managed policies: [ListPolicies](#)
 - To retrieve detailed information about a managed policy: [GetPolicy](#)
2. (Optional) To find out about the relationships between the policies and identities, call the following operations:
 - To list the identities (users, groups, and roles) to which a managed policy is attached:
 - [ListEntitiesForPolicy](#)
 - To list the managed policies attached to an identity (a user, group, or role):
 - [ListAttachedUserPolicies](#)
 - [ListAttachedGroupPolicies](#)
 - [ListAttachedRolePolicies](#)
3. To edit a customer managed policy, call the following operation:
 - [CreatePolicyVersion](#)

To set the default version of a customer managed policy (AWS API)

1. (Optional) To list managed policies, call the following operation:
 - [ListPolicies](#)
2. To set the default version of a customer managed policy, call the following operation:
 - [SetDefaultPolicyVersion](#)

To delete a version of a customer managed policy (AWS API)

1. (Optional) To list managed policies, call the following operation:
 - [ListPolicies](#)
2. To delete a customer managed policy, call the following operation:
 - [DeletePolicyVersion](#)

Deleting IAM policies

You can delete IAM policies using the AWS Management Console, the AWS Command Line Interface (AWS CLI), or the IAM API.

For more information about the difference between managed and inline policies, see [Managed policies and inline policies \(p. 359\)](#).

For general information about IAM policies, see [Policies and permissions in IAM \(p. 351\)](#).

The number and size of IAM resources in an AWS account are limited. For more information, see [IAM and STS quotas \(p. 606\)](#).

Topics

- [View policy access \(p. 470\)](#)
- [Deleting IAM policies \(console\) \(p. 470\)](#)

- [Deleting IAM policies \(AWS CLI\) \(p. 470\)](#)
- [Deleting IAM policies \(AWS API\) \(p. 471\)](#)

View policy access

Before you delete a policy, you should review its recent service-level activity. This is important because you don't want to remove access from a principal (person or application) who is using it. For more information about viewing last accessed information, see [Refining permissions in AWS using last accessed information \(p. 472\)](#).

Deleting IAM policies (console)

You can delete a customer managed policy to remove it from your AWS account. You cannot delete AWS managed policies.

To delete a customer managed policy (console)

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, choose **Policies**.
3. Select the check box next to the customer managed policy to delete. You can use the **Filter** menu and the search box to filter the list of policies.
4. Choose **Policy actions**, and then choose **Delete**.
5. Confirm that you want to delete the policy, and then choose **Delete**.

To delete an inline policy for a group, user, or role (console)

1. In the navigation pane, choose **Groups, Users, or Roles**.
2. Choose the name of the group, user, or role with the policy that you want to delete. Then choose the **Permissions** tab. If you chose **Users or Roles**, expand the policy.
3. To delete an inline policy in **Groups**, choose **Remove Policy**. To delete an inline policy in **Users or Roles**, choose **X**.

Deleting IAM policies (AWS CLI)

You can delete a customer managed policy from the AWS Command Line Interface.

To delete a customer managed policy (AWS CLI)

1. (Optional) To view information about a policy, run the following commands:
 - To list managed policies: `list-policies`
 - To retrieve detailed information about a managed policy: `get-policy`
2. (Optional) To find out about the relationships between the policies and identities, run the following commands:
 - To list the identities (users, groups, and roles) to which a managed policy is attached, run the following command:
 - `list-entities-for-policy`
 - To list the managed policies attached to an identity (a user, group, or role), run one of the following commands:

- [list-attached-user-policies](#)
 - [list-attached-group-policies](#)
 - [list-attached-role-policies](#)
3. To delete a customer managed policy, run the following command:
- [delete-policy](#)

To delete an inline policy (AWS CLI)

1. (Optional) To list all inline policies that are attached to an identity (user, group, role), use one of the following commands:
 - [aws iam list-user-policies](#)
 - [aws iam list-group-policies](#)
 - [aws iam list-role-policies](#)
2. (Optional) To retrieve an inline policy document that is embedded in an identity (user, group, or role), use one of the following commands:
 - [aws iam get-user-policy](#)
 - [aws iam get-group-policy](#)
 - [aws iam get-role-policy](#)
3. To delete an inline policy from an identity (user, group, or role that is not a [service-linked role \(p. 168\)](#)), use one of the following commands:
 - [aws iam delete-user-policy](#)
 - [aws iam delete-group-policy](#)
 - [aws iam delete-role-policy](#)

Deleting IAM policies (AWS API)

You can delete a customer managed policy using the AWS API.

To delete a customer managed policy (AWS API)

1. (Optional) To view information about a policy, call the following operations:
 - To list managed policies: [ListPolicies](#)
 - To retrieve detailed information about a managed policy: [GetPolicy](#)
2. (Optional) To find out about the relationships between the policies and identities, call the following operations:
 - To list the identities (users, groups, and roles) to which a managed policy is attached, call the following operation:
 - [ListEntitiesForPolicy](#)
 - To list the managed policies attached to an identity (a user, group, or role), call one of the following operations:
 - [ListAttachedUserPolicies](#)
 - [ListAttachedGroupPolicies](#)
 - [ListAttachedRolePolicies](#)
3. To delete a customer managed policy, call the following operation:
 - [DeletePolicy](#)

To delete an inline policy (AWS API)

1. (Optional) To list all inline policies that are attached to an identity (user, group, role), call one of the following operations:
 - [ListUserPolicies](#)
 - [ListGroupPolicies](#)
 - [ListRolePolicies](#)
2. (Optional) To retrieve an inline policy document that is embedded in an identity (user, group, or role), call one of the following operations:
 - [GetUserPolicy](#)
 - [GetGroupPolicy](#)
 - [GetRolePolicy](#)
3. To delete an inline policy from an identity (user, group, or role that is not a *service-linked role* (p. 168)), call one of the following operations:
 - [DeleteUserPolicy](#)
 - [DeleteGroupPolicy](#)
 - [DeleteRolePolicy](#)

Refining permissions in AWS using last accessed information

As an administrator, you might grant permissions to entities (users or roles) beyond what they require. IAM provides last accessed information to help you identify unused permissions so that you can remove them. You can use last accessed information to refine your policies and allow access to only the services and actions that your entities use. This helps you to better adhere to the [best practice of least privilege. \(p. 529\)](#) You can view last accessed information for entities or policies that exist in IAM or AWS Organizations.

Topics

- [Last accessed information types for IAM \(p. 472\)](#)
- [Last accessed information for AWS Organizations \(p. 473\)](#)
- [Things to know about last accessed information \(p. 473\)](#)
- [Permissions required \(p. 474\)](#)
- [Troubleshooting activity for IAM and Organizations entities \(p. 476\)](#)
- [Where AWS tracks last accessed information \(p. 476\)](#)
- [Viewing last accessed information for IAM \(p. 477\)](#)
- [Viewing last accessed information for Organizations \(p. 481\)](#)
- [Example scenarios for using last accessed information \(p. 485\)](#)

Last accessed information types for IAM

You can view two types of last accessed information for IAM entities: allowed AWS service information and allowed action information. The information includes the date and time when the attempt was made. Action last accessed information is available for Amazon S3 management actions. Management actions include creation, deletion, and modification actions. To learn more about how to view last accessed information for IAM, see [Viewing last accessed information for IAM \(p. 477\)](#).

For example scenarios for using last accessed information to make decisions about the permissions that you grant to your IAM entities, see [Example scenarios for using last accessed information \(p. 485\)](#).

To learn more about how the information for management actions is provided, see [Things to know about last accessed information \(p. 473\)](#).

Last accessed information for AWS Organizations

If you sign in using management account credentials, you can view service last accessed information for an AWS Organizations entity or policy in your organization. AWS Organizations entities include the organization root, organizational units (OUs), or accounts. Last accessed information for AWS Organizations includes information about services that are allowed by a service control policy (SCP). The information indicates which principals in an organization or account last attempted to access the service and when. To learn more about the report and how to view last accessed information for AWS Organizations, see [Viewing last accessed information for Organizations \(p. 481\)](#).

For example scenarios for using last accessed information to make decisions about the permissions that you grant to your Organizations entities, see [Example scenarios for using last accessed information \(p. 485\)](#).

Things to know about last accessed information

Before you use last accessed information from a report to change the permissions for an IAM entity or Organizations entity, review the following details about the information.

- **Tracking period** – Recent activity usually appears in the IAM console within four hours. The tracking period for service information is the last 400 days. The tracking period for actions information began on April, 12, 2020. For more information, see [Where AWS tracks last accessed information \(p. 476\)](#).
- **Attempts reported** – The service last accessed data includes all attempts to access an AWS API, not just the successful attempts. This includes all attempts that were made using the AWS Management Console, the AWS API through any of the SDKs, or any of the command line tools. An unexpected entry in the service last accessed data does not mean that your account has been compromised, because the request might have been denied. Refer to your CloudTrail logs as the authoritative source for information about all API calls and whether they were successful or denied access.
- **PassRole** – The `iam:PassRole` action is not tracked and is not included in IAM service last accessed information.
- **Action last accessed information** – Action last accessed information is available for Amazon S3 management actions accessed by IAM entities. IAM provides action information for Amazon S3 management events that are logged by CloudTrail. Sometimes, CloudTrail management events are also called control plane operations or control plane events. Management events provide visibility into administrative operations that are performed on resources in your AWS account. To learn more about management events in CloudTrail, see [Logging Management Events with Cloudtrail](#).
- **Report owner** – Only the principal that generates a report can view the report details. This means that when you view the information in the AWS Management Console, you might have to wait for it to generate and load. If you use the AWS CLI or AWS API to get report details, your credentials must match the credentials of the principal that generated the report. If you use temporary credentials for a role or federated user, you must generate and retrieve the report during the same session. For more information about assumed-role session principals, see [AWS JSON policy elements: Principal \(p. 631\)](#).
- **IAM entities** – The information for IAM includes IAM entities (users or roles) in your account. Information for Organizations includes principals (IAM users, IAM roles, or the AWS account root user) in the specified Organizations entity. The information does not include unauthenticated attempts.
- **IAM policy types** – The information for IAM includes services that are allowed by an IAM entity's policies. These are policies attached to a role or attached to a user directly or through a group. Access allowed by other policy types is not included in your report. The excluded policy types include resource-based policies, access control lists, AWS Organizations SCPs, IAM permissions boundaries, and session policies. Permissions that are provided by service-linked roles are defined by the service that

they are linked to and can't be modified in IAM. To learn more about service-linked roles, see [Using service-linked roles \(p. 213\)](#) To learn how the different policy types are evaluated to allow or deny access, see [Policy evaluation logic \(p. 666\)](#).

- **Organizations policy types** – The information for AWS Organizations includes only services that are allowed by an Organizations entity's inherited service control policies (SCPs). SCPs are policies attached to a root, OU, or account. Access allowed by other policy types is not included in your report. The excluded policy types include identity-based policies, resource-based policies, access control lists, IAM permissions boundaries, and session policies. To learn how the different policy types are evaluated to allow or deny access, see [Policy evaluation logic \(p. 666\)](#).
- **Specifying a policy ID** – When you use the AWS CLI or AWS API to generate a report for last accessed information in Organizations, you can optionally specify a policy ID. The resulting report includes information for the services that are allowed by only that policy. The information includes the most recent account activity in the specified Organizations entity or the entity's children. For more information, see [aws iam generate-organizations-access-report](#) or [GenerateOrganizationsAccessReport](#).
- **Organizations management account** – You must sign in to your organization's management account to view service last accessed information. You can choose to view information for the management account using the IAM console, the AWS CLI, or the AWS API. The resulting report lists all AWS services, because the management account is not limited by SCPs. If you specify a policy ID in the CLI or API, the policy is ignored. For each service, the report includes information for only the management account. However, reports for other Organizations entities do not return information for activity in the management account.
- **Organizations settings** – An administrator must [enable SCPs in your organization root](#) before you can generate data for Organizations.

Permissions required

To view the last accessed information in the AWS Management Console, you must have a policy that grants the necessary permissions.

Permissions for IAM information

To use the IAM console to view the last accessed information for an IAM user, role, or policy, you must have a policy that includes the following actions:

- `iam:GenerateServiceLastAccessedDetails`
- `iam:Get*`
- `iam>List*`

These permissions allow a user to see the following:

- Which users, groups, or roles are attached to a [managed policy](#)
- Which services a user or role can access
- The last time they accessed the service
- The last time they attempted to use a specific S3 action

To use the AWS CLI or AWS API to view last accessed information for IAM, you must have permissions that match the operation you want to use:

- `iam:GenerateServiceLastAccessedDetails`
- `iam:GetServiceLastAccessedDetails`
- `iam:GetServiceLastAccessedDetailsWithEntities`

- `iam>ListPoliciesGrantingServiceAccess`

This example shows how you might create a policy that allows viewing IAM last accessed information. Additionally, it allows read-only access to all of IAM. This policy also grants the necessary permissions to complete this action on the console.

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

Permissions for AWS Organizations information

To use the IAM console to view a report for the root, OU, or account entities in Organizations, you must have a policy that includes the following actions:

- `iam:GenerateOrganizationsAccessReport`
- `iam:GetOrganizationsAccessReport`
- `organizations:DescribeAccount`
- `organizations:DescribeOrganization`
- `organizations:DescribeOrganizationalUnit`
- `organizations:DescribePolicy`
- `organizations>ListChildren`
- `organizations>ListParents`
- `organizations>ListPoliciesForTarget`
- `organizations>ListRoots`
- `organizations>ListTargetsForPolicy`

To use the AWS CLI or AWS API to view service last accessed information for Organizations, you must have a policy that includes the following actions:

- `iam:GenerateOrganizationsAccessReport`
- `iam:GetOrganizationsAccessReport`
- `organizations:DescribePolicy`
- `organizations>ListChildren`
- `organizations>ListParents`
- `organizations>ListPoliciesForTarget`
- `organizations>ListRoots`
- `organizations>ListTargetsForPolicy`

This example shows how you might create a policy that allows viewing service last accessed information for Organizations. Additionally, it allows read-only access to all of Organizations. This policy also grants the necessary permissions to complete this action on the console.

```
{
```

```
"Version": "2012-10-17",
"Statement": [
    "Effect": "Allow",
    "Action": [
        "iam:GenerateOrganizationsAccessReport",
        "iam:GetOrganizationsAccessReport",
        "organizations:Describe*",
        "organizations>List*"
    ],
    "Resource": "*"
}
```

You can also use the [iam:OrganizationsPolicyId \(p. 709\)](#) condition key to allow generating a report only for a specific Organizations policy. For an example policy, see [IAM: View service last accessed information for an Organizations policy \(p. 428\)](#).

Troubleshooting activity for IAM and Organizations entities

In some cases, your AWS Management Console last accessed information table might be empty. Or perhaps your AWS CLI or AWS API request returns an empty set of information or a null field. In these cases, review the following issues:

- For action last accessed information, an action that you are expecting to see might not be returned in the list. This can happen either because the IAM entity does not have permissions for the action, or AWS does not yet track the action for last accessed information.
- For an IAM user, make sure that the user has at least one inline or managed policy attached, either directly or through group memberships.
- For an IAM group, verify that the group has at least one inline or managed policy attached.
- For an IAM group, the report returns only the service last accessed information for members that used the group's policies to access a service. To learn whether a member used other policies, review the last accessed information for that user.
- For an IAM role, verify that the role has at least one inline or managed policy attached.
- For an IAM entity (user or role), review other policy types that might affect the permissions of that entity. These include resource-based policies, access control lists, AWS Organizations policies, IAM permissions boundaries, or session policies. For more information, see [Policy types \(p. 351\)](#) or [Evaluating policies within a single account \(p. 667\)](#).
- For an IAM policy, make sure that the specified managed policy is attached to at least one user, group with members, or role.
- For an Organizations entity (root, OU, or account), make sure that you are signed using Organizations management account credentials.
- Verify that [SCPs are enabled in your organization root](#).
- Action last accessed information is only available for some Amazon S3 actions.

When you make changes, wait at least four hours for activity to appear in your IAM console report. If you use the AWS CLI or AWS API, you must generate a new report to view the updated information.

Where AWS tracks last accessed information

AWS collects last accessed information for the standard AWS Regions. When AWS adds additional Regions, those Regions are added to the following table, including the date that AWS started tracking information in each Region.

- **Service information** – The tracking period for services is the last 400 days, or less if your Region began supporting this feature within the last year.

- Actions information** – The tracking period for Amazon S3 management actions began on April, 12, 2020. If the Region tracking start date is after April 12, 2020, then that date is also the actions tracking start date for the Region.

Region name	Region	Tracking start date
US East (Ohio)	us-east-2	October 27, 2017
US East (N. Virginia)	us-east-1	October 1, 2015
US West (N. California)	us-west-1	October 1, 2015
US West (Oregon)	us-west-2	October 1, 2015
Asia Pacific (Hong Kong)	ap-east-1	April 24, 2019
Asia Pacific (Mumbai)	ap-south-1	June 27, 2016
Asia Pacific (Seoul)	ap-northeast-2	January 6, 2016
Asia Pacific (Singapore)	ap-southeast-1	October 1, 2015
Asia Pacific (Sydney)	ap-southeast-2	October 1, 2015
Asia Pacific (Tokyo)	ap-northeast-1	October 1, 2015
Canada (Central)	ca-central-1	October 28, 2017
Europe (Frankfurt)	eu-central-1	October 1, 2015
Europe (Stockholm)	eu-north-1	December 12, 2018
Europe (Ireland)	eu-west-1	October 1, 2015
Europe (London)	eu-west-2	October 28, 2017
Europe (Milan)	eu-south-1	April 28, 2020
Europe (Paris)	eu-west-3	December 18, 2017
Middle East (Bahrain)	me-south-1	July 29, 2019
Africa (Cape Town)	af-south-1	April 22, 2020
South America (São Paulo)	sa-east-1	December 11, 2015

If a Region is not listed in the previous table, then that Region does not yet provide last accessed information.

An AWS Region is a collection of AWS resources in a geographic area. Regions are grouped into partitions. The standard Regions are the Regions that belong to the `aws` partition. For more information about the different partitions, see [Amazon Resource Names \(ARNs\) Format](#) in the AWS General Reference. For more information about Regions, see [About AWS Regions](#) also in the AWS General Reference.

Viewing last accessed information for IAM

You can view last accessed information for IAM using the AWS Management Console, AWS CLI, or AWS API. Last accessed information includes information about some actions that were last accessed for

Amazon S3. For more information about last accessed information, see [Refining permissions in AWS using last accessed information \(p. 472\)](#).

You can view information for each type of resource in IAM. In each case, the information includes allowed services for the given reporting period:

- **User** – View the last time that the user tried to access each allowed service.
- **Group** – View information about the last time that a group member attempted to access each allowed service. This report also includes the total number of members that attempted access.
- **Role** – View the last time that someone used the role in an attempt to access each allowed service.
- **Policy** – View information about the last time that a user or role attempted to access each allowed service. This report also includes the total number of entities that attempted access.

Note

Before you view the access information for a resource in IAM, make sure you understand the reporting period, reported entities, and the evaluated policy types for your information. For more details, see the section called ["Things to know about last accessed information" \(p. 473\)](#).

Viewing information for IAM (console)

You can view last accessed information for IAM on the **Access Advisor** tab in the IAM console.

To view information for IAM (console)

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, choose either **Groups**, **Users**, **Roles**, or **Policies**.
3. Choose any user, group, role, or policy name to open its **Summary** page and choose the **Access Advisor** tab. View the following information, based on the resource that you chose:
 - **Group** – View the list of services that group members (users) can access. You can also view when a member last accessed the service, what group policies they used, and which group member made the request. Choose the name of the policy to learn whether it is a managed policy or an inline group policy. Choose the name of the group member to see all of the members of the group and when they last accessed the service.
 - **User** – View the list of services that the user can access. You can also view when they last accessed the service, and what policies they used. Choose the name of the policy to learn whether it is a managed policy, an inline user policy, or an inline policy for the group.
 - **Role** – View the list of services that the role can access, when the role last accessed the service, and what policies were used. Choose the name of the policy to learn whether it is a managed policy or an inline role policy.
 - **Policy** – View the list of services with allowed actions in the policy. You can also view when the policy was last used to access the service, and which entity (user or role) used the policy. Choose the name of the entity to learn which entities have this policy attached and when they last accessed the service.
4. (Optional) In the **Service** column of the table, choose **Amazon S3** to view a list of Amazon S3 management actions that IAM entities have attempted to access. You can view the AWS Region and a timestamp that shows when someone last attempted to perform the action.
5. The **Last accessed** column is displayed for services and Amazon S3 management actions. Review the following possible results that are returned in this column. These results vary depending on whether a service or action is allowed, was accessed, and whether it is tracked by AWS for last accessed information.

<number of> days ago

The number of days since the service or action was used in the tracking period. The tracking period for services is for the last 400 days. The tracking period for actions started on April 12, 2020. To learn more about the tracking start dates for each AWS Region, see [Where AWS tracks last accessed information \(p. 476\)](#).

Not accessed in the tracking period

The tracked service or action has not been used by an entity in the tracking period.

It is possible for you to have permissions for an action that doesn't appear in the list. This can happen if the tracking information for the action is not currently supported by AWS. You should not make permissions decisions based solely on the absence of tracking information. Instead, we recommend that you use this information to inform and support your overall strategy of granting least privilege. Check your policies to confirm that the level of access is appropriate.

Viewing information for IAM (AWS CLI)

You can use the AWS CLI to retrieve information about the last time that an IAM resource was used to attempt to access AWS services and Amazon S3 actions. An IAM resource can be a user, group, role, or policy.

To view information for IAM (AWS CLI)

1. Generate a report. The request must include the ARN of the IAM resource (user, group, role, or policy) for which you want a report. You can specify the level of granularity that you want to generate in the report to view access details for either services or both services and actions. The request returns a job-id that you can then use in the `get-service-last-accessed-details` and `get-service-last-accessed-details-with-entities` operations to monitor the job-status until the job is complete.
 - [aws iam generate-service-last-accessed-details](#)
2. Retrieve details about the report using the job-id parameter from the previous step.
 - [aws iam get-service-last-accessed-details](#)

This operation returns the following information, based on the type of resource and level of granularity that you requested in the `generate-service-last-accessed-details` operation:

- **User** – Returns a list of services that the specified user can access. For each service, the operation returns the date and time of the user's last attempt and the ARN of the user.
 - **Group** – Returns a list of services that members of the specified group can access using the policies attached to the group. For each service, the operation returns the date and time of the last attempt made by any group member (user). It also returns the ARN of that user and the total number of group members that have attempted to access the service. Use the [GetServiceLastAccessedDetailsWithEntities](#) operation to retrieve a list of all of the members.
 - **Role** – Returns a list of services that the specified role can access. For each service, the operation returns the date and time of the role's last attempt and the ARN of the role.
 - **Policy** – Returns a list of services for which the specified policy allows access. For each service, the operation returns the date and time that an entity (user or role) last attempted to access the service using the policy. It also returns the ARN of that entity and the total number of entities that attempted access.
3. Learn more about the entities that used group or policy permissions in an attempt to access a specific service. This operation returns a list of entities with each entity's ARN, ID, name, path, type

(user or role), and when they last attempted to access the service. You can also use this operation for users and roles, but it only returns information about that entity.

- [aws iam get-service-last-accessed-details-with-entities](#)
4. Learn more about the identity-based policies that an identity (user, group, or role) used in an attempt to access a specific service. When you specify an identity and service, this operation returns a list of permissions policies that the identity can use to access the specified service. This operation gives the current state of policies and does not depend on the generated report. It also does not return other policy types, such as resource-based policies, access control lists, AWS Organizations policies, IAM permissions boundaries, or session policies. For more information, see [Policy types \(p. 351\)](#) or [Evaluating policies within a single account \(p. 667\)](#).
- [aws iam list-policies-granting-service-access](#)

Viewing information for IAM (AWS API)

You can use the AWS API to retrieve information about the last time that an IAM resource was used to attempt to access AWS services and Amazon S3 actions. An IAM resource can be a user, group, role, or policy. You can specify the level of granularity to generate in the report to view details for either services or both services and actions.

To view information for IAM (AWS API)

1. Generate a report. The request must include the ARN of the IAM resource (user, group, role, or policy) for which you want a report. It returns a `JobId` that you can then use in the `GetServiceLastAccessedDetails` and `GetServiceLastAccessedDetailsWithEntities` operations to monitor the `JobStatus` until the job is complete.
 - [GenerateServiceLastAccessedDetails](#)
2. Retrieve details about the report using the `JobId` parameter from the previous step.
 - [GetServiceLastAccessedDetails](#)

This operation returns the following information, based on the type of resource and level of granularity that you requested in the `GenerateServiceLastAccessedDetails` operation:

- **User** – Returns a list of services that the specified user can access. For each service, the operation returns the date and time of the user's last attempt and the ARN of the user.
 - **Group** – Returns a list of services that members of the specified group can access using the policies attached to the group. For each service, the operation returns the date and time of the last attempt made by any group member (user). It also returns the ARN of that user and the total number of group members that have attempted to access the service. Use the `GetServiceLastAccessedDetailsWithEntities` operation to retrieve a list of all of the members.
 - **Role** – Returns a list of services that the specified role can access. For each service, the operation returns the date and time of the role's last attempt and the ARN of the role.
 - **Policy** – Returns a list of services for which the specified policy allows access. For each service, the operation returns the date and time that an entity (user or role) last attempted to access the service using the policy. It also returns the ARN of that entity and the total number of entities that attempted access.
3. Learn more about the entities that used group or policy permissions in an attempt to access a specific service. This operation returns a list of entities with each entity's ARN, ID, name, path, type (user or role), and when they last attempted to access the service. You can also use this operation for users and roles, but it only returns information about that entity.
- [GetServiceLastAccessedDetailsWithEntities](#)

4. Learn more about the identity-based policies that an identity (user, group, or role) used in an attempt to access a specific service. When you specify an identity and service, this operation returns a list of permissions policies that the identity can use to access the specified service. This operation gives the current state of policies and does not depend on the generated report. It also does not return other policy types, such as resource-based policies, access control lists, AWS Organizations policies, IAM permissions boundaries, or session policies. For more information, see [Policy types \(p. 351\)](#) or [Evaluating policies within a single account \(p. 667\)](#).
 - [ListPoliciesGrantingServiceAccess](#)

Viewing last accessed information for Organizations

You can view service last accessed information for AWS Organizations using the IAM console, AWS CLI, or AWS API. For important information about the data, permissions required, troubleshooting, and supported Regions, see [Refining permissions in AWS using last accessed information \(p. 472\)](#).

When you sign in to the IAM console using AWS Organizations management account credentials, you can view information for any entity in your organization. Organizations entities include the organization root, organizational units (OUs), and accounts. You can also use the IAM console to view information for any service control policies (SCPs) in your organization. IAM shows a list of services that are allowed by any SCPs that apply to the entity. For each service, you can view the most recent account activity information for the chosen Organizations entity or the entity's children.

When you use the AWS CLI or AWS API with management account credentials, you can generate a report for any entities or policies in your organization. A programmatic report for an entity includes a list of services that are allowed by any SCPs that apply to the entity. For each service, the report includes the most recent activity for accounts in the specified Organizations entity or the entity's subtree.

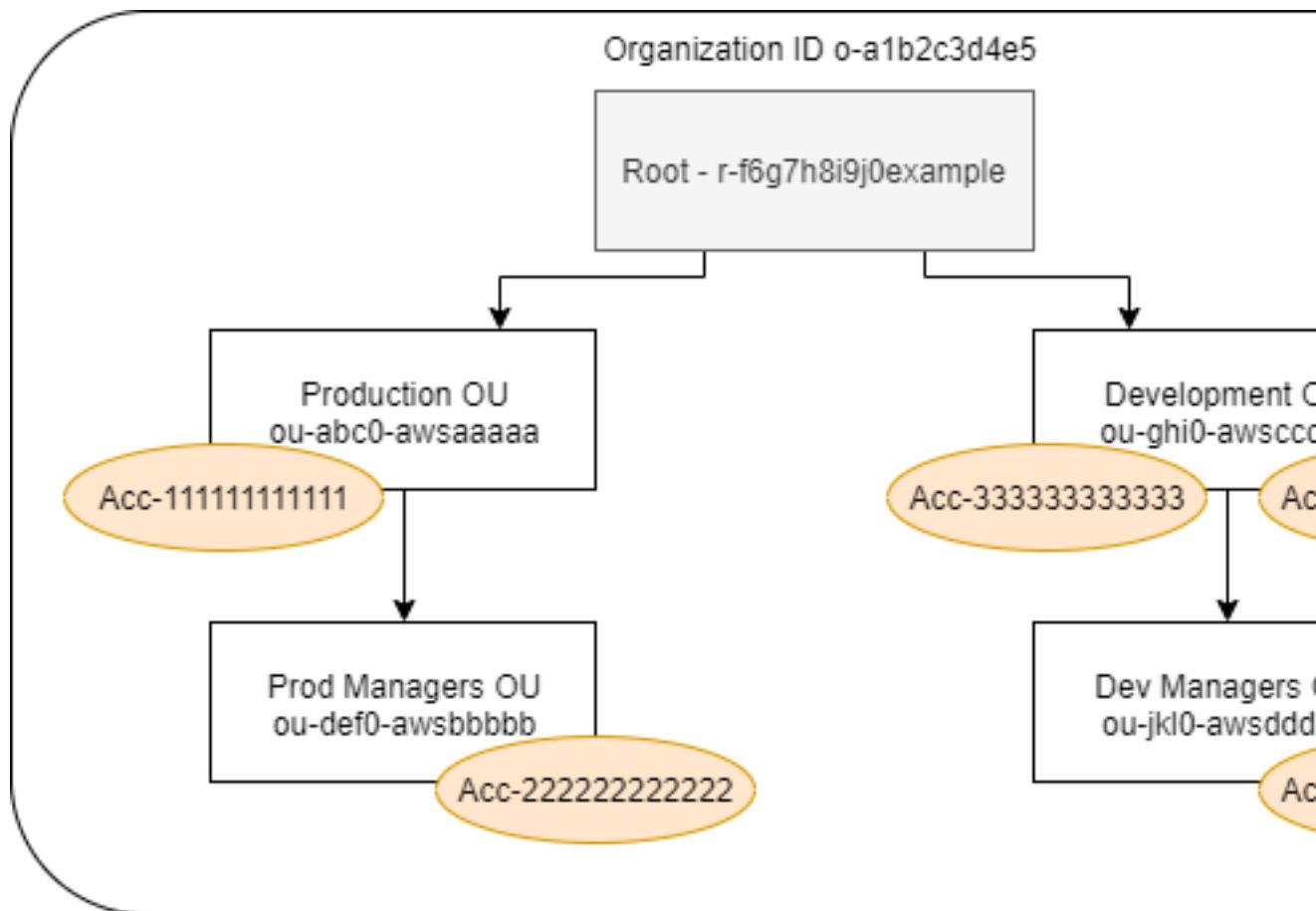
When you generate a programmatic report for a policy, you must specify an Organizations entity. This report includes a list of services that are allowed by the specified SCP. For each service, it includes the most recent account activity in the entity or entity's children that are granted permission by that policy. For more information, see [aws iam generate-organizations-access-report](#) or [GenerateOrganizationsAccessReport](#).

Before you view the report, make sure that you understand the management account requirements and information, reporting period, reported entities, and the evaluated policy types. For more details, see [the section called “Things to know about last accessed information” \(p. 473\)](#).

Understand the AWS Organizations entity path

When you use the AWS CLI or AWS API to generate an AWS Organizations access report, you must specify an entity path. A path is a text representation of the structure of an Organizations entity.

You can build an entity path using the known structure of your organization. For example, assume that you have the following organizational structure in AWS Organizations.



The path for the **Dev Managers** OU is built using the IDs of the organization, root, and all OUs in the path down to and including the OU.

`o-a1b2c3d4e5/r-f6g7h8i9j0example/ou-ghi0-awscccc/ou-jkl0-awsdddd`

The path for the account in the **Production** OU is built using the IDs of the organization, root, the OU, and the account number.

`o-a1b2c3d4e5/r-f6g7h8i9j0example/ou-abc0-awsaaaaa/111111111111`

Note

Organization IDs are globally unique but OU IDs and root IDs are unique only within an organization. This means that no two organizations share the same organization ID. However, another organization might have an OU or root with the same ID as yours. We recommend that you always include the organization ID when you specify an OU or root.

Viewing information for Organizations (console)

You can use the IAM console to view service last accessed information for your root, OU, account, or policy.

To view information for the root (console)

1. Sign in to the AWS Management Console using Organizations management account credentials, and open the IAM console at <https://console.aws.amazon.com/iam/>.

2. In the navigation pane below the **Access reports** section, choose **Organization activity**.
3. On the **Organization activity** page, choose **Root**.
4. On the **Details and activity** tab, view the **Service access report** section. The information includes a list of services that are allowed by the policies that are attached directly to the root. The information shows you from which account the service was last accessed and when. For more details about which principal accessed the service, sign in as an administrator in that account and [view the IAM service last accessed information \(p. 477\)](#).
5. Choose the **Attached SCPs** tab to view the list of the service control policies (SCPs) that are attached to the root. IAM shows you the number of target entities to which each policy is attached. You can use this information to decide which SCP to review.
6. Choose the name of an SCP to view all of the services that the policy allows. For each service, view from which account the service was last accessed, and when.
7. Choose **Edit in AWS Organizations** to view additional details and edit the SCP in the Organizations console. For more information, see [Updating an SCP](#) in the *AWS Organizations User Guide*.

To view information for an OU or account (console)

1. Sign in to the AWS Management Console using Organizations management account credentials, and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane below the **Access reports** section, choose **Organization activity**.
3. On the **Organization activity** page, expand the structure of your organization. Then choose the name of the OU or any account that you want to view except the management account.
4. On the **Details and activity** tab, view the **Service access report** section. The information includes a list of services that are allowed by the SCPs attached to the OU or account *and* all of its parents. The information shows you from which account the service was last accessed and when. For more details about which principal accessed the service, sign in as an administrator in that account and [view the IAM service last accessed information \(p. 477\)](#).
5. Choose the **Attached SCPs** tab to view the list of the service control policies (SCPs) that are attached directly to the OU or account. IAM shows you the number of target entities to which each policy is attached. You can use this information to decide which SCP to review.
6. Choose the name of an SCP to view all of the services that the policy allows. For each service, view from which account the service was last accessed, and when.
7. Choose **Edit in AWS Organizations** to view additional details and edit the SCP in the Organizations console. For more information, see [Updating an SCP](#) in the *AWS Organizations User Guide*.

To view information for the management account (console)

1. Sign in to the AWS Management Console using Organizations management account credentials, and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane below the **Access reports** section, choose **Organization activity**.
3. On the **Organization activity** page, expand the structure of your organization and choose the name your management account.
4. On the **Details and activity** tab, view the **Service access report** section. The information includes a list of all AWS services. The management account is not limited by SCPs. The information shows you whether the account last accessed the service and when. For more details about which principal accessed the service, sign in as an administrator in that account and [view the IAM service last accessed information \(p. 477\)](#).
5. Choose the **Attached SCPs** tab to confirm that there are no attached SCPs because the account is the management account.

To view information for a policy (console)

1. Sign in to the AWS Management Console using Organizations management account credentials, and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane below the **Access reports** section, choose **Service control policies (SCPs)**.
3. On the **Service control policies (SCPs)** page, view a list of the policies in your organization. You can view the number of target entities to which each policy is attached.
4. Choose the name of an SCP to view all of the services that the policy allows. For each service, view from which account the service was last accessed, and when.
5. Choose **Edit in AWS Organizations** to view additional details and edit the SCP in the Organizations console. For more information, see [Updating an SCP](#) in the *AWS Organizations User Guide*.

Viewing information for Organizations (AWS CLI)

You can use the AWS CLI to retrieve service last accessed information for your Organizations root, OU, account, or policy.

To view Organizations service last accessed information (AWS CLI)

1. Use your Organizations management account credentials with the required IAM and Organizations permissions, and confirm that SCPs are enabled for your root. For more information, see [Things to know about last accessed information \(p. 473\)](#).
2. Generate a report. The request must include the path of the Organizations entity (root, OU, or account) for which you want a report. You can optionally include an `organization-policy-id` parameter to view a report for a specific policy. The command returns a `job-id` that you can then use in the `get-organizations-access-report` command to monitor the `job-status` until the job is complete.
 - [aws iam generate-organizations-access-report](#)
3. Retrieve details about the report using the `job-id` parameter from the previous step.
 - [aws iam get-organizations-access-report](#)

This command returns a list of services that entity members can access. For each service, the command returns the date and time of an account member's last attempt and the entity path of the account. It also returns the total number of services that are available to access and the number of services that were not accessed. If you specified the optional `organization-policy-id` parameter, then the services that are available to access are those that are allowed by the specified policy.

Viewing information for Organizations (AWS API)

You can use the AWS API to retrieve service last accessed information for your Organizations root, OU, account, or policy.

To view Organizations service last accessed information (AWS API)

1. Use your Organizations management account credentials with the required IAM and Organizations permissions, and confirm that SCPs are enabled for your root. For more information, see [Things to know about last accessed information \(p. 473\)](#).
2. Generate a report. The request must include the path of the Organizations entity (root, OU, or account) for which you want a report. You can optionally include an `OrganizationPolicyId` parameter to view a report for a specific policy. The operation returns a `JobId` that you can then

use in the `GetOrganizationsAccessReport` operation to monitor the `JobStatus` until the job is complete.

- [GenerateOrganizationsAccessReport](#)
3. Retrieve details about the report using the `JobId` parameter from the previous step.
- [GetOrganizationsAccessReport](#)

This operation returns a list of services that entity members can access. For each service, the operation returns the date and time of an account member's last attempt and the entity path of the account. It also returns the total number of services that are available to access, and the number of services that were not accessed. If you specified the optional `OrganizationsPolicyId` parameter, then the services that are available to access are those that are allowed by the specified policy.

Example scenarios for using last accessed information

You can use last accessed information to make decisions about the permissions that you grant to your IAM entities or AWS Organizations entities. For more information, see [Refining permissions in AWS using last accessed information \(p. 472\)](#).

Note

Before you view the access information for an entity or policy in IAM or AWS Organizations, make sure that you understand the reporting period, reported entities, and the evaluated policy types for your data. For more details, see [the section called "Things to know about last accessed information" \(p. 473\)](#).

It's up to you as an administrator to balance the accessibility and least privilege that's appropriate for your company.

Using information to reduce permissions for an IAM group

You can use last accessed information to reduce IAM group permissions to include only those services that your users need. This method is an important step in [granting least privilege \(p. 529\)](#) at a service level.

For example, Paulo Santos is the administrator in charge of defining AWS user permissions for Example Corp. This company just started using AWS, and the software development team has not yet defined what AWS services they will use. Paulo wants to give the team permission to access only the services they need, but since that is not yet defined, he temporarily gives them power-user permissions. Then he uses last accessed information to reduce the group's permissions.

Paulo creates a managed policy named `ExampleDevelopment` using the following JSON text. He then attaches it to a group named `Development` and adds all of the developers to the group.

Note

Paulo's power users might need `iam:CreateServiceLinkedRole` permissions to use some services and features. He understands that adding this permission allows the users to create any service-linked role. He accepts this risk for his power users.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "FullAccessToAllServicesExceptPeopleManagement",  
            "Effect": "Allow",  
            "NotAction": [  
                "iam:CreateServiceLinkedRole"  
            ]  
        }  
    ]  
}
```

```

        "iam:*",
        "organizations:)"
    ],
    "Resource": "*"
},
{
    "Sid": "RequiredIamAndOrgsActions",
    "Effect": "Allow",
    "Action": [
        "iam:CreateServiceLinkedRole",
        "iam>ListRoles",
        "organizations:DescribeOrganization"
    ],
    "Resource": "*"
}
]
}

```

Paulo decides to wait for 90 days before he [views the last accessed information \(p. 478\)](#) for the Development group using the AWS Management Console. He views the list of services that the group members accessed. He learns that the users accessed five services within the last week: AWS CloudTrail, Amazon CloudWatch Logs, Amazon EC2, AWS KMS, and Amazon S3. They accessed a few other services when they were first evaluating AWS, but not since then.

Paulo decides to reduce the policy permissions to include only those five services and the required IAM and Organizations actions. He edits ExampleDevelopment policy using the following JSON text.

Note

Paulo's power users might need `iam:CreateServiceLinkedRole` permissions to use some services and features. He understands that adding this permission allows the users to create any service-linked role. He accepts this risk for his power users.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "FullAccessToListedServices",
            "Effect": "Allow",
            "Action": [
                "s3:*",
                "kms:*",
                "cloudtrail:*",
                "logs:*",
                "ec2:)"
            ],
            "Resource": "*"
        },
        {
            "Sid": "RequiredIamAndOrgsActions",
            "Effect": "Allow",
            "Action": [
                "iam:CreateServiceLinkedRole",
                "iam>ListRoles",
                "organizations:DescribeOrganization"
            ],
            "Resource": "*"
        }
    ]
}
```

To further reduce permissions, Paulo can view the account's events in AWS CloudTrail [Event history](#). There he can view detailed event information that he can use to reduce the policy's permissions to

include only the actions and resources that the developers need. For more information, see [Viewing CloudTrail Events in the CloudTrail Console](#) in the *AWS CloudTrail User Guide*.

Using information to reduce permissions for an IAM user

You can use last accessed information to reduce the permissions for an individual IAM user.

For example, Martha Rivera is an IT administrator responsible for ensuring that people in her company do not have excess AWS permissions. As part of a periodic security check, she reviews the permissions of all IAM users. One of these users is an application developer named Nikhil Jayashankar, who previously filled the role of a security engineer. Because of the change in job requirements, Nikhil is a member of both the `app-dev` group and the `security-team` group. The `app-dev` group for his new job grants permissions to multiple services including Amazon EC2, Amazon EBS, Auto Scaling, Amazon S3, Route 53, and Elastic Transcoder. The `security-team` group for his old job grants permissions to IAM and CloudTrail.

As an administrator, Martha signs into the IAM console and chooses **Users**, chooses the name `nikhilj`, and then chooses the **Access Advisor** tab.

Martha reviews the **Last Accessed** column and notices that Nikhil has not recently accessed IAM, CloudTrail, Route 53, Amazon Elastic Transcoder, and a number of other AWS services. Nikhil has accessed Amazon S3. Martha chooses **S3** from the list of services and learns that Nikhil has performed some `S3 List` actions in the last two weeks. Within her company, Martha confirms that Nikhil has no business need to access IAM and CloudTrail anymore because he is no longer a member of the internal security team.

Martha is now ready to act on the service and action last accessed information. However, unlike the group in the previous example, an IAM user like `nikhilj` might be subject to multiple policies and be a member of multiple groups. Martha must proceed with caution to avoid inadvertently disrupting access for `nikhilj` or other group members. In addition to learning what access Nikhil should have, she must determine *how* he is receiving these permissions.

Martha chooses the **Permissions** tab, where she views which policies are attached directly to `nikhilj` and those attached from a group. She expands each policy and views the policy summary to learn which policy allows access to the services that Nikhil is not using:

- IAM – The `IAMFullAccess` AWS managed policy is attached directly to `nikhilj` and attached to the `security-team` group.
- CloudTrail – The `AWSCloudTrailReadOnlyAccess` AWS managed policy is attached to the `security-team` group.
- Route 53 – The `App-Dev-Route53` customer managed policy is attached to the `app-dev` group.
- Elastic Transcoder – The `App-Dev-ElasticTranscoder` customer managed policy is attached to the `app-dev` group.

Martha decides to remove the `IAMFullAccess` AWS managed policy that is attached directly to `nikhilj`. She also removes Nikhil's membership to the `security-team` group. These two actions remove the unnecessary access to IAM and CloudTrail.

Nikhil's permissions to access to Route 53 and Elastic Transcoder are granted by the `app-dev` group. Although Nikhil isn't using those services, other members of the group might be. Martha reviews the last accessed information for the `app-dev` group and learns that several members recently accessed Route 53 and Amazon S3. But no group members have accessed Elastic Transcoder in the last year. She removes the `App-Dev-ElasticTranscoder` customer managed policy from the group.

Martha then reviews the last accessed information for the `App-Dev-ElasticTranscoder` customer managed policy. She learns that the policy is not attached to any other IAM identities. She investigates within her company to make sure that the policy will not be needed in the future, and then she deletes it.

Using information before deleting IAM resources

You can use last accessed information before you delete an IAM resource to make sure that a certain amount of time has passed since someone last used the resource. This applies to users, groups, roles, and policies. To learn more about these actions, see the following topics:

- **Users** – [Deleting a user \(p. 85\)](#)
- **Groups** – [Deleting a group \(p. 165\)](#)
- **Roles** – [Deleting a role \(p. 283\)](#)
- **Policies** – [Deleting a managed policy \(this also detaches the policy from identities\) \(p. 469\)](#)

Using information before editing IAM policies

You can review last accessed information for an IAM identity (user, group, or role), or for an IAM policy before editing a policy that affects that resource. This is important because you don't want to remove access for someone that is using it.

For example, Arnav Desai is a developer and AWS administrator for Example Corp. When his team started using AWS, they gave all developers power-user access that allowed them full access to all services except IAM and Organizations. As a first step towards [granting least privilege \(p. 529\)](#), Arnav wants to use the AWS CLI to review the managed policies in his account.

To do this, Arnav first lists the customer managed permissions policies in his account that are attached to an identity, using the following command:

```
aws iam list-policies --scope Local --only-attached --policy-usage-filter PermissionsPolicy
```

From the response, he captures the ARN for each policy. Arnav then generates a report for last accessed information for each policy using the following command.

```
aws iam generate-service-last-accessed-details --arn arn:aws:iam::123456789012:policy/ExamplePolicy1
```

From that response, he captures the ID of the generated report from the `JobId` field. Arnav then polls the following command until the `JobStatus` field returns a value of `COMPLETED` or `FAILED`. If the job failed, he captures the error.

```
aws iam get-service-last-accessed-details --job-id 98a765b4-3cde-2101-2345-example678f9
```

When the job has a status of `COMPLETED`, Arnav parses the contents of the JSON-formatted `ServicesLastAccessed` array.

```
"ServicesLastAccessed": [
    {
        "TotalAuthenticatedEntities": 1,
        "LastAuthenticated": 2018-11-01T21:24:33.222Z,
        "ServiceNamespace": "dynamodb",
        "LastAuthenticatedEntity": "arn:aws:iam::123456789012:user/IAMExampleUser",
        "ServiceName": "Amazon DynamoDB"
    },
    {
        "TotalAuthenticatedEntities": 0,
        "ServiceNamespace": "ec2",
        "ServiceName": "Amazon EC2"
    }
]
```

```
        },
        {
            "TotalAuthenticatedEntities": 3,
            "LastAuthenticated": "2018-08-25T15:29:51.156Z",
            "ServiceNamespace": "s3",
            "LastAuthenticatedEntity": "arn:aws:iam::123456789012:role/IAMExampleRole",
            "ServiceName": "Amazon S3"
        }
    ]
```

From this information, Arnav learns that the `ExamplePolicy1` policy allows access to three services, Amazon DynamoDB, Amazon S3, and Amazon EC2. The IAM user named `IAMExampleUser` last attempted to access DynamoDB on November 1, and someone used the `IAMExampleRole` role to attempt to access Amazon S3 on August 25. There are also two more entities that attempted to access Amazon S3 in the last year. However, nobody has attempted to access Amazon EC2 in the last year.

This means that Arnav can safely remove the Amazon EC2 actions from the policy. Arnav wants to review the current JSON document for the policy. First, he must determine the version number of the policy using the following command.

```
aws iam list-policy-versions --policy-arn arn:aws:iam::123456789012:policy/ExamplePolicy1
```

From the response, Arnav collects the current default version number from the `Versions` array. He then uses that version number (`v2`) to request the JSON policy document using the following command.

```
aws iam get-policy-version --policy-arn arn:aws:iam::123456789012:policy/ExamplePolicy1 --version-id v2
```

Arnav stores the JSON policy document returned in the `Document` field of the `PolicyVersion` array. Within the policy document, Arnav searches for actions with in the `ec2` namespace. If there are no actions from other namespaces remaining in the policy, then he detaches the policy from the affected identities (users, groups, and roles). He then deletes the policy. In this case, the policy does include the Amazon DynamoDB and Amazon S3 services. So Arnav removes the Amazon EC2 actions from the document and saves his changes. He then uses the following command to update the policy using the new version of the document and to set that version as the default policy version.

```
aws iam create-policy-version --policy-arn arn:aws:iam::123456789012:policy/ExamplePolicy1 --policy-document file://UpdatedPolicy.json --set-as-default
```

The `ExamplePolicy1` policy is now updated to remove access to the unnecessary Amazon EC2 service.

Other IAM scenarios

Information about when an IAM resource (user, group, role, or policy) last attempted to access a service can help you when you complete any of the following tasks:

- [Policies – Editing an existing customer-managed or inline policy to remove permissions \(p. 465\)](#)
- [Policies – Converting an inline policy to a managed policy and then deleting it \(p. 530\)](#)
- [Policies – Adding an explicit deny to an existing policy \(p. 673\)](#)
- [Policies – Detaching a managed policy from an identity \(user, group, or role\) \(p. 457\)](#)
- [Entities – Set a permissions boundary to control the maximum permissions that an entity \(user or role\) can have \(p. 454\)](#)
- [Groups – Removing users from a group \(p. 163\)](#)

Using information to refine permissions for an organizational unit

You can use last accessed information to refine the permissions for an organizational unit (OU) in AWS Organizations.

For example, John Stiles is an AWS Organizations administrator. He is responsible for ensuring that people in company AWS accounts do not have excess permissions. As part of a periodic security audit, he reviews the permissions of his organization. His Development OU contains accounts that are often used to test new AWS services. John decides to periodically review the report for services that have not been accessed in more than 180 days. He then removes permissions for the OU members to access those services.

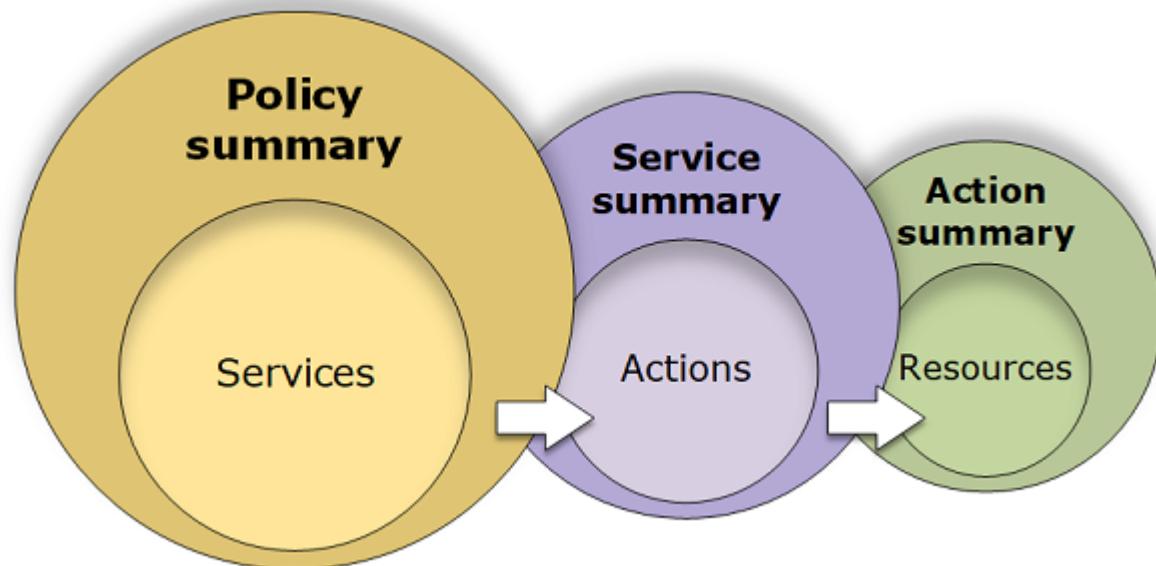
John signs into the IAM console using his management account credentials. In the IAM console, he locates the Organizations data for the Development OU. He reviews the **Service access report** table and sees two AWS services that have not been accessed in more than his preferred period of 180 days. He remembers adding permissions for the development teams to access Amazon Lex and AWS Database Migration Service. John contacts the development teams and confirms that they no longer have a business need to test these services.

John is now ready to act on the last accessed information. He chooses **Edit in AWS Organizations** and is reminded that the SCP is attached to multiple entities. He chooses **Continue**. In AWS Organizations, he reviews the targets to learn to which Organizations entities that the SCP is attached. All of entities are within the Development OU.

John decides to deny access to the Amazon Lex and AWS Database Migration Service actions in the NewServiceTest SCP. This action removes the unnecessary access to the services.

Understanding permissions granted by a policy

The IAM console includes *policy summary* tables that describe the access level, resources, and conditions that are allowed or denied for each service in a policy. Policies are summarized in three tables: the [policy summary \(p. 491\)](#), the [service summary \(p. 501\)](#), and the [action summary \(p. 506\)](#). The *policy summary* table includes a list of services. Choose a service there to see the *service summary*. This summary table includes a list of the actions and associated permissions for the chosen service. You can choose an action from that table to view the *action summary*. This table includes a list of resources and conditions for the chosen action.



You can view policy summaries on the **Users** page or **Roles** page for all policies (managed and inline) that are attached to that user. View summaries on the **Policies** page for all managed policies. Managed policies include AWS managed policies, AWS managed job function policies, and customer managed policies. You can view summaries for these policies on the **Policies** page regardless of whether they are attached to a user or other IAM identity.

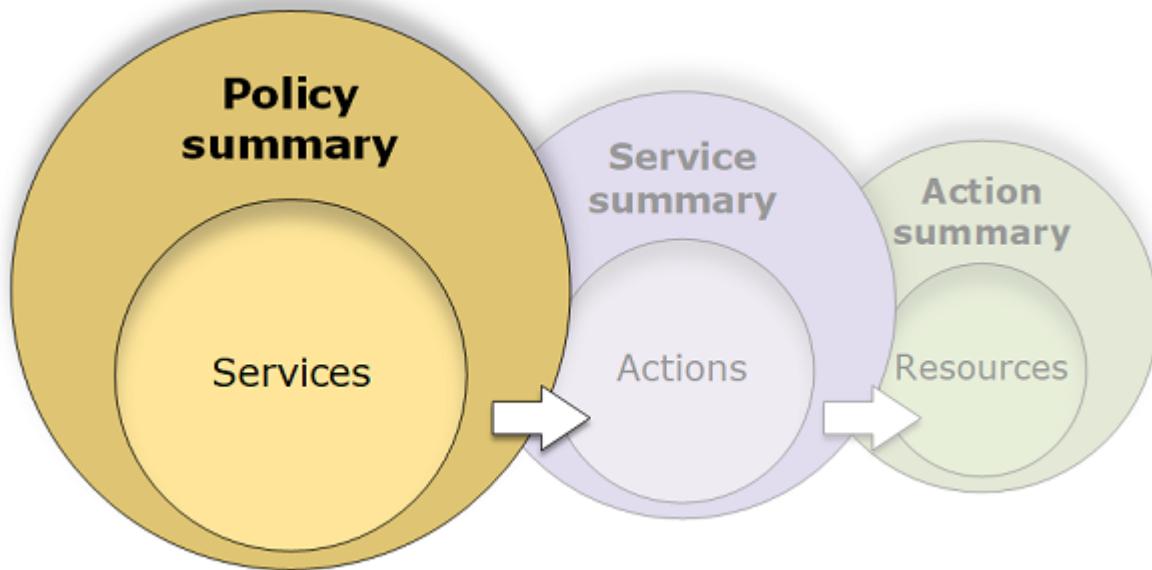
You can use the information in the policy summaries to understand the permissions that are allowed or denied by your policy. Policy summaries can help you [troubleshoot \(p. 570\)](#) and fix policies that are not providing the permissions that you expect.

Topics

- [Policy summary \(list of services\) \(p. 491\)](#)
- [Service summary \(list of actions\) \(p. 501\)](#)
- [Action summary \(list of resources\) \(p. 506\)](#)
- [Examples of policy summaries \(p. 509\)](#)

Policy summary (list of services)

Policies are summarized in three tables: the policy summary, the [service summary \(p. 501\)](#), and the [action summary \(p. 506\)](#). The *policy summary* table includes a list of services and summaries of the permissions that are defined by the chosen policy.



The policy summary table is grouped into one or more **Uncategorized services**, **Explicit deny**, and **Allow** sections. If the policy includes a service that IAM does not recognize, then the service is included in the **Uncategorized services** section of the table. If IAM recognizes the service, then it is included under the **Explicit deny** or **Allow** sections of the table, depending on the effect of the policy (Deny or Allow).

Viewing policy summaries

You can view the summaries for any policies that are attached to a user on the **Users** page. You can view the summaries for any policies that are attached to a role on the **Roles** page. You can view the policy summary for managed policies on the **Policies** page. If your policy does not include a policy summary, see [Missing policy summary \(p. 574\)](#) to learn why.

To view the policy summary from the Policies page

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, choose **Policies**.
3. In the list of policies, choose the name of the policy that you want to view.
4. On the **Summary** page for the policy, view the **Permissions** tab to see the policy summary.

To view the summary for a policy attached to a user

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. Choose **Users** from the navigation pane.
3. In the list of users, choose the name of the user whose policy you want to view.
4. On the **Summary** page for the user, view the **Permissions** tab to see the list of policies that are attached to the user directly or from a group.
5. In the table of policies for the user, expand the row of the policy that you want to view.

To view the summary for a policy attached to a role

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, choose **Roles**.
3. In the list of roles, choose the name of the role whose policy you want to view.
4. On the **Summary** page for the role, view the **Permissions** tab to see the list of policies that are attached to the role.
5. In the table of policies for the role, expand the row of the policy that you want to view.

Editing policies to fix warnings

While viewing a policy summary, you might find a typo or notice that the policy does not provide the permissions that you expected. You cannot edit a policy summary directly. However, you can edit a customer managed policy using the visual policy editor, which catches many of the same errors and warnings that the policy summary reports. You can then view the changes in the policy summary to confirm that you fixed all of the issues. To learn how to edit an inline policy, see [the section called "Editing IAM policies" \(p. 465\)](#). You cannot edit AWS managed policies.

To edit a policy for your policy summary using the Visual editor tab

1. Open the policy summary as explained in the previous procedures.
2. Choose **Edit policy**.

If you are on the **Users** page and choose to edit a customer managed policy that is attached to that user, you are redirected to the **Policies** page. You can edit customer managed policies only on the **Policies** page.

3. Choose the **Visual editor** tab to view the editable visual representation of your policy. IAM might restructure your policy to optimize it for the visual editor and to make it easier for you to find and fix any problems. The warnings and error messages on the page can guide you to fix any issues with your policy. For more information about how IAM restructures policies, see [Policy restructuring \(p. 571\)](#).
4. Edit your policy and choose **Review policy** to see your changes reflected in the policy summary. If you still see a problem, choose **Previous** to return to the editing screen.

5. Choose **Save** to save your changes.

To edit a policy for your policy summary using the JSON tab

1. Open the policy summary as explained in the previous procedures.
2. Choose **{ } JSON** and **Policy summary** to compare the policy summary to the JSON policy document. You can use this information to determine which lines in the policy document you want to change.
3. Choose **Edit policy** and then choose the **JSON** tab to edit the JSON policy document.

Note

You can switch between the **Visual editor** and **JSON** tabs any time. However, if you make changes or choose **Review policy** in the **Visual editor** tab, IAM might restructure your policy to optimize it for the visual editor. For more information, see [Policy restructuring \(p. 571\)](#).

If you are on the **Users** page and choose to edit a customer managed policy that is attached to that user, you are redirected to the **Policies** page. You can edit customer managed policies only on the **Policies** page.

4. Edit your policy and choose **Review policy** to see your changes reflected in the policy summary. If you still see a problem, choose **Previous** to return to the editing screen.
5. Choose **Save** to save your changes.

Understanding the elements of a policy summary

In the following example of a user details page, the **PolSumUser** user has eight attached policies. The **SummaryAllElements** policy is a managed policy (customer managed policy) that is attached directly to the user. This policy is expanded to show the policy summary. To view the JSON policy document for this policy, see [the section called “SummaryAllElements JSON policy document” \(p. 498\)](#).

The screenshot shows the AWS IAM User Details page for a user named PolSumUser. The 'Permissions' tab is selected. At the top, it displays the User ARN (arn:aws:iam::072398337363:user/PolSumUser), Path (/), and Creation time (2017-02-16 12:58 PDT). Below this, there are tabs for 'Permissions', 'Groups (1)', 'Security credentials', and 'Access Advisor'. A blue button labeled 'Add permissions' is visible. The 'Attached policies' section shows 8 policies. One policy, 'SummaryAllElements', is expanded, revealing a warning message: 'This policy defines some actions, resources, or conditions that do not provide permissions. To grant access, policies must have an action that has an applicable resource or condition. For details, choose Show remaining. Learn more'. Below this, there are buttons for 'Policy summary' (highlighted with a red circle 4), '{ } JSON', 'Edit policy', and 'Simulate policy' (highlighted with a red circle 5). A search bar labeled 'Filter' (highlighted with a red circle 6) is also present. A large red box highlights the 'Service' table below, which lists various services and their access levels. The table includes columns for Service, Access level, Resource, and Request condition. It shows entries for Unrecognized services (codedploy), Explicit deny (S3, Billing, EC2), and Allow (S3). A red circle 7 points to the top-left corner of this table.

Service	Access level	Resource	Request condition
Unrecognized services			
codedploy			
Explicit deny (1 of 103 services)			
S3	Full: Read, Write, Permissions management Limited: List	Multiple	None
Allow (3 of 103 services) Show remaining 100			
Billing	Full: Read Limited: Write	All resources	Multiple
EC2	None	All resources	None
S3	Limited: Write, Permissions management	BucketName = developer_bucket, ObjectPath = All	s3:x-amz-acl = public-read

In the preceding image, the policy summary is visible from within the user details page:

1. The **Permissions** tab for a user includes the policies that are attached to the **PolSumUser** user.
2. The **SummaryAllElements** policy is one of several policies that are attached to the user. The policy is expanded in order to view the policy summary.
3. If the policy does not grant permissions to all the actions, resources, and conditions defined in the policy, then a warning or error banner appears at the top of the page. The policy summary then includes details about the problem. To learn how policy summaries help you to understand and troubleshoot the permissions that your policy grants, see [the section called "My policy does not grant the expected permissions" \(p. 576\)](#).
4. Use the **Policy summary** and **{ } JSON** buttons to toggle between the policy summary and the JSON policy document.
5. **Simulate policy** opens the policy simulator for testing the policy.
6. Use the search box to reduce the list of services and easily find a specific service.
7. The expanded view shows additional details of the **SummaryAllElements** policy.

The following policy summary table image shows the expanded **SummaryAllElements** policy on the **PolSumUser** user details page.

A Service	G Access level	H Resource	I Request condition
B Unrecognized services			
codeddeploy			
C Explicit deny (1 of 103 services)			
S3	Full: Read, Write, Permissions management Limited: List	Multiple	None
D Allow (3 of 103 services) Show remaining 100			
Billing	Full: Read Limited: Write	All resources	Multiple
E EC2	None	All resources	None
F S3	Limited: Write, Permissions management	BucketName = developer_bucket, ObjectPath = All	s3:x-amz-acl = public-read

In the preceding image, the policy summary is visible from within the user details page:

- A. **Service** – This column lists the services that are defined within the policy and provides details for each service. Each service name in the policy summary table is a link to the *service summary* table, which is explained in [Service summary \(list of actions\) \(p. 501\)](#). In this example, permissions are defined for the Amazon S3, Billing, and Amazon EC2 services. The policy also defines permissions for a (misspelled) codedploy service, which IAM does not recognize.
- B. **Unrecognized services** – This policy includes an unrecognized service (in this case **codedploy**). You can use this warning to check whether a service name might include a typo. If the service name is correct, then the service might not support policy summaries, might be in preview, or might be a custom service. To request policy summary support for a generally available (GA) service, see [Service does not support IAM policy summaries \(p. 575\)](#). In this example, the policy includes an unrecognized codedploy service that is missing an e. Because of this typo, the policy does not provide the expected AWS CodeDeploy permissions. You can [edit the policy \(p. 492\)](#) to include the accurate codedploy service name; the service then appears in the policy summary.
- C. For those services that IAM recognizes, it arranges services according to whether the policy allows or explicitly denies the use of the service. In this example, the policy includes **Allow** and **Deny** statements for the Amazon S3 service. Therefore the policy summary includes S3 within both the **Explicit deny** and **Allow** sections.
- D. **Show remaining 100** – Choose this link to expand the table to include the services that are not defined by the policy. These services are *implicitly denied* (or denied by default) within this policy. However, a statement in another policy might still allow or explicitly deny using the service. The policy summary summarizes the permissions of a single policy. To learn about how the AWS service decides whether a given request should be allowed or denied, see [Policy evaluation logic \(p. 666\)](#).
- E. **EC2** – This service includes an unrecognized action. IAM recognizes service names, actions, and resource types for services that support policy summaries. When a service is recognized but contains an action that is not recognized, IAM includes a warning next to that service. In this example, IAM can't recognize at least one Amazon EC2 action. To learn more about unrecognized actions and to view the unrecognized action in an S3 service summary, see [Service summary \(list of actions\) \(p. 501\)](#).
- F. **S3** – This service includes an unrecognized resource. IAM recognizes service names, actions, and resource types for services that support policy summaries. When a service is recognized but contains

Note

IAM reviews service names, actions, and resource types for services that support policy summaries. However, your policy summary might include a resource value or condition that does not exist. Always test your policies with the [policy simulator \(p. 445\)](#).

- F. **S3** – This service includes an unrecognized resource. IAM recognizes service names, actions, and resource types for services that support policy summaries. When a service is recognized but contains

a resource type that is not recognized, IAM includes a warning next to that service. In this example, IAM can't recognize at least one Amazon S3 action. To learn more about unrecognized resources and to view the unrecognized resource type in an S3 service summary, see [Service summary \(list of actions\) \(p. 501\)](#).

G. Access level – This column tells whether the actions in each access level (List, Read, Write, and Permissions management) have Full or Limited permissions defined in the policy. For additional details and examples of the access level summary, see [Understanding access level summaries within policy summaries \(p. 500\)](#).

- **Full access** – This entry indicates that the service has access to all actions within all four of the access levels available for the service. In this example, because this row is in the **Explicit deny** section of the table, all Amazon S3 actions are denied for the resources included in the policy.
- If the entry does not include **Full access**, then the service has access to some but not all of the actions for the service. The access is then defined by following descriptions for each of the four access level classifications (List, Read, Write, and Permissions management):

Full: The policy provides access to all actions within each access level classification listed. In this example, the policy provides access to all of the Billing Read actions.

Limited: The policy provides access to one or more but not all actions within each access level classification listed. In this example, the policy provides access to some of the Billing Write actions.

H. Resource – This column shows the resources that the policy specifies for each service.

- **Multiple** – The policy includes more than one but not all of the resources within the service. In this example, access is explicitly denied to more than one Amazon S3 resource.
- **All resources** -- The policy is defined for all resources within the service. In this example, the policy allows the listed actions to be performed on all Billing resources.
- **Resource text** – The policy includes one resource within the service. In this example, the listed actions are allowed on only the developer_bucket Amazon S3 bucket resource. Depending on the information that the service provides to IAM, you might see an ARN such as arn:aws:s3:::developer_bucket/*, or you might see the defined resource type, such as BucketName = developer_bucket.

Note

This column can include a resource from a different service. If the policy statement that includes the resource does not include both actions and resources from the same service, then your policy includes mismatched resources. IAM does not warn you about mismatched resources when you create a policy, or when you view a policy in the policy summary. If this column includes a mismatched resource, then you should review your policy for errors. To better understand your policies, always test them with the [policy simulator \(p. 445\)](#).

I. Request condition – This column indicates whether the services or actions associated with the resource are subject to conditions.

- **None** – The policy includes no conditions for the service. In this example no conditions are applied to the denied actions in the Amazon S3 service.
- **Condition text** – The policy includes one condition for the service. In this example, the listed **Billing** actions are allowed only if the IP address of the source matches 203.0.113.0/24.
- **Multiple** – The policy includes more than one condition for the service. In this example, access to the listed Amazon S3 actions is allowed based on more than one condition. To view each of the multiple conditions for the policy, choose **{ } JSON** to view the policy document.

When a policy or an element within the policy does not grant permissions, IAM provides additional warnings and information in the policy summary. The following policy summary table shows the expanded **Show remaining 100** services on the **PolSumUser** user details page with the possible warnings.

Service	Access level	a Resource	b Request condition
Unrecognized services			
codedploy			
Explicit deny (1 of 103 services)			
S3	Full: Read, Write, Permissions management Limited: List	Multiple ! One or more actions do not have an applicable resource.	None
Allow (3 of 103 services) Hide remaining 100			
d ...	e None	f	g
Billing	Full: Read Limited: Write	All resources	Multiple
CodeBuild	! None - No actions are defined.	arn:aws:codebuild:us-east-1:123456789012:project/my-demo-project	None
CodeCommit	None	! No resources are defined.	None
CodeDeploy	! None - No actions are defined.	arn:aws:codedeploy:us-west-2:123456789012:deploymentgroup:*	None
EC2	None	All resources	None
S3	Limited: Write, Permissions management	BucketName = developer_bucket, ObjectPath = All ! One or more resources do not have an applicable action.	s3:x-amz-acl = public-read ! One or more conditions do not have an applicable action.

In the preceding image, you can see all services that include defined actions, resources, or conditions with no permissions:

- a. **Resource warnings** – For services that do not provide permissions for all of the included actions or resources, you see one of the following warnings in the **Resource** column of the table:

- ! **No resources are defined.** – This means that the service has defined actions but no supported resources are included in the policy.
- ! **One or more actions do not have an applicable resource.** – This means that the service has defined actions, but that some of those actions don't have a supported resource.
- ! **One or more resources do not have an applicable action.** – This means that the service has defined resources, but that some of those resources don't have a supporting action.

If a service includes both actions that do not have an applicable resource and resources that do not have an applicable resource, then only the **One or more resources do not have an applicable action** warning is shown. This is because when you view the service summary for the service, resources that do not apply to any action are not shown. For the `ListAllMyBuckets` action, this policy includes the last warning because the action does not support resource-level permissions, and does not support the `s3:x-amz-acl` condition key. If you fix either the resource problem or the condition problem, the remaining issue appears in a detailed warning.

- b. **Request condition warnings** – For services that do not provide permissions for all of the included conditions, you see one of the following warnings in the **Request condition** column of the table:

- ! **One or more actions do not have an applicable condition.** – This means that the service has defined actions, but that some of those actions don't have a supported condition.
- ! **One or more conditions do not have an applicable action.** – This means that the service has defined conditions, but that some of those conditions don't have a supporting action.

- c. **Multiple |  One or more actions do not have an applicable resource.** – The Deny statement for Amazon S3 includes more than one resource. It also includes more than one action, and some actions support the resources and some do not. To view this policy, see [the section called "SummaryAllElements JSON policy document" \(p. 498\)](#). In this case, the policy includes all Amazon S3 actions, and only the actions that can be performed on a bucket or bucket object are denied.
- d. The ellipses (...) indicate that all the services are included in the page, but we are showing only the rows with information relevant to this policy. When you view this page in the AWS Management Console, you see all the AWS services.
- e. The background color in the table rows indicates services that do not grant any permissions. You cannot get any additional information about these services in the policy summary. For services in white rows, you can choose the name of the service to view the service summary (list of actions) page. There you can learn more about the permissions granted for that service.
- f. ** None - No actions are defined.** – This means that the service is defined as a resource or condition, but that no actions are included for the service, and therefore the service provides no permissions. In this case, the policy includes a CodeBuild resource but no CodeBuild actions.
- g. ** No resources are defined** – The service has defined actions, but no supported resources are included in the policy, and therefore the service provides no permissions. In this case, the policy includes CodeCommit actions but no CodeCommit resources.
- h. **BucketName = developer_bucket, ObjectPath = All |  One or more resources do not have an applicable action.** – The service has a defined bucket object resource, and at least one more resource that does not have a supporting action.
- i. **s3:x-amz-acl = public-read |  One or more conditions do not have an applicable action.** – The service has a defined s3:x-amz-acl condition key, and at least one more condition key that does not have a supporting action.

SummaryAllElements JSON policy document

The **SummaryAllElements** policy is not intended for you to use to define permissions in your account. Rather, it is included to demonstrate the errors and warnings that you might encounter while viewing a policy summary.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "aws-portal:ViewBilling",
                "aws-portal:ViewPaymentMethods",
                "aws-portal:ModifyPaymentMethods",
                "aws-portal:ViewAccount",
                "aws-portal:ModifyAccount",
                "aws-portal:ViewUsage"
            ],
            "Resource": [
                "*"
            ],
            "Condition": {
                "IpAddress": {
                    "aws:SourceIp": "203.0.113.0/24"
                }
            }
        },
    ]
},
```

```
{  
    "Effect": "Deny",  
    "Action": [  
        "s3:*"  
    ],  
    "Resource": [  
        "arn:aws:s3:::customer",  
        "arn:aws:s3:::customer/*"  
    ]  
},  
{  
    "Effect": "Allow",  
    "Action": [  
        "ec2:GetConsoleScreenshots"  
    ],  
    "Resource": [  
        "*"  
    ]  
},  
{  
    "Effect": "Allow",  
    "Action": [  
        "codedeploy:*",  
        "codecommit:*"  
    ],  
    "Resource": [  
        "arn:aws:codedeploy:us-west-2:123456789012:deploymentgroup:*,  
        "arn:aws:codebuild:us-east-1:123456789012:project/my-demo-project"  
    ]  
},  
{  
    "Effect": "Allow",  
    "Action": [  
        "s3>ListAllMyBuckets",  
        "s3:GetObject",  
        "s3>DeleteObject",  
        "s3:PutObject",  
        "s3:PutObjectAcl"  
    ],  
    "Resource": [  
        "arn:aws:s3:::developer_bucket",  
        "arn:aws:s3:::developer_bucket/*",  
        "arn:aws:autoscaling:us-east-2:123456789012:autoscalgrp"  
    ],  
    "Condition": {  
        "StringEquals": {  
            "s3:x-amz-acl": [  
                "public-read"  
            ],  
            "s3:prefix": [  
                "custom",  
                "other"  
            ]  
        }  
    }  
}  
]  
}
```

Understanding access level summaries within policy summaries

AWS access level summary

Policy summaries include an access level summary that describes the action permissions defined for each service that is mentioned in the policy. To learn about policy summaries, see [Understanding permissions granted by a policy \(p. 490\)](#). Access level summaries indicate whether the actions in each access level (List, Read, Write, and Permissions management) have Full or Limited permissions defined in the policy. To view the access level classification that is assigned to each action in a service, see [Actions, Resources, and Condition Keys for AWS Services](#).

The following example describes the access provided by a policy for the given services. For examples of full JSON policy documents and their related summaries, see [Examples of policy summaries \(p. 509\)](#).

Service	Access level	This policy provides the following
IAM	Full access	Access to all actions within the IAM service.
CloudWatch	Full: List	Access to all CloudWatch actions in the List access level, but no access to actions with the Read, Write, or Permissions management access level classification.
Data Pipeline	Limited: List, Read	Access to at least one but not all AWS Data Pipeline actions in the List and Read access level, but not the Write or Permissions management actions.
EC2	Full: List, Read Limited: Write	Access to all Amazon EC2 List and Read actions and access to at least one but not all Amazon EC2 Write actions, but no access to actions with the Permissions management access level classification.
S3	Limited: Read, Write, Permissions management	Access to at least one but not all Amazon S3 Read, Write and Permissions management actions.
CodeDeploy	(empty)	Unknown access, because IAM does not recognize this service.
API Gateway	None	No access is defined in the policy.
CodeBuild	 No actions are defined.	No access because no actions are defined for the service. To learn how to understand and troubleshoot this issue, see the section called "My policy does not grant the expected permissions" (p. 576) .

As [previously mentioned \(p. 496\)](#), **Full access** indicates that the policy provides access to all the actions within the service. Policies that provide access to some but not all actions within a service are further grouped according to the access level classification. This is indicated by one of the following access-level groupings:

- **Full:** The policy provides access to all actions within the specified access level classification.

- **Limited:** The policy provides access to one or more but not all actions within the specified access level classification.
- **None:** The policy provides no access.
- (empty): IAM does not recognize this service. If the service name includes a typo, then the policy provides no access to the service. If the service name is correct, then the service might not support policy summaries or might be in preview. In this case, the policy might provide access, but that access cannot be shown in the policy summary. To request policy summary support for a generally available (GA) service, see [Service does not support IAM policy summaries \(p. 575\)](#).

Access level summaries that include limited (partial) access to actions are grouped using the AWS access level classifications **List**, **Read**, **Write**, **Permissions Management**, or **Tagging**.

AWS access levels

AWS defines the following access level classifications for the actions in a service:

- **List:** Permission to list resources within the service to determine whether an object exists. Actions with this level of access can list objects but cannot see the contents of a resource. For example, the Amazon S3 action `ListBucket` has the **List** access level.
- **Read:** Permission to read but not edit the contents and attributes of resources in the service. For example, the Amazon S3 actions `GetObject` and `GetBucketLocation` have the **Read** access level.
- **Write:** Permission to create, delete, or modify resources in the service. For example, the Amazon S3 actions `CreateBucket`, `DeleteBucket` and `PutObject` have the **Write** access level. **Write** actions might also allow modifying a resource tag. However, an action that allows only changes to tags has the **Tagging** access level.
- **Permissions management:** Permission to grant or modify resource permissions in the service. For example, most IAM and AWS Organizations actions, as well as actions like the Amazon S3 actions `PutBucketPolicy` and `DeleteBucketPolicy` have the **Permissions management** access level.

Tip

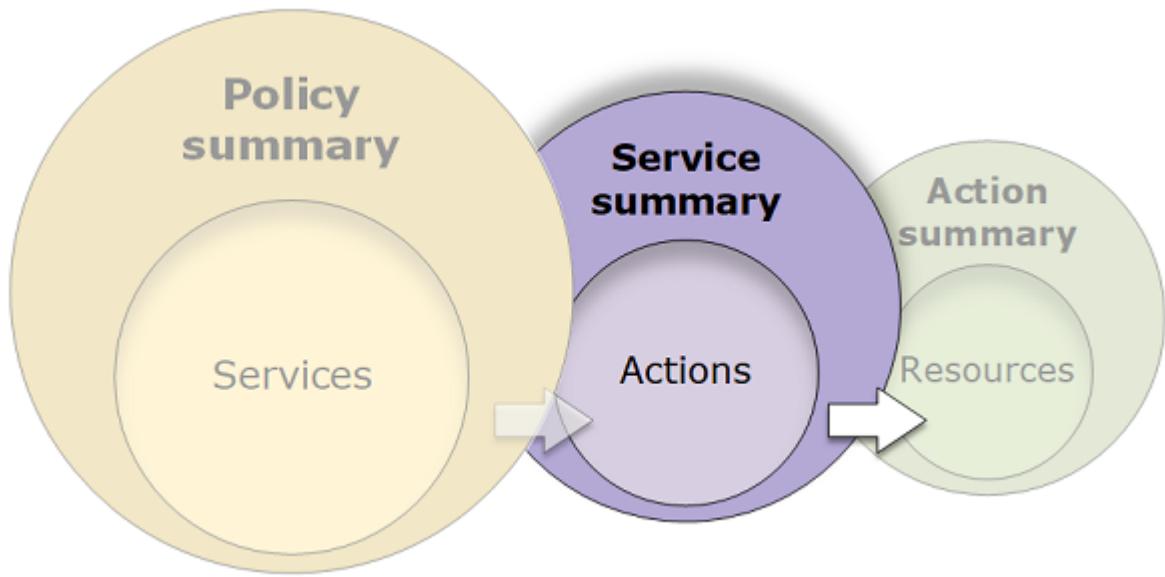
To improve the security of your AWS account, restrict or regularly monitor policies that include the **Permissions management** access level classification.

- **Tagging:** Permission to perform actions that only change the state of resource tags. For example, the IAM actions `TagRole` and `UntagRole` have the **Tagging** access level because they allow only tagging or untagging a role. However, the `CreateRole` action allows tagging a role resource when you create that role. Because the action does not only add a tag, it has the **Write** access level.

To view the access level classification for all of the actions in a service, see [Actions, Resources, and Condition Keys for AWS Services](#).

Service summary (list of actions)

Policies are summarized in three tables: the [policy summary \(p. 491\)](#), the service summary, and the [action summary \(p. 506\)](#). The *service summary* table includes a list of the actions and summaries of the permissions that are defined by the policy for the chosen service.



You can view a service summary for each service listed in the policy summary that grants permissions. The table is grouped into **Uncategorized actions**, **Uncategorized resource types**, and access level sections. If the policy includes an action that IAM does not recognize, then the action is included in the **Uncategorized actions** section of the table. If IAM recognizes the action, then it is included under one of the access level (**List**, **Read**, **Write** and **Permissions management**) sections of the table. To view the access level classification that is assigned to each action in a service, see [Actions, Resources, and Condition Keys for AWS Services](#).

Viewing service summaries

You can view the service summary for managed policies on the **Policies** page, or view service summaries for inline and managed policies attached to a user or role through the **Users** page and **Roles** page. However, if you choose a service name on the **Users** page or **Roles** page from a managed policy, you are redirected to the **Policies** page. Service summaries for managed policies must be viewed on the **Policies** page.

To view the service summary for a managed 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, choose **Policies**.
3. In the list of policies, choose the name of the policy that you want to view.
4. On the **Summary** page for the policy, view the **Permissions** tab to see the policy summary.
5. In the policy summary list of services, choose the name of the service that you want to view.

To view the service summary for a policy attached to 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, choose **Users**.
3. In the list of users, choose the name of the user whose policy you want to view.
4. On the **Summary** page for the user, view the **Permissions** tab to see the list of policies that are attached to the user directly or from a group.

5. In the table of policies for the user, expand the row of the policy that you want to view.
6. In the policy summary list of services, choose the name of the service that you want to view.

Note

If the policy that you select is an inline policy that is attached directly to the user, then the service summary table appears. If the policy is an inline policy attached from a group, then you are taken to the JSON policy document for that group. If the policy is a managed policy, then you are taken to the service summary for that policy on the **Policies** page.

To view the service summary for a policy attached to a role

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. Choose **Roles** from the navigation pane.
3. In the list of roles, choose the name of the role whose policy you want to view.
4. On the **Summary** page for the role, view the **Permissions** tab to see the list of policies that are attached to the role.
5. In the table of policies for the role, expand the row of the policy that you want to view.
6. In the policy summary list of services, choose the name of the service that you want to view.

Understanding the elements of a service summary

The example below is the service summary for Amazon S3 actions that are allowed from the **SummaryAllElements** policy summary (see [the section called "SummaryAllElements JSON policy document" \(p. 498\)](#)). The actions for this service are grouped by **Uncategorized actions**, **Uncategorized resource types**, and access level. For example, two **Write** actions are defined out of the total 29 **Write** actions available for the service.

1 This policy defines some actions, resources, or conditions that do not provide permissions. To grant access, policies must have an action that has an applicable resource or condition. For details, choose Show remaining. [Learn more](#)

2 Back

3 S3

4 Policy summary

5 Action (2 of 69) Hide remaining 67

6 Resource

7 Unrecognized resource types

8 Unrecognized actions

9 List (0 of 4 actions)

10 ... (No access)

11 ListAllMyBuckets(No access)

12 PutObject

13 Request condition

14 This action does not have an applicable resource and condition.

15 s3:x-amz-acl = public-read is not a supported condition key for this action.

16 s3:x-amz-acl = public-read

Action	Resource	Condition
ListAllMyBuckets(No access)		None
GetObject(No access)	BucketName = developer_bucket, ObjectPath = All	s3:x-amz-acl = public-read
PutObject	BucketName = developer_bucket, ObjectPath = All	s3:x-amz-acl = public-read
PutObjectAcl	BucketName = developer_bucket, ObjectPath = All	s3:x-amz-acl = public-read

The service summary page for a managed policy includes the following information:

- If the policy does not grant permissions to all the actions, resources, and conditions defined for the service in the policy, then a warning banner appears at the top of the page. The service summary then includes details about the problem. To learn how policy summaries help you to understand and troubleshoot the permissions that your policy grants, see [the section called "My policy does not grant the expected permissions" \(p. 576\)](#).
- Next to the **Back** link appears the name of the service (in this case S3). The service summary for this service includes the list of allowed actions that are defined in the policy. If instead, the text **(Explicitly denied)** appears next to the name of a service, then the actions listed in the service summary table are explicitly denied.
- Choose **{ } JSON** to see additional details about the policy. You can do this to view all conditions that are applied to the actions. (If you are viewing the service summary for an inline policy that is attached directly to a user, you must close the service summary dialog box and return to the policy summary to access the JSON policy document.)
- To view the summary for a specific action, type keywords into the search box to reduce the list of available actions.
- Action (2 of 69 actions)** – This column lists the actions that are defined within the policy and provides the resources and conditions for each action. If the policy grants permissions to the action, then the action name links to the [action summary \(p. 506\)](#) table. The count indicates the number of recognized actions that provide permissions. The total is the number of known actions for the service. In this example, 2 actions provide permissions out of 69 total known S3 actions.
- Show/Hide remaining 67** – Choose this link to expand or hide the table to include actions that are known but do not provide permissions for this service. Expanding the link also displays warnings for any elements that do not provide permissions.

7. Unrecognized resource types – This policy includes at least one unrecognized resource type within the policy for this service. You can use this warning to check whether a resource type might include a typo. If the resource type is correct, then the service might not fully support policy summaries, might be in preview, or might be a custom service. To request policy summary support for a specific resource type in a generally available (GA) service, see [Service does not support IAM policy summaries \(p. 575\)](#). In this example, the autoscaling service name is missing an a.

8. Unrecognized actions – This policy includes at least one unrecognized action within the policy for this service. You can use this warning to check whether an action might include a typo. If the action name is correct, then the service might not fully support policy summaries, might be in preview, or might be a custom service. To request policy summary support for a specific action in a generally available (GA) service, see [Service does not support IAM policy summaries \(p. 575\)](#). In this example,

the DeleteObject  action is missing an e.

Note

IAM reviews service names, actions, and resource types for services that support policy summaries. However, your policy summary might include a resource value or condition that does not exist. Always test your policies with the [policy simulator \(p. 445\)](#).

9. For those actions that IAM recognizes, the table groups these actions into at least one or up to four sections, depending on the level of access that the policy allows or denies. The sections are **List**, **Read**, **Write**, and **Permissions management**. You can also see the number of actions that are defined out of the total number of actions available within each access level. To view the access level classification that is assigned to each action in a service, see [Actions, Resources, and Condition Keys for AWS Services](#).

10. The ellipses (...) indicate that all the actions are included in the page, but we are showing only the rows with information relevant to this policy. When you view this page in the AWS Management Console, you see all the actions for your service.

11 (No access) – This policy includes an action that does not provide permissions.

12 Actions that provide permissions include a link to the action summary.

13 Resource – This column shows the resources that the policy defines for the service. IAM does not check whether the resource applies to each action. In this example, actions in the S3 service are allowed on only the developer_bucket Amazon S3 bucket resource. Depending on the information that the service provides to IAM, you might see an ARN such as arn:aws:s3:::developer_bucket/*, or you might see the defined resource type, such as BucketName = developer_bucket.

Note

This column can include a resource from a different service. If the policy statement that includes the resource does not include both actions and resources from the same service, then your policy includes mismatched resources. IAM does not warn you about mismatched resources when you create a policy, or when you view a policy in the service summary.

IAM also does not indicate whether the action applies to the resources, only whether the service matches. If this column includes a mismatched resource, then you should review your policy for errors. To better understand your policies, always test them with the [policy simulator \(p. 445\)](#).

14 Resource warning – For actions with resources that do not provide full permissions, you see one of the following warnings:

- **This action does not support resource-level permissions. This requires a wildcard (*) for the resource.** – This means that the policy includes resource-level permissions but must include "Resource": ["*"] to provide permissions for this action.
- **This action does not have an applicable resource.** – This means that the action is included in the policy without a supported resource.
- **This action does not have an applicable resource and condition.** – This means that the action is included in the policy without a supported resource and without a supported condition. In this case, there is also condition included in the policy for this service, but there are no conditions that apply to this action.

For the `ListAllMyBuckets` action, this policy includes the last warning because the action does not support resource-level permissions and does not support the `s3:x-amz-acl` condition key. If you fix either the resource problem or the condition problem, the remaining issue appears in a detailed warning.

15**Request condition** – This column tells whether the actions associated with the resource are subject to conditions. To learn more about those conditions, choose `{ } JSON` to review the JSON policy document.

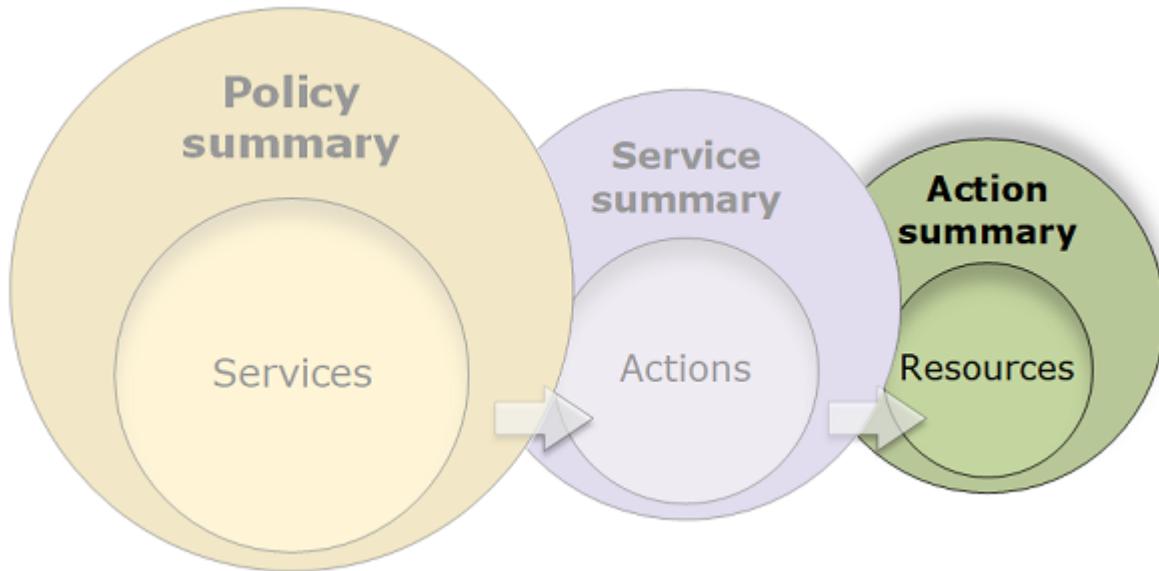
16**Condition warning** – For actions with conditions that do not provide full permissions, you see one of the following warnings:

- **<CONDITION_KEY> is not a supported condition key for this action.** – This means that the policy includes a condition key for the service that is not supported for this action.
- **Multiple condition keys are not supported for this action.** – This means that the policy includes more than one condition keys for the service that are not supported for this action.

For `GetObject`, this policy includes the `s3:x-amz-acl` condition key, which will not work with this action. Although the action supports the resource, the policy does not grant any permissions for this action because the condition will never be true for this action.

Action summary (list of resources)

Policies are summarized in three tables: the [policy summary \(p. 491\)](#), the [service summary \(p. 501\)](#), and the action summary. The *action summary* table includes a list of resources and the associated conditions that apply to the chosen action.



To view an action summary for each action that grants permissions, choose the link in the service summary. The action summary table includes details about the resource, including its **Region** and **Account**. You can also view the conditions that apply to each resource. This shows you conditions that apply to some resources but not others.

Viewing action summaries

You can view the action summary for any policy that is attached to a user on the **Users** page. You can view the action summary for any policy that is attached to a role on the **Roles** page. You can view the action summary for managed policies on the **Policies** page. However, if you try to view the action

summary for a managed policy from the **Users** page or the **Roles** page, you are redirected to the **Policies** page.

To view the action summary for a managed 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, choose **Policies**.
3. In the list of policies, choose the name of the policy that you want to view.
4. On the **Summary** page for the policy, view the **Permissions** tab to see the policy summary.
5. In the policy summary list of services, choose the name of the service that you want to view.
6. In the service summary list of actions, choose the name of the action that you want to view.

To view the action summary for a policy attached to a user

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. Choose **Users** from the navigation pane.
3. In the list of users, choose the name of the user whose policy you want to view.
4. On the **Summary** page for the user, view the **Permissions** tab to see the list of policies that are attached to the user directly or from a group.
5. In the table of policies for the user, expand the row of the policy that you want to view.
6. In the policy summary list of services, choose the name of the service that you want to view.

Note

If the policy that you select is an inline policy that is attached directly to the user, then the service summary table appears. If the policy is an inline policy attached from a group, then you are taken to the JSON policy document for that group. If the policy is a managed policy, then you are taken to the service summary for that policy on the **Policies** page.

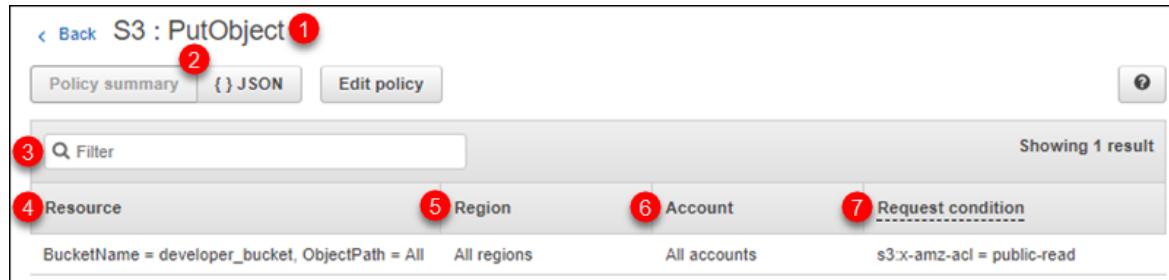
7. In the service summary list of actions, choose the name of the action that you want to view.

To view the action summary for a policy attached to a role

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, choose **Roles**.
3. In the list of roles, choose the name of the role whose policy you want to view.
4. On the **Summary** page for the role, view the **Permissions** tab to see the list of policies that are attached to the role.
5. In the table of policies for the role, expand the row of the policy that you want to view.
6. In the policy summary list of services, choose the name of the service that you want to view.
7. In the service summary list of actions, choose the name of the action that you want to view.

Understanding the elements of an action summary

The example below is the action summary for the `PutObject` (Write) action from the Amazon S3 service summary (see [Service summary \(list of actions\) \(p. 501\)](#)). For this action, the policy defines multiple conditions on a single resource.



The action summary page includes the following information:

1. Next to the **Back** link appears the name of the service and action in the format **service: action** (in this case **S3: PutObject**). The action summary for this service includes the list of resources that are defined in the policy.
2. Choose **{ } JSON** to see additional details about the policy, such as viewing the multiple conditions that are applied to the actions. (If you are viewing the action summary for an inline policy that is attached directly to a user, the steps differ. To access the JSON policy document in that case, you must close the action summary dialog box and return to the policy summary.)
3. To view the summary for a specific resource, type keywords into the search box to reduce the list of available resources.
4. **Resource** – This column lists the resources that the policy defines for the chosen service. In this example, the **PutObject** action is allowed on all object paths, but on only the developer_bucket Amazon S3 bucket resource. Depending on the information that the service provides to IAM, you might see an ARN such as `arn:aws:s3:::developer_bucket/*`, or you might see the defined resource type, such as `BucketName = developer_bucket, ObjectPath = All`.
5. **Region** – This column shows the Region in which the resource is defined. Resources can be defined for all Regions, or a single Region. They cannot exist in more than one specific Region.
 - **All Regions** – The actions that are associated with the resource apply to all Regions. In this example, the action belongs to a global service, Amazon S3. Actions that belong to global services apply to all Regions.
 - Region text – The actions associated with the resource apply to one Region. For example, a policy can specify the `us-east-2` Region for a resource.
6. **Account** – This column indicates whether the services or actions associated with the resource apply to a specific account. Resources can exist in all accounts or a single account. They cannot exist in more than one specific account.
 - **All accounts** – The actions that are associated with the resource apply to all accounts. In this example, the action belongs to a global service, Amazon S3. Actions that belong to global services apply to all accounts.
 - **This account** – The actions that are associated with the resource apply only to the account that you are currently logged in to.
 - Account number – The actions that are associated with the resource apply to one account (one that you are not currently logged in to). For example, if a policy specifies the `123456789012` account for a resource, then the account number appears in the policy summary.
7. **Request condition** – This column shows whether the actions that are associated with the resource are subject to conditions. This example includes the `s3:x-amz-acl = public-read` condition. To learn more about those conditions, choose **{ } JSON** to review the JSON policy document.

Examples of policy summaries

The following examples include JSON policies with their associated [policy summaries \(p. 491\)](#), the [service summaries \(p. 501\)](#), and the [action summaries \(p. 506\)](#) to help you understand the permissions given through a policy.

Policy 1: DenyCustomerBucket

This policy demonstrates an allow and a deny for the same service.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "FullAccess",
            "Effect": "Allow",
            "Action": ["s3:*"],
            "Resource": ["*"]
        },
        {
            "Sid": "DenyCustomerBucket",
            "Action": ["s3:*"],
            "Effect": "Deny",
            "Resource": ["arn:aws:s3:::customer", "arn:aws:s3:::customer/*"]
        }
    ]
}
```

DenyCustomerBucket Policy Summary:

This policy defines some actions, resources, or conditions that do not provide permissions. To grant access, policies must have an action that has an applicable resource or condition. For details, choose Show remaining. [Learn more](#)

Policy summary	{ } JSON	Edit policy	Simulate policy																				
<div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <input style="width: 100%; height: 15px; border: none; border-bottom: 1px solid #ccc; font-size: 12px; padding-left: 5px;" type="text"/> </div> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 25%; padding: 5px;">Service</th> <th style="width: 25%; padding: 5px;">Access level</th> <th style="width: 25%; padding: 5px;">Resource</th> <th style="width: 25%; padding: 5px;">Request condition</th> </tr> </thead> <tbody> <tr> <td colspan="4" style="padding: 5px; text-align: center; font-weight: bold;">Explicit deny (1 of 103 services)</td> </tr> <tr> <td style="padding: 5px;">S3</td> <td style="padding: 5px;">Full: Read, Write, Permissions management Limited: List</td> <td style="padding: 5px;">Multiple</td> <td style="padding: 5px;">None</td> </tr> <tr> <td colspan="4" style="padding: 5px; text-align: center; font-weight: bold;">Allow (1 of 103 services) Show remaining 102</td> </tr> <tr> <td style="padding: 5px;">S3</td> <td style="padding: 5px;">Full access</td> <td style="padding: 5px;">All resources</td> <td style="padding: 5px;">None</td> </tr> </tbody> </table>				Service	Access level	Resource	Request condition	Explicit deny (1 of 103 services)				S3	Full: Read, Write, Permissions management Limited: List	Multiple	None	Allow (1 of 103 services) Show remaining 102				S3	Full access	All resources	None
Service	Access level	Resource	Request condition																				
Explicit deny (1 of 103 services)																							
S3	Full: Read, Write, Permissions management Limited: List	Multiple	None																				
Allow (1 of 103 services) Show remaining 102																							
S3	Full access	All resources	None																				

DenyCustomerBucket S3 (Explicit deny) Service Summary:

Action (66 of 69) Hide remaining 3	Resource	Request condition
List (1 of 4 actions)		
HeadBucket (No access)	⚠ This action does not support resource-level permissions. This requires a wildcard (*) for the resource.	None
ListAllMyBuckets(No access)	⚠ This action does not support resource-level permissions. This requires a wildcard (*) for the resource.	None
ListBucket	BucketName = customer	None
ListObjects (No access)	⚠ This action does not support resource-level permissions. This requires a wildcard (*) for the resource.	None
Read (30 of 30 actions)		
GetAccelerateConfiguration	BucketName = customer	None
GetAnalyticsConfiguration	BucketName = customer	None
GetBucketAcl	BucketName = customer	None
GetBucketCORS	BucketName = customer	None
GetBucketLocation	BucketName = customer	None
GetBucketLogging	BucketName = customer	None
GetBucketNotification	BucketName = customer	None
GetBucketPolicy	BucketName = customer	None
GetBucketRequestPayment	BucketName = customer	None
GetBucketTagging	BucketName = customer	None
GetBucketVersioning	BucketName = customer	None
GetBucketWebsite	BucketName = customer	None
GetInventoryConfiguration	BucketName = customer	None
GetIpConfiguration	BucketName = customer	None
GetLifecycleConfiguration	BucketName = customer	None
GetMetricsConfiguration	BucketName = customer	None
GetObject	BucketName = customer, ObjectPath = All	None
GetObjectAcl	BucketName = customer, ObjectPath = All	None
GetObjectTagging	BucketName = customer, ObjectPath = All	None
GetObjectTorrent	BucketName = customer, ObjectPath = All	None
GetObjectVersion	BucketName = customer, ObjectPath = All	None

GetObject (Read) Action Summary:

Resource	Region	Account	Request condition
BucketName = customer, ObjectPath = All	All regions	All accounts	None

Policy 2: DynamoDbRowCognitoID

This policy provides row-level access to Amazon DynamoDB based on the user's Amazon Cognito ID.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb>DeleteItem",
                "dynamodb>GetItem",
                "dynamodb>PutItem",
                "dynamodb>UpdateItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-west-1:123456789012:table/myDynamoTable"
            ],
            "Condition": {
                "ForAllValues:StringEquals": {
                    "dynamodb:LeadingKeys": [
                        "${cognito-identity.amazonaws.com:sub}"
                    ]
                }
            }
        }
    ]
}
```

DynamoDbRowCognitoID Policy Summary:

Service	Access level	Resource	Request condition
Allow (1 of 102 services) Show remaining 101			
DynamoDB	Limited: Read, Write	TableName = myDynamoTable	dynamodb:LeadingKeys = \${cognito-identity.amazonaws.com:sub}

DynamoDbRowCognitoID DynamoDB (Allow) Service Summary:

Action (4 of 25) Show remaining 21	Resource	Request condition
Read (1 of 14 actions)		
GetItem	TableName = myDynamoTable	dynamodb:LeadingKeys = \${cognito-identity.amazonaws.com:sub}
Write (3 of 10 actions)		
DeleteItem	TableName = myDynamoTable	dynamodb:LeadingKeys = \${cognito-identity.amazonaws.com:sub}
PutItem	TableName = myDynamoTable	dynamodb:LeadingKeys = \${cognito-identity.amazonaws.com:sub}
UpdateItem	TableName = myDynamoTable	dynamodb:LeadingKeys = \${cognito-identity.amazonaws.com:sub}

.GetItem (List) Action Summary:

Resource	Region	Account	Request
TableName = myDynamoTable	us-west-1	123456789012	dynamodb:LeadingKeys = \${cognito-identity.amazonaws.com:sub}

Policy 3: MultipleResourceCondition

This policy includes multiple resources and conditions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
            "Resource": ["arn:aws:s3:::Apple_bucket/*"],
            "Condition": {"StringEquals": {"s3:x-amz-acl": ["public-read"]}}
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
            "Resource": ["arn:aws:s3:::Orange_bucket/*"],
            "Condition": {"StringEquals": {
                "s3:x-amz-acl": ["custom"],
                "s3:x-amz-grant-full-control": ["1234"]
            }}
        }
    ]
}
```

MultipleResourceCondition Policy Summary:

Service	Access level	Resource	Request condition
Allow (1 of 100 services) Show remaining 99			
S3	Limited: Write, Permissions management	Multiple	Multiple

MultipleResourceCondition S3 (Allow) Service Summary:

Action (2 of 52 actions)	Show remaining 50	Resource	Request condition
Write (1 of 21 actions)			
PutObject		Multiple	Multiple
Permissions management (1 of 5 actions)			
PutObjectAcl		Multiple	Multiple

PutObject (Write) Action Summary:

Resource	Region	Account	Request condition
BucketName = Orange_bucket, ObjectPath = All	All regions	All accounts	Multiple
BucketName = Apple_bucket, ObjectPath = All	All regions	All accounts	s3:x-amz-acl = public-read

Policy 4: EC2_troubleshoot

The following policy allows users to get a screenshot of a running Amazon EC2 instance, which can help with EC2 troubleshooting. This policy also permits viewing information about the items in the Amazon S3 developer bucket.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:GetConsoleScreenshot"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3>ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::developer"
            ]
        }
    ]
}
```

EC2_Troubleshoot Policy Summary:

Service	Access level	Resource	Request condition
Allow (2 of 102 services) Show remaining 100			
EC2	Limited: Read	All resources	None
S3	Limited: List	BucketName = developer	None

EC2_Troubleshoot S3 (Allow) Service Summary:

Action (1 of 52) Show remaining 51	Resource	Request condition
List (1 of 4 actions)		
ListBucket	BucketName = developer	None

ListBucket (List) Action Summary:

<input type="text"/> Filter			
Resource	Region	Account	Request
BucketName = developer	All regions	All accounts	None

Policy 5: Unrecognized_Service_Action

The following policy was intended to provide full access to DynamoDB, but that access fails because dynamodb is misspelled as dynamobd. This policy was intended to allow access to some Amazon EC2 actions in the us-east-2 Region, but deny that access to the ap-northeast-2 Region. However,

access to reboot instances in the `ap-northeast-2` Region is not explicitly denied because of the unrecognized `o` in the middle of the `RebootInstances` action. This example shows how you can use policy summaries to locate errors in your policies. To learn how to edit policies based on information in a policy summary, see [Editing policies to fix warnings \(p. 492\)](#).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:*"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Action": [
                "ec2:RunInstances",
                "ec2:StartInstances",
                "ec2:StopInstances",
                "ec2:RebootInstances"
            ],
            "Resource": "*",
            "Effect": "Deny",
            "Condition": {
                "StringEquals": {
                    "ec2:Region": "ap-northeast-2"
                }
            }
        },
        {
            "Action": [
                "ec2:RunInstances",
                "ec2:StartInstances",
                "ec2:StopInstances",
                "ec2:RebootInstances"
            ],
            "Resource": "*",
            "Effect": "Allow",
            "Condition": {
                "StringEquals": {
                    "ec2:Region": "us-east-2"
                }
            }
        }
    ]
}
```

Unrecognized_Service_Action Policy Summary:

Service	Access level	Resource	Request condition
<u>Unrecognized services</u>			
dynamodb	⚠		
<u>Explicit deny (1 of 103 services)</u>			
EC2	⚠ Limited: Write	All resources	ec2:Region = ap-northeast-2
<u>Allow (1 of 103 services)</u> Show remaining 102			
EC2	⚠ Limited: Write	All resources	ec2:Region = us-east-2

Unrecognized_Service_Action EC2 (Explicit deny) Service Summary:

Action (3 of 229) Show remaining 226	Resource	Request condition
Unrecognized actions		
RebootInstances ⚠		
Write (3 of 157 actions)		
RunInstances	All resources	ec2:Region = ap-northeast-2
StartInstances	All resources	ec2:Region = ap-northeast-2
StopInstances	All resources	ec2:Region = ap-northeast-2

Unrecognized_Service_Action StartInstances (Write) Action Summary:

Resource	Region	Account	Request condition
All resources	All regions	All accounts	ec2:Region = ap-northeast-2

Policy 6: CodeBuild_CodeCommit_CodeDeploy

This policy provides access to specific CodeBuild, CodeCommit, and CodeDeploy resources. Because these resources are specific to each service, they appear only with the matching service. If you include a resource that does not match any services in the `Action` element, then the resource appears in all action summaries.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1487980617000",
            "Effect": "Allow",
            "Action": [
                "codebuild:*",
                "codecommit:*",
                "codedeploy:*"
            ],
            "Resource": [
                "arn:aws:codebuild:us-east-2:123456789012:project/my-demo-project",
                "arn:aws:codecommit:us-east-2:123456789012:MyDemoRepo",
                "arn:aws:codedeploy:us-east-2:123456789012:application:WordPress_App",
                "arn:aws:codedeploy:us-east-2:123456789012:instance/AssetTag*"
            ]
        }
    ]
}
```

CodeBuild_CodeCommit_CodeDeploy Policy Summary:

Service	Access level	Resource	Request condition
Allow (3 of 103 services) Show remaining 100			
CodeBuild	Limited: List, Read, Write	arn:aws:codebuild:us-east-2:123456789012:project/my-demo-project	None
CodeCommit	Full: Read, Write Limited: List	arn:aws:codecommit:us-east-2:123456789012:MyDemoRepo	None
CodeDeploy	Limited: List, Read, Write	Multiple	None

CodeBuild_CodeCommit_CodeDeploy CodeBuild (Allow) Service Summary:

Action (9 of 15) Show remaining 6	Resource	Request condition
List (1 of 3 actions)		
ListBuildsForProject	arn:aws:codebuild:us-east-2:123456789012:project/my-demo-project	None
Read (2 of 5 actions)		
BatchGetBuilds	arn:aws:codebuild:us-east-2:123456789012:project/my-demo-project	None
BatchGetProjects	arn:aws:codebuild:us-east-2:123456789012:project/my-demo-project	None
Write (6 of 7 actions)		
BatchDeleteBuilds	arn:aws:codebuild:us-east-2:123456789012:project/my-demo-project	None
CreateProject	arn:aws:codebuild:us-east-2:123456789012:project/my-demo-project	None
DeleteProject	arn:aws:codebuild:us-east-2:123456789012:project/my-demo-project	None
StartBuild	arn:aws:codebuild:us-east-2:123456789012:project/my-demo-project	None
StopBuild	arn:aws:codebuild:us-east-2:123456789012:project/my-demo-project	None
UpdateProject	arn:aws:codebuild:us-east-2:123456789012:project/my-demo-project	None

CodeBuild_CodeCommit_CodeDeploy StartBuild (Write) Action Summary:

Resource	Region	Account	Request condition
arn:aws:codebuild:us-east-2:123456789012:project/my-demo-project	us-east-2	123456789012	None

Permissions required to access IAM resources

Resources are objects within a service. IAM resources include groups, users, roles, and policies. If you are signed in with AWS account root user credentials, you have no restrictions on administering IAM.

credentials or IAM resources. However, IAM users must explicitly be given permissions to administer credentials or IAM resources. You can do this by attaching an identity-based policy to the user.

Note

Throughout the AWS documentation, when we refer to an IAM policy without mentioning any of the specific categories, we mean an identity-based, customer managed policy. For details about policy categories, see [the section called "Policies and permissions" \(p. 351\)](#).

Permissions for administering IAM identities

The permissions that are required to administer IAM groups, users, roles, and credentials usually correspond to the API actions for the task. For example, in order to create IAM users, you must have the `iam:CreateUser` permission that has the corresponding API command: `CreateUser`. To allow an IAM user to create other IAM users, you could attach an IAM policy like the following one to that user:

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

In a policy, the value of the `Resource` element depends on the action and what resources the action can affect. In the preceding example, the policy allows a user to create any user (* is a wildcard that matches all strings). In contrast, a policy that allows users to change only their own access keys (API actions `CreateAccessKey` and `UpdateAccessKey`) typically has a `Resource` element. In this case the ARN includes a variable (`#{aws:username}`) that resolves to the current user's name, as in the following example:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "ListUsersForConsole",  
            "Effect": "Allow",  
            "Action": "iam>ListUsers",  
            "Resource": "arn:aws:iam::*:/*"  
        },  
        {  
            "Sid": "ViewAndUpdateAccessKeys",  
            "Effect": "Allow",  
            "Action": [  
                "iam:UpdateAccessKey",  
                "iam>CreateAccessKey",  
                "iam>ListAccessKeys"  
            ],  
            "Resource": "arn:aws:iam:::user/#{aws:username}"  
        }  
    ]  
}
```

In the previous example, `#{aws:username}` is a variable that resolves to the user name of the current user. For more information about policy variables, see [IAM policy elements: Variables and tags \(p. 658\)](#).

Using a wildcard character (*) in the action name often makes it easier to grant permissions for all the actions related to a specific task. For example, to allow users to perform any IAM action, you can use `iam:*` for the action. To allow users to perform any action related just to access keys, you can use `iam:AccessKey*` in the `Action` element of a policy statement. This gives the user permission to

perform the [CreateAccessKey](#), [DeleteAccessKey](#), [GetAccessKeyLastUsed](#), [ListAccessKeys](#), and [UpdateAccessKey](#) actions. (If an action is added to IAM in the future that has "AccessKey" in the name, using `iam:*AccessKey*` for the Action element will also give the user permission to that new action.) The following example shows a policy that allows users to perform all actions pertaining to their own access keys (replace ACCOUNT-ID-WITHOUT-HYPHENS with your AWS account ID):

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

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

Permissions for working in the AWS Management Console

The preceding examples show policies that allow a user to perform the actions with the [AWS CLI](#) or the [AWS SDKs](#).

As users work with the console, the console issues requests to IAM to list groups, users, roles, and policies, and to get the policies associated with a group, user, or role. The console also issues requests to get AWS account information and information about the principal. The principal is the user making requests in the console.

In general, to perform an action, you must have only the matching action included in a policy. To create a user, you need permission to call the `CreateUser` action. Often, when you use the console to perform an action, you must have permissions to display, list, get, or otherwise view resources in the console. This is necessary so that you can navigate through the console to make the specified action. For example, if user Jorge wants to use the console to change his own access keys, he goes to the IAM console and chooses **Users**. This action causes the console to make a `ListUsers` request. If Jorge doesn't have permission for the `iam>ListUsers` action, the console is denied access when it tries to list users. As a result, Jorge can't get to his own name and to his own access keys, even if he has permissions for the `CreateAccessKey` and `UpdateAccessKey` actions.

If you want to give users permissions to administer groups, users, roles, policies, and credentials with the AWS Management Console, you need to include permissions for the actions that the console performs. For some examples of policies that you can use to grant a user for these permissions, see [Example policies for administering IAM resources \(p. 520\)](#).

Granting permissions across AWS accounts

You can directly grant IAM users in your own account access to your resources. If users from another account need access to your resources, you can create an IAM role, which is an entity that includes permissions but that isn't associated with a specific user. Users from other accounts can then use the role and access resources according to the permissions you've assigned to the role. For more information, see [Providing access to an IAM user in another AWS account that you own \(p. 171\)](#).

Note

Some services support resource-based policies as described in [Identity-based policies and resource-based policies \(p. 374\)](#) (such as Amazon S3, Amazon SNS, and Amazon SQS). For

those services, an alternative to using roles is to attach a policy to the resource (bucket, topic, or queue) that you want to share. The resource-based policy can specify the AWS account that has permissions to access the resource.

Permissions for one service to access another

Many AWS services access other AWS services. For example, several AWS services—including Amazon EMR, Elastic Load Balancing, and Amazon EC2 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 Amazon EC2 Auto Scaling, users must have permission to use Auto Scaling, but don't need to be explicitly granted permission to manage Amazon EC2 instances.
- In AWS Data Pipeline, an IAM role determines what a pipeline can do; users need permission to assume the role. (For details, see [Granting Permissions to Pipelines with IAM](#) in the *AWS Data Pipeline Developer Guide*.)

For details about how to configure permissions properly so that an AWS service is able to accomplish the tasks you intend, refer to the documentation for the service you are calling. To learn how to create a role for a service, see [Creating a role to delegate permissions to an AWS service \(p. 229\)](#).

Configuring a service with an IAM role to work on your behalf

When you want to configure an AWS service to work on your behalf, you typically provide the ARN for an IAM role that defines what the service is allowed to do. AWS checks to ensure that you have permissions to pass a role to a service. For more information, see [Granting a user permissions to pass a role to an AWS service \(p. 251\)](#).

Required actions

Actions are the things that you can do to a resource, such as viewing, creating, editing, and deleting that resource. Actions are defined by each AWS service.

To allow someone to perform an action, you must include the necessary actions in a policy that applies to the calling identity or the affected resource. In general, to provide the permission required to perform an action, you must include that action in your policy. For example, to create a user, you need add the `CreateUser` action to your policy.

In some cases, an action might require that you include additional related actions in your policy. For example, to provide permission for someone to create a directory in AWS Directory Service using the `ds:CreateDirectory` operation, you must include the following actions in their policy:

- `ds:CreateDirectory`
- `ec2:DescribeSubnets`
- `ec2:DescribeVpcs`
- `ec2:CreateSecurityGroup`
- `ec2:CreateNetworkInterface`
- `ec2:DescribeNetworkInterfaces`
- `ec2:AuthorizeSecurityGroupIngress`
- `ec2:AuthorizeSecurityGroupEgress`

When you create or edit a policy using the visual editor, you receive warnings and prompts to help you choose all of the required actions for your policy.

For more information about the permissions required to create a directory in AWS Directory Service, see [Example 2: Allow a User to Create a Directory](#).

Example policies for administering IAM resources

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

For examples of policies that let users perform tasks with other AWS services, like Amazon S3, Amazon EC2, and DynamoDB, see [Example IAM identity-based policies \(p. 389\)](#).

Topics

- [Allow a user to list the account's groups, users, policies, and more for reporting purposes \(p. 520\)](#)
- [Allow a user to manage a group's membership \(p. 520\)](#)
- [Allow a user to manage IAM users \(p. 520\)](#)
- [Allow users to set account password policy \(p. 521\)](#)
- [Allow users to generate and retrieve IAM credential reports \(p. 521\)](#)
- [Allow all IAM actions \(admin access\) \(p. 522\)](#)

Allow a user to list the account's groups, users, policies, and more for reporting purposes

The following policy allows the user to call any IAM action that starts with the string `Get` or `List`, and to generate reports. To view the example policy, see [IAM: Allows read-only access to the IAM console \(p. 423\)](#).

Allow a user to manage a group's membership

The following policy allows the user to update the membership of the group called `MarketingGroup`. To view the example policy, see [IAM: Allows managing a group's membership programmatically and in the console \(p. 421\)](#).

Allow a user to manage IAM users

The following policy allows a user to perform all the tasks associated with managing IAM users but not to perform actions on other entities, such as creating groups or policies. Allowed actions include these:

- Creating the user (the `CreateUser` action).
- Deleting the user. This task requires permissions to perform all of the following actions: `DeleteSigningCertificate`, `DeleteLoginProfile`, `RemoveUserFromGroup`, and `DeleteUser`.
- Listing users in the account and in groups (the `GetUser`, `ListUsers` and `ListGroupsForUser` actions).
- Listing and removing policies for the user (the `ListUserPolicies`, `ListAttachedUserPolicies`, `DetachUserPolicy`, `DeleteUserPolicy` actions)
- Renaming or changing the path for the user (the `UpdateUser` action). The `Resource` element must include an ARN that covers both the source path and the target path. For more information on paths, see [Friendly names and paths \(p. 600\)](#).

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

```
{  
    "Sid": "AllowUsersToPerformUserActions",  
    "Effect": "Allow",  
    "Action": [  
        "iam>ListPolicies",  
        "iam:GetPolicy",  
        "iam>UpdateUser",  
        "iam>AttachUserPolicy",  
        "iam>ListEntitiesForPolicy",  
        "iam>DeleteUserPolicy",  
        "iam>DeleteUser",  
        "iam>ListUserPolicies",  
        "iam>CreateUser",  
        "iam>RemoveUserFromGroup",  
        "iam>AddUserToGroup",  
        "iam GetUserPolicy",  
        "iam>ListGroupsForUser",  
        "iam>PutUserPolicy",  
        "iam>ListAttachedUserPolicies",  
        "iam>ListUsers",  
        "iam GetUser",  
        "iam>DetachUserPolicy"  
    ],  
    "Resource": "*"  
},  
{  
    "Sid": "AllowUsersToSeeStatsOnIAMConsoleDashboard",  
    "Effect": "Allow",  
    "Action": [  
        "iam>GetAccount*",  
        "iam>ListAccount*"  
    ],  
    "Resource": "*"  
}  
]
```

A number of the permissions included in the preceding policy allow the user to perform tasks in the AWS Management Console. Users who perform user-related tasks from the [AWS CLI](#), the [AWS SDKs](#), or the IAM HTTP query API only might not need certain permissions. For example, if users already know the ARN of policies to detach from a user, they do not need the `iam>ListAttachedUserPolicies` permission. The exact list of permissions that a user requires depends on the tasks that the user must perform while managing other users.

The following permissions in the policy allow access to user tasks via the AWS Management Console:

- `iam>GetAccount*`
- `iam>ListAccount*`

Allow users to set account password policy

You might give some users permissions to get and update your AWS account's [password policy \(p. 93\)](#). To view the example policy, see [IAM: Allows setting the account password requirements programmatically and in the console \(p. 424\)](#).

Allow users to generate and retrieve IAM credential reports

You can give users permission to generate and download a report that lists all users in your AWS account. The report also lists the status of various user credentials, including passwords, access keys, MFA devices, and signing certificates. For more information about credential reports, see [Getting credential reports](#)

for your AWS account (p. 148). To view the example policy, see [IAM: Generate and retrieve IAM credential reports \(p. 420\)](#).

Allow all IAM actions (admin access)

You might give some users administrative permissions to perform all actions in IAM, including managing passwords, access keys, MFA devices, and user certificates. The following example policy grants these permissions.

Warning

When you give a user full access to IAM, there is no limit to the permissions that user can grant to him/herself or others. The user can create new IAM entities (users or roles) and grant those entities full access to all resources in your AWS account. When you give a user full access to IAM, you are effectively giving them full access to all resources in your AWS account. This includes access to delete all resources. You should grant these permissions to only trusted administrators, and you should enforce multi-factor authentication (MFA) for these administrators.

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

Security in IAM and AWS STS

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from data centers and network architectures that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The [shared responsibility model](#) describes this as security of the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the [AWS Compliance Programs](#). To learn about the compliance programs that apply to AWS Identity and Access Management (IAM), see [AWS Services in Scope by Compliance Program](#).
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using AWS Identity and Access Management (IAM) and AWS Security Token Service (AWS STS). The following topics show you how to configure IAM and AWS STS to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your IAM resources.

Contents

- [Data protection in AWS Identity and Access Management \(p. 523\)](#)
- [Logging and monitoring in AWS Identity and Access Management \(p. 524\)](#)
- [Compliance validation for AWS Identity and Access Management \(p. 525\)](#)
- [Resilience in AWS Identity and Access Management \(p. 526\)](#)
- [Infrastructure security in AWS Identity and Access Management \(p. 526\)](#)
- [Configuration and vulnerability analysis in AWS Identity and Access Management \(p. 526\)](#)
- [Security best practices and use cases in AWS Identity and Access Management \(p. 527\)](#)

Data protection in AWS Identity and Access Management

The AWS [shared responsibility model](#) applies to data protection in AWS Identity and Access Management. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. This content includes the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the [Data Privacy FAQ](#). For information about data protection in Europe, see the [AWS Shared Responsibility Model and GDPR](#) blog post on the [AWS Security Blog](#).

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management (IAM). That way each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We recommend TLS 1.2 or later.

- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon S3.
- If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see [Federal Information Processing Standard \(FIPS\) 140-2](#).

We strongly recommend that you never put sensitive identifying information, such as your customers' account numbers, into free-form fields such as a **Name** field. This includes when you work with IAM or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into IAM or other services might get picked up for inclusion in diagnostic logs. When you provide a URL to an external server, don't include credentials information in the URL to validate your request to that server.

Data encryption in IAM and AWS STS

Data encryption typically falls into two categories: encryption at rest and encryption in transit.

Encryption at rest

Data that is collected and stored by IAM is encrypted at rest.

- **IAM** – Data collected and stored within IAM includes IP addresses, customer account metadata, and customer identifying data that includes passwords. Customer account metadata and customer identifying data are encrypted at rest using AES 256 or is hashed using SHA 256.
- **AWS STS** – AWS STS does not collect customer content except for service logs that log successful, erroneous, and faulty requests to the service.

Encryption in transit

Customer identifying data, including passwords, is encrypted in transit using TLS 1.1 and 1.2. All AWS STS endpoints support HTTPS for encrypting data in transit. For a list of AWS STS endpoints, see [Regions and endpoints \(p. 329\)](#).

Key management in IAM and AWS STS

You can't manage encryption keys using IAM or AWS STS. For more information about encryption keys, see [What is AWS KMS?](#) in the AWS Key Management Service Developer Guide

Internet traffic privacy in IAM and AWS STS

Requests to IAM must be made using Transport Layer Security protocol (TLS). You can secure connections to the AWS STS service by using VPC endpoints. To learn more, see [Using AWS STS interface VPC endpoints \(p. 330\)](#).

Logging and monitoring in AWS Identity and Access Management

Monitoring is an important part of maintaining the reliability, availability, and performance of AWS Identity and Access Management (IAM), AWS Security Token Service (AWS STS) and your other AWS

solutions. AWS provides several tools for monitoring your AWS resources and responding to potential incidents:

- *AWS CloudTrail* captures all API calls for IAM and AWS STS as events, including calls from the console and API calls. To learn more about using CloudTrail with IAM and AWS STS, see [Logging IAM and AWS STS API calls with AWS CloudTrail \(p. 336\)](#). For more information about CloudTrail, see the [AWS CloudTrail User Guide](#).
- *AWS Identity and Access Management Access Analyzer* helps you identify the resources in your organization and accounts, such as Amazon S3 buckets or IAM roles, that are shared with an external entity. This helps you identify unintended access to your resources and data, which is a security risk. To learn more, see [What is IAM Access Analyzer?](#)
- *Amazon CloudWatch* monitors your AWS resources and the applications that you run on AWS in real time. You can collect and track metrics, create customized dashboards, and set alarms that notify you or take actions when a specified metric reaches a threshold that you specify. For example, you can have CloudWatch track CPU usage or other metrics of your Amazon EC2 instances and automatically launch new instances when needed. For more information, see the [Amazon CloudWatch User Guide](#).
- *Amazon CloudWatch Logs* helps you monitor, store, and access your log files from Amazon EC2 instances, CloudTrail, and other sources. CloudWatch Logs can monitor information in the log files and notify you when certain thresholds are met. You can also archive your log data in highly durable storage. For more information, see the [Amazon CloudWatch Logs User Guide](#).

For additional resources and security best practices for IAM, see [Security best practices and use cases in AWS Identity and Access Management \(p. 527\)](#).

Compliance validation for AWS Identity and Access Management

Third-party auditors assess the security and compliance of AWS Identity and Access Management (IAM) as part of multiple AWS compliance programs. These include SOC, PCI, FedRAMP, ISO, and others.

For a list of AWS services in scope of specific compliance programs, see [AWS Services in Scope by Compliance Program](#). For general information, see [AWS Compliance Programs](#).

You can download third-party audit reports using AWS Artifact. For more information, see [Downloading Reports in AWS Artifact](#).

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- [Security and Compliance Quick Start Guides](#) – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- [Architecting for HIPAA Security and Compliance Whitepaper](#) – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.

Note

Not all services are compliant with HIPAA.

- [AWS Compliance Resources](#) – This collection of workbooks and guides might apply to your industry and location.
- [Evaluating Resources with Rules](#) in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.

- [AWS Security Hub](#) – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

Resilience in AWS Identity and Access Management

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see [AWS Global Infrastructure](#).

AWS Identity and Access Management (IAM) and AWS Security Token Service (AWS STS) are available globally. By default, AWS STS requests go to a single global endpoint. But you can choose to use a Regional AWS STS endpoint to reduce latency or provide additional redundancy for your applications. To learn more, see [Managing AWS STS in an AWS Region \(p. 327\)](#).

Infrastructure security in AWS Identity and Access Management

As managed services, AWS Identity and Access Management (IAM) and AWS Security Token Service (AWS STS) are protected by the AWS global network security procedures that are described in the [Amazon Web Services: Overview of Security Processes](#) whitepaper.

You use AWS published API calls to access IAM through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use AWS STS to generate temporary security credentials to sign requests.

IAM can be accessed programmatically by using the IAM HTTPS API, which lets you issue HTTPS requests directly to the service. The Query API returns sensitive information, including security credentials. Therefore, you must use HTTPS with all API requests. When you use the HTTPS API, you must include code to digitally sign requests using your credentials.

You can call these API operations from any network location, but IAM does support resource-based access policies, which can include restrictions based on the source IP address. You can also use IAM policies to control access from specific Amazon Virtual Private Cloud (Amazon VPC) endpoints or specific VPCs. Effectively, this isolates network access to a given IAM resource from only the specific VPC within the AWS network.

Configuration and vulnerability analysis in AWS Identity and Access Management

AWS handles basic security tasks like guest operating system (OS) and database patching, firewall configuration, and disaster recovery. These procedures have been reviewed and certified by the appropriate third parties. For more details, see the following resources:

- [Shared Responsibility Model](#)
- [Amazon Web Services: Overview of Security Processes](#) (whitepaper)

The following resources also address configuration and vulnerability analysis in AWS Identity and Access Management (IAM):

- [Compliance validation for AWS Identity and Access Management \(p. 525\)](#)
- [Security best practices and use cases in AWS Identity and Access Management \(p. 527\)](#)

Security best practices and use cases in AWS Identity and Access Management

AWS Identity and Access Management (IAM) provides a number of security features to consider as you develop and implement your own security policies. The following best practices are general guidelines and don't represent a complete security solution. Because these best practices might not be appropriate or sufficient for your environment, treat them as helpful considerations rather than prescriptions.

To get the greatest benefits from IAM, take time to learn the recommended best practices. One way to do this is to see how IAM is used in real-world scenarios to work with other AWS services.

Topics

- [Security best practices in IAM \(p. 527\)](#)
- [Business use cases for IAM \(p. 534\)](#)

Security best practices in IAM

 [Follow us on Twitter](#)

To help secure your AWS resources, follow these recommendations for the AWS Identity and Access Management (IAM) service.

Topics

- [Lock away your AWS account root user access keys \(p. 528\)](#)
- [Create individual IAM users \(p. 528\)](#)
- [Use groups to assign permissions to IAM users \(p. 528\)](#)
- [Grant least privilege \(p. 529\)](#)
- [Get started using permissions with AWS managed policies \(p. 529\)](#)
- [Use customer managed policies instead of inline policies \(p. 530\)](#)
- [Use access levels to review IAM permissions \(p. 531\)](#)
- [Configure a strong password policy for your users \(p. 531\)](#)
- [Enable MFA \(p. 531\)](#)
- [Use roles for applications that run on Amazon EC2 instances \(p. 532\)](#)
- [Use roles to delegate permissions \(p. 532\)](#)
- [Do not share access keys \(p. 532\)](#)
- [Rotate credentials regularly \(p. 532\)](#)
- [Remove unnecessary credentials \(p. 533\)](#)
- [Use policy conditions for extra security \(p. 533\)](#)

- [Monitor activity in your AWS account \(p. 533\)](#)
- [Video presentation about IAM best practices \(p. 534\)](#)

Lock away your AWS account root user access keys

You use an access key (an access key ID and secret access key) to make programmatic requests to AWS. However, do not use your AWS account root user access key. The access key for your AWS account root user gives full access to all your resources for all AWS services, including your billing information. You cannot reduce the permissions associated with your AWS account root user access key.

Therefore, protect your root user access key like you would your credit card numbers or any other sensitive secret. Here are some ways to do that:

- If you don't already have an access key for your AWS account root user, don't create one unless you absolutely need to. Instead, use your account email address and password to sign in to the AWS Management Console and [create an IAM user for yourself \(p. 20\)](#) that has administrative permissions.
- If you do have an access key for your AWS account root user, delete it. If you must keep it, rotate (change) the access key regularly. To delete or rotate your root user access keys, go to the [My Security Credentials page](#) in the AWS Management Console and sign in with your account's email address and password. You can manage your access keys in the **Access keys** section. For more information about rotating access keys, see [Rotating access keys \(p. 106\)](#).
- Never share your AWS account root user password or access keys with anyone. The remaining sections of this document discuss various ways to avoid having to share your AWS account root user credentials with other users. They also explain how to avoid having to embed them in an application.
- Use a strong password to help protect account-level access to the AWS Management Console. For information about managing your AWS account root user password, see [Changing the AWS account root user password \(p. 92\)](#).
- Enable AWS multi-factor authentication (MFA) on your AWS account root user account. For more information, see [Using multi-factor authentication \(MFA\) in AWS \(p. 111\)](#).

Create individual IAM users

Don't use your AWS account root user credentials to access AWS, and don't give your credentials to anyone else. Instead, create individual users for anyone who needs access to your AWS account. Create an IAM user for yourself as well, give that user administrative permissions, and use that IAM user for all your work. For information about how to do this, see [Creating your first IAM admin user and group \(p. 20\)](#).

By creating individual IAM users for people who access 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 anytime. (If you give out your root user credentials, it can be difficult to revoke them, and it is impossible to restrict their permissions.)

AWS recommends that you create new users without permissions and require them to change their password immediately. After they sign in for the first time, you can add policies to the user. For more information, see [How do I securely create IAM users? \(p. 570\)](#).

Note

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

Use groups to assign permissions to IAM users

Instead of defining permissions for individual IAM users, it's usually more convenient to create groups that relate to job functions (administrators, developers, accounting, etc.). Next, define the relevant

permissions for each group. Finally, assign IAM users to those groups. All the users in an IAM group inherit the permissions assigned to the group. 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:

- [Creating your first IAM admin user and group \(p. 20\)](#)
- [Managing IAM groups \(p. 162\)](#)

Grant least privilege

When you create IAM policies, follow the standard security advice of granting *least privilege*, or granting only the permissions required to perform a task. Determine what users (and roles) need to do and then craft policies that allow them to perform *only* those tasks.

Start with a minimum set of permissions and grant additional permissions as necessary. Doing so is more secure than starting with permissions that are too lenient and then trying to tighten them later.

You can use access level groupings to understand the level of access that a policy grants. [Policy actions \(p. 637\)](#) are classified as List, Read, Write, Permissions management, or Tagging. For example, you can choose actions from the List and Read access levels to grant read-only access to your users. To learn how to use policy summaries to understand access level permissions, see [Use access levels to review IAM permissions \(p. 531\)](#).

One feature that can help with this is *last accessed information*. View this information on the **Access Advisor** tab on the IAM console details page for an IAM user, group, role, or policy. Last accessed information also includes information about the actions last accessed for some services, such as Amazon S3. If you sign in using AWS Organizations management account credentials, you can view service last accessed information in the **AWS Organizations** section of the IAM console. You can also use the AWS CLI or AWS API to retrieve a report for last accessed information for entities or policies in IAM or Organizations. You can use this information to identify unnecessary permissions so that you can refine your IAM or Organizations policies to better adhere to the principle of least privilege. For more information, see [Refining permissions in AWS using last accessed information \(p. 472\)](#).

To further reduce permissions, you can view your account's events in AWS CloudTrail **Event history**. CloudTrail event logs include detailed event information that you can use to reduce the policy's permissions. The logs include only the actions and resources that your IAM entities need. For more information, see [Viewing CloudTrail Events in the CloudTrail Console](#) in the *AWS CloudTrail User Guide*.

For more information, see the following:

- [Access management for AWS resources \(p. 350\)](#)
- Policy topics for individual services, which provide examples of how to write policies for service-specific resources. Examples:
 - [Authentication and Access Control for Amazon DynamoDB](#) in the *Amazon DynamoDB Developer Guide*
 - [Using Bucket Policies and User Policies](#) in the *Amazon Simple Storage Service Developer Guide*
 - [Access Control List \(ACL\) Overview](#) in the *Amazon Simple Storage Service Developer Guide*

Get started using permissions with AWS managed policies

Providing your employees with only the permissions they need requires time and detailed knowledge of IAM policies. Employees need time to learn which AWS services they want or need to use. Administrators need time to learn about and test IAM.

To get started quickly, you can use AWS managed policies to give your employees the permissions they need to get started. These policies are already available in your account and are maintained and updated by AWS. For more information about AWS managed policies, see [AWS managed policies \(p. 360\)](#).

AWS managed policies are designed to provide permissions for many common use cases. Full access AWS managed policies such as [AmazonDynamoDBFullAccess](#) and [IAMFullAccess](#) define permissions for service administrators by granting full access to a service. Power-user AWS managed policies such as [AWSCodeCommitPowerUser](#) and [AWSKeyManagementServicePowerUser](#) provide multiple levels of access to AWS services without allowing permissions management permissions. Partial-access AWS managed policies such as [AmazonMobileAnalyticsWriteOnlyAccess](#) and [AmazonEC2ReadOnlyAccess](#) provide specific levels of access to AWS services. AWS managed policies make it easier for you to assign appropriate permissions to users, groups, and roles than if you had to write the policies yourself.

AWS managed policies for job functions can span multiple services and align with common job functions in the IT industry. For a list and descriptions of job function policies, see [AWS managed policies for job functions \(p. 684\)](#).

Use customer managed policies instead of inline policies

For custom policies, we recommend that you use managed policies instead of inline policies. A key advantage of using these policies is that you can view all of your managed policies in one place in the console. You can also view this information with a single AWS CLI or AWS API operation. Inline policies are policies that exist only on an IAM identity (user, group, or role). Managed policies are separate IAM resources that you can attach to multiple identities. For more information, see [Managed policies and inline policies \(p. 359\)](#).

If you have inline policies in your account, you can convert them to managed policies. To do this, copy the policy to a new managed policy. Next, attach the new policy to the identity that has the inline policy. Then delete the inline policy. You can do this using the instructions below.

To convert an inline policy to a managed 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, choose **Groups, Users, or Roles**.
3. In the list, choose the name of the group, user, or role that has the policy you want to remove.
4. Choose the **Permissions** tab. If you chose **Groups**, expand the **Inline Policies** section if necessary.
5. For groups, choose **Show Policy** next to the inline policy that you want to remove. For users and roles, choose **Show n more**, if necessary, and then choose the arrow next to the inline policy that you want to remove.
6. Copy the JSON policy document for the policy.
7. In the navigation pane, choose **Policies**.
8. Choose **Create policy** and then choose the **JSON** tab.
9. Replace the existing text with your JSON policy text, and then choose **Review policy**.
10. Enter a name for your policy and choose **Create policy**.
11. In the navigation pane, choose **Groups, Users, or Roles**, and again choose the name of the group, user, or role that has the policy you want to remove.
12. For groups, choose **Attach Policy**. For users and roles, choose **Add permissions**.
13. For groups, select the check box next to the name of your new policy, and then choose **Attach Policy**. For users or roles, choose **Add permissions**. On the next page, choose **Attach existing policies directly**, select the check box next to the name of your new policy, choose **Next: Review**, and then choose **Add permissions**.

You are returned to the **Summary** page for your group, user, or role.

14. For groups, choose **Remove Policy** next to the inline policy that you want to remove. For users or roles, choose **X** next to the inline policy that you want to remove.

In some circumstances, we do recommend choosing inline policies over managed policies. For details, see [Choosing between managed policies and inline policies \(p. 363\)](#).

Use access levels to review IAM permissions

To improve the security of your AWS account, you should regularly review and monitor each of your IAM policies. Make sure that your policies grant the [least privilege \(p. 529\)](#) that is needed to perform only the necessary actions.

When you review a policy, you can view the [policy summary \(p. 490\)](#) that includes a summary of the access level for each service within that policy. AWS categorizes each service action into one of five *access levels* based on what each action does: List, Read, Write, Permissions management, or Tagging. You can use these access levels to determine which actions to include in your policies.

For example, in the Amazon S3 service, you might want to allow a large group of users to access List and Read actions. Such actions permit those users to list the buckets and get objects in Amazon S3. However, you should allow only a small group of users to access the Amazon S3 Write actions to delete buckets or put objects into an S3 bucket. Additionally, you should reduce permissions to allow only administrators to access the Amazon S3 Permissions management actions. This ensures that only a limited number of people can manage bucket policies in Amazon S3. This is especially important for Permissions management actions in IAM and AWS Organizations services. Allowing Tagging actions grants a user permission to perform actions that only modify tags for a resource. However, some Write actions, such as CreateRole, allow tagging a resource when you create the resource or modify other attributes for that resource. Therefore, denying access to Tagging actions does not prevent a user from tagging resources. For details and examples of the access level classification, see [Understanding access level summaries within policy summaries \(p. 500\)](#).

To view the access level classification that is assigned to each action in a service, see [Actions, Resources, and Condition Keys for AWS Services](#).

To see the access levels for a policy, you must first locate the policy's summary. The policy summary is included on the [Policies](#) page for managed policies, and on the [Users](#) page for policies that are attached to a user. For more information, see [Policy summary \(list of services\) \(p. 491\)](#).

Within a policy summary, the **Access level** column shows that the policy provides **Full** or **Limited** access to one or more of the four AWS access levels for the service. Alternately, it might show that the policy provides **Full access** to all the actions within the service. You can use the information within this **Access level** column to understand the level of access that the policy provides. You can then take action to make your AWS account more secure. For details and examples of the access level classification, see [Understanding access level summaries within policy summaries \(p. 500\)](#).

Configure a strong password policy for your users

If you allow users to change their own passwords, create a custom password policy that requires them to create strong passwords and rotate their passwords periodically. On the [Account Settings](#) page of the IAM console, you can create a custom password policy for your account. You upgrade from the AWS default password policy to define password requirements, such as minimum length, whether it requires nonalphanumeric characters, and how frequently it must be rotated. For more information, see [Setting an account password policy for IAM users \(p. 93\)](#).

Enable MFA

For extra security, we recommend that you require multi-factor authentication (MFA) for all users in your account. With MFA, users have a device that generates a response to an authentication challenge. Both the user's credentials and the device-generated response are required to complete the sign-in process. If

a user's password or access keys are compromised, your account resources are still secure because of the additional authentication requirement.

The response is generated in one of the following ways:

- Virtual and hardware MFA devices generate a code that you view on the app or device and then enter on the sign-in screen.
- U2F security keys generate a response when you tap the device. The user does not manually enter a code on the sign-in screen.

For privileged IAM users who are allowed to access sensitive resources or API operations, we recommend using U2F or hardware MFA devices.

For more information about MFA, see [Using multi-factor authentication \(MFA\) in AWS \(p. 111\)](#).

To learn how to configure MFA-protected API access for access keys, see [Configuring MFA-protected API access \(p. 138\)](#).

Use roles for applications that run on Amazon EC2 instances

Applications that run on an Amazon EC2 instance need credentials in order to access other AWS services. To provide credentials to the application in a secure way, use IAM *roles*. A role is an entity that has its own set of permissions, but that isn't a user or group. Roles also don't have their own permanent set of credentials the way IAM users do. In the case of Amazon EC2, IAM dynamically provides temporary credentials to the EC2 instance, and these credentials are automatically rotated for you.

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 [Using an IAM role to grant permissions to applications running on Amazon EC2 instances \(p. 263\)](#).

Use roles to delegate permissions

Don't share security credentials between accounts to allow users from another AWS account to access resources in your AWS account. Instead, use IAM roles. You can define a role that specifies what permissions the IAM users in the other account are allowed. You can also designate which AWS accounts have the IAM users that are allowed to assume the role. To learn whether principals in accounts outside of your zone of trust (trusted organization or account) have access to assume your roles, see [What is IAM Access Analyzer?](#).

For more information, see [Roles terms and concepts \(p. 168\)](#).

Do not share access keys

Access keys provide programmatic access to AWS. Do not embed access keys within unencrypted code or share these security credentials between users in your AWS account. For applications that need access to AWS, configure the program to retrieve temporary security credentials using an IAM role. To allow your users individual programmatic access, create an IAM user with personal access keys.

For more information, see [Switching to an IAM role \(AWS API\) \(p. 262\)](#) and [Managing access keys for IAM users \(p. 102\)](#).

Rotate credentials regularly

Change your own passwords and access keys regularly, and make sure that all IAM users in your account do as well. That way, if a password or access key is compromised without your knowledge, you limit how

long the credentials can be used to access your resources. You can apply a custom password policy to your account to require all your IAM users to rotate their AWS Management Console passwords. You can also choose how often they must do so.

For more information about setting a custom password policy in your account, see [Setting an account password policy for IAM users \(p. 93\)](#).

For more information about rotating access keys for IAM users, see [Rotating access keys \(p. 106\)](#).

Remove unnecessary credentials

Remove IAM user credentials (passwords and access keys) that are not needed. For example, if you created an IAM user for an application that does not use the console, then the IAM user does not need a password. Similarly, if a user only uses the console, remove their access keys. Passwords and access keys that have not been used recently might be good candidates for removal. You can find unused passwords or access keys using the console, using the CLI or API, or by downloading the credentials report.

For more information about finding IAM user credentials that have not been used recently, see [Finding unused credentials \(p. 146\)](#).

For more information about deleting passwords for an IAM user, see [Managing passwords for IAM users \(p. 96\)](#).

For more information about deactivating or deleting access keys for an IAM user, see [Managing access keys for IAM users \(p. 102\)](#).

For more information about IAM credential reports, see [Getting credential reports for your AWS account \(p. 148\)](#).

Use policy conditions for extra security

To the extent that it's practical, define the conditions under which your IAM policies allow access to a resource. For example, you can write conditions to specify a range of allowable IP addresses that a request must come from. You can also 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 (multi-factor authentication). For example, you can require that a user has authenticated with an MFA device in order to be allowed to terminate an Amazon EC2 instance.

For more information, see [IAM JSON policy elements: Condition \(p. 641\)](#) in the IAM Policy Elements Reference.

Monitor activity in your AWS account

You can use logging features in AWS to determine the actions users have taken in your account and the resources that were used. The log files show the time and date of actions, the source IP for an action, which actions failed due to inadequate permissions, and more.

Logging features are available in the following AWS services:

- [Amazon CloudFront](#) – Logs user requests that CloudFront receives. For more information, see [Access Logs](#) in the [Amazon CloudFront Developer Guide](#).
- [AWS CloudTrail](#) – Logs AWS API calls and related events made by or on behalf of an AWS account. For more information, see the [AWS CloudTrail User Guide](#).
- [Amazon CloudWatch](#) – Monitors your AWS Cloud resources and the applications you run on AWS. You can set alarms in CloudWatch based on metrics that you define. For more information, see the [Amazon CloudWatch User Guide](#).
- [AWS Config](#) – Provides detailed historical information about the configuration of your AWS resources, including your IAM users, groups, roles, and policies. For example, you can use AWS Config to

determine the permissions that belonged to a user or group at a specific time. For more information, see the [AWS Config Developer Guide](#).

- [Amazon Simple Storage Service \(Amazon S3\)](#) – Logs access requests to your Amazon S3 buckets. For more information, see [Server Access Logging](#) in the [Amazon Simple Storage Service Developer Guide](#).

Video presentation about IAM best practices

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

[AWS re:Invent 2015 - IAM Best Practices](#)

Business use cases for IAM

A simple business use case for IAM can help you understand basic ways you might implement the service to control the AWS access that your users have. The use case is described in general terms, without the mechanics of how you'd use the IAM API to achieve the results you want.

This use case looks at two typical ways a fictional company called Example Corp might use IAM. The first scenario considers Amazon Elastic Compute Cloud (Amazon EC2). The second considers Amazon Simple Storage Service (Amazon S3).

For more information about using IAM with other services from AWS, see [AWS services that work with IAM \(p. 611\)](#).

Topics

- [Initial setup of example corp \(p. 535\)](#)

- [Use case for IAM with Amazon EC2 \(p. 535\)](#)
- [Use case for IAM with Amazon S3 \(p. 536\)](#)

Initial setup of example corp

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

John uses the AWS Management Console with the AWS account root user credentials to create a user for himself called *John*, and a group called *Admins*. He gives the Admins group permissions to perform all actions on all the AWS account's resources using the AWS managed policy [AdministratorAccess](#). Then he adds the John user to the Admins group. For a step-by-step guide to creating an Administrators group and an IAM user for yourself, then adding your user to the Administrators group, see [Creating your first IAM admin user and group \(p. 20\)](#).

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

John also creates a group called *AllUsers* so that 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 creating groups, see [Creating IAM groups \(p. 161\)](#). For information about creating users, see [Creating an IAM user in your AWS account \(p. 76\)](#). For information about adding users to groups, see [Managing IAM groups \(p. 162\)](#).

Use case for IAM with Amazon EC2

A company like Example Corp typically uses IAM to interact with services like Amazon EC2. To understand this part of the use case, you need a basic understanding of Amazon EC2. For more information about Amazon EC2, go to the [Amazon EC2 User Guide for Linux Instances](#).

Amazon EC2 permissions for the groups

To provide "perimeter" control, John attaches a policy to the AllUsers group. This policy denies any AWS request from a user if the originating IP address is outside 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. John attaches the [AmazonEC2FullAccess](#) AWS managed policy to the SysAdmins group that gives members of the group permission to use all the Amazon EC2 actions.
- **Developers** – Need the ability to work with instances only. John therefore creates and attaches a policy to the Developers group that allows developers to call `DescribeInstances`, `RunInstances`, `StopInstances`, `StartInstances`, and `TerminateInstances`.

Note

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

- **Managers** – Should not be able to perform any Amazon EC2 actions except listing the Amazon EC2 resources currently available. Therefore, John creates and attaches a policy to the Managers group that only lets them call Amazon EC2 "Describe" API operations.

For examples of what these policies might look like, see [Example IAM identity-based policies \(p. 389\)](#) and [Using AWS Identity and Access Management](#) in the [Amazon EC2 User Guide for Linux Instances](#).

User's job function change

At some point, one of the developers, Paulo, changes job functions and becomes a manager. John moves Paulo from the Developers group to the Managers group. Now that he's in the Managers group, Paulo'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

Companies like Example Corp would also typically use IAM with Amazon S3. John has created an Amazon S3 bucket for the company called *aws-s3-bucket*.

Creation of other users and groups

As employees, Zhang and Mary each need to be able to create their own data in the company's bucket. They also need to read and write shared data that all developers work on. To enable this, John logically arranges the data in *aws-s3-bucket* using an Amazon S3 key prefix scheme as shown in the following figure.

```
/aws-s3-bucket
  /home
    /zhang
    /mary
  /share
    /developers
    /managers
```

John divides the */aws-s3-bucket* into a set of home directories for each employee, and a shared area for groups of developers and managers.

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

- **Home directory access for Zhang** – John attaches a policy to Zhang that lets him read, write, and list any objects with the Amazon S3 key prefix */aws-s3-bucket/home/zhang/*
- **Home directory access for Mary** – John attaches a policy to Mary that lets her read, write, and list any objects with the Amazon S3 key prefix */aws-s3-bucket/home/mary/*
- **Shared directory access for the developers group** – John attaches a policy to the group that lets developers read, write, and list any objects in */aws-s3-bucket/share/developers/*
- **Shared directory access for the managers group** – John attaches a policy to the group that lets managers read, write, and list objects in */aws-s3-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.

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 runtime, see [Policy evaluation logic \(p. 666\)](#).

User's job function change

At some point, one of the developers, Zhang, changes job functions and becomes a manager. We assume that he no longer needs access to the documents in the *share/developers* directory. John, as an admin, moves Zhang to the Managers group and out of the Developers group. With just that simple

reassignment, Zhang 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 Shirley needs to put data into a bucket for Example Corp's use. John creates a group called *WidgetCo* and a user named Shirley and adds Shirley to the WidgetCo group. John also creates a special bucket called *aws-s3-bucket1* for Shirley to use.

John updates existing policies or adds new ones to accommodate the partner Widget Company. For example, John 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.

Using AWS IAM Access Analyzer

AWS IAM Access Analyzer helps you identify the resources in your organization and accounts, such as Amazon S3 buckets or IAM roles, that are shared with an external entity. This lets you identify unintended access to your resources and data, which is a security risk. Access Analyzer identifies resources that are shared with external principals by using logic-based reasoning to analyze the resource-based policies in your AWS environment. For each instance of a resource that is shared outside of your account, Access Analyzer generates a finding. Findings include information about the access and the external principal that it is granted to. You can review findings to determine whether the access is intended and safe, or the access is unintended and a security risk.

Note

An external entity can be another AWS account, a root user, an IAM user or role, a federated user, an AWS service, an anonymous user, or other entity that you can use to create a filter. For more information, see [AWS JSON Policy Elements: Principal](#).

When you enable Access Analyzer, you create an analyzer for your entire organization or your account. The organization or account you choose is known as the zone of trust for the analyzer. The analyzer monitors all of the [supported resources \(p. 539\)](#) within your zone of trust. Any access to resources by principals that are within your zone of trust is considered trusted. Once enabled, Access Analyzer analyzes the policies applied to all of the supported resources in your zone of trust. After the first analysis, Access Analyzer analyzes these policies periodically. If a new policy is added, or an existing policy is changed, Access Analyzer analyzes the new or updated policy within about 30 minutes.

When analyzing the policies, if Access Analyzer identifies one that grants access to an external principal that isn't within your zone of trust, it generates a finding. Each finding includes details about the resource, the external entity that has access to it, and the permissions granted so that you can take appropriate action. You can view the details included in the finding to determine whether the resource access is intentional or a potential risk that you should resolve. When you add a policy to a resource, or update an existing policy, Access Analyzer analyzes the policy. Access Analyzer also analyzes all resource-based policies periodically.

On rare occasions under certain conditions, Access Analyzer is not notified that a policy was added or updated. For example, a change to account-level block public access settings on an S3 bucket can take up to 6 hours. Also, if there is a delivery issue with AWS CloudTrail log delivery the policy change does not trigger a rescan of the resource that was reported in the finding. When this happens, Access Analyzer analyzes the new or updated policy during the next periodic scan, which is within 24 hours. If you want to confirm that a change you make to a policy resolves an access issue reported in a finding, you can rescan the resource reported in a finding by using the [Rescan](#) link in the Finding details page, or by using the [StartResourceScan](#) operation of the Access Analyzer API. To learn more, see [Resolving findings \(p. 554\)](#).

Important

Access Analyzer analyzes only policies that are applied to resources in the same AWS Region that it's enabled in. To monitor all resources in your AWS environment, you must create an analyzer to enable Access Analyzer in each Region where you're using supported AWS resources.

Access Analyzer analyzes the following resource types:

- [Amazon Simple Storage Service buckets \(p. 539\)](#)
- [AWS Identity and Access Management roles \(p. 539\)](#)
- [AWS Key Management Service keys \(p. 539\)](#)
- [AWS Lambda functions and layers \(p. 540\)](#)

- [Amazon Simple Queue Service queues \(p. 541\)](#)

Access Analyzer resource types

Access Analyzer analyzes the resource-based policies that are applied to AWS resources in the Region where you enabled Access Analyzer. Only resource-based policies are analyzed. Review the information about each resource for details about how Access Analyzer generates findings for each resource type.

Supported resource types:

- [Amazon Simple Storage Service buckets \(p. 539\)](#)
- [AWS Identity and Access Management roles \(p. 539\)](#)
- [AWS Key Management Service keys \(p. 539\)](#)
- [AWS Lambda functions and layers \(p. 540\)](#)
- [Amazon Simple Queue Service queues \(p. 541\)](#)

Amazon Simple Storage Service buckets

When Access Analyzer analyzes Amazon S3 buckets, it generates a finding when an Amazon S3 bucket policy, ACL, or access point applied to a bucket grants access to an external entity. An external entity is a principal or other entity that you can use to [create a filter \(p. 551\)](#) that isn't within your zone of trust. For example, if a bucket policy grants access to another account or allows public access, Access Analyzer generates a finding. However, if you enable [Block public access](#) on your bucket, you can block access at the account level or the bucket level.

Amazon S3 *block public access* settings override the bucket policies that are applied to the bucket. The settings also override the access point policies applied to the bucket's access points. Access Analyzer analyzes block public access settings at the bucket level whenever a policy changes. However, it evaluates the block public access settings at the account level only once every 6 hours. This means that Access Analyzer might not generate or resolve a finding for public access to a bucket for up to 6 hours. For example, if you have a bucket policy that allows public access, Access Analyzer generates a finding for that access. If you then enable block public access to block all public access to the bucket at the account level, Access Analyzer doesn't resolve the finding for the bucket policy for up to 6 hours, even though all public access to the bucket is blocked.

AWS Identity and Access Management roles

For IAM roles, Access Analyzer analyzes [trust policies](#). In a role trust policy, you define the principals that you trust to assume the role. A role trust policy is a required resource-based policy that is attached to a role in IAM. Access Analyzer generates findings for roles within the zone of trust that can be accessed by an external entity that is outside your zone of trust.

Note

An IAM role is a global resource. If a role trust policy grants access to an external entity, Access Analyzer generates a finding in each enabled Region.

AWS Key Management Service keys

For AWS KMS customer master keys (CMKs), Access Analyzer analyzes the key policies and grants applied to a key. Access Analyzer generates a finding if a key policy or grant allows an external entity to access the key. For example, if you use the `kms:CallerAccount` condition key in a policy statement to allow access to all users in a specific AWS account, and you specify an account other than the current account (the zone of trust for the current analyzer), Access Analyzer generates a finding. To learn more about KMS condition keys in IAM policy statements, see [AWS KMS Condition Keys](#).

When Access Analyzer analyzes a KMS key it reads key metadata, such as the key policy and list of grants. If the key policy doesn't allow the Access Analyzer role to read the key metadata, an Access Denied error finding is generated. For example, if the following example policy statement is the only policy applied to a key, it results in an Access Denied error finding in Access Analyzer:

```
{
    "Sid": "Allow access for Key Administrators",
    "Effect": "Allow",
    "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/Admin"
    },
    "Action": "kms:*",
    "Resource": "*"
}
```

Because this statement allows only the role named *Admin* from the AWS account 111122223333 to access the key, an Access Denied error finding is generated because Access Analyzer isn't able to fully analyze the key. An error finding is displayed in red text in the **Findings** table. The finding looks similar to the following:

```
{
    "error": "ACCESS_DENIED",
    "id": "12345678-1234-abcd-dcba-111122223333",
    "analyzedAt": "2019-09-16T14:24:33.352Z",
    "resource": "arn:aws:kms:us-west-2:1234567890:key/1a2b3c4d-5e6f-7a8b-9c0d-1a2b3c4d5e6f7g8a",
    "resourceType": "AWS::KMS::Key",
    "status": "ACTIVE",
    "updatedAt": "2019-09-16T14:24:33.352Z"
}
```

When you create a KMS CMK, the permissions granted to access the key depend on how you create the key. If you receive an Access Denied error finding for a key resource, apply the following policy statement to the key resource to grant Access Analyzer permission to access the key.

```
{
    "Sid": "Allow Access Analyzer access to key metadata",
    "Effect": "Allow",
    "Principal": {
        "AWS": "arn:aws:iam::111122223333:root"
    },
    "Action": [
        "kms:DescribeKey",
        "kms:GetKeyPolicy",
        "kms>List*"
    ],
    "Resource": "*"
},
```

After you receive an Access Denied finding for a KMS key resource, and then resolve the finding by updating the key policy, the finding is updated to a status of Resolved. If there are policy statements or key grants that grant permission to the key to an external entity, you might see additional findings for the key resource.

AWS Lambda functions and layers

For AWS Lambda functions, Access Analyzer analyzes policies, including condition statements in a policy, that grant access to the function to an external entity. Access Analyzer also analyzes permissions granted when using the [AddPermission](#) operation of the AWS Lambda API with an `EventSourceToken`.

Amazon Simple Queue Service queues

For Amazon SQS queues, Access Analyzer analyzes policies, including condition statements in a policy, that allow an external entity access to a queue.

How Access Analyzer works

This topic describes the concepts and terms that are used in Access Analyzer to help you become familiar with how Access Analyzer monitors access to your AWS resources.

AWS IAM Access Analyzer is built on [Zelkova](#), which translates IAM policies into equivalent logical statements, and runs a suite of general-purpose and specialized logical solvers (satisfiability modulo theories) against the problem. Access Analyzer applies Zelkova repeatedly to a policy with increasingly specific queries to characterize classes of behaviors the policy allows, based on the content of the policy. To learn more about satisfiability modulo theories, see [Satisfiability Modulo Theories](#).

Access Analyzer does not examine access logs to determine whether an external entity accessed a resource within your zone of trust. It generates a finding when a resource-based policy allows access to a resource, even if the resource was not accessed by the external entity. Access Analyzer also does not consider the state of any external accounts when making its determination. That is, if it indicates that account 11112222333 can access your S3 bucket, it knows nothing about the state of users, roles, service control policies (SCP), and other relevant configurations in that account. This is for customer privacy – Access Analyzer doesn't consider who owns the other account. It is also for security – if the account is not owned by the Access Analyzer customer, it is still important to know that an external entity could gain access to their resources even if there are currently no principals in the account that could access the resources.

Access Analyzer considers only certain IAM condition keys that external users cannot directly influence, or that are otherwise impactful to authorization.

Access Analyzer does not currently report findings from AWS service principals or internal service accounts. In rare cases where Access Analyzer isn't able to fully determine whether a policy statement grants access to an external entity, it errs on the side of declaring a false positive finding. Access Analyzer is designed to provide a comprehensive view of the resource sharing in your account, and strives to minimize false negatives.

Getting started with AWS IAM Access Analyzer

Use the information in this topic to learn about the requirements necessary to use and manage AWS IAM Access Analyzer, and then how to enable Access Analyzer. To learn more about the service-linked role for Access Analyzer, see [Using service-linked roles for AWS IAM Access Analyzer \(p. 545\)](#).

Permissions required to use Access Analyzer

To successfully configure and use Access Analyzer, the account you use must be granted the required permissions. To access and use all Access Analyzer features, you can apply the IAMAccessAnalyzerFullAccess managed policy to the account. The full access policy grants the following permissions:

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

```

"Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "access-analyzer:*"
        ],
        "Resource": "*"
    },
    {
        "Effect": "Allow",
        "Action": "iam:CreateServiceLinkedRole",
        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "iam:AWSServiceName": "access-analyzer.amazonaws.com"
            }
        }
    },
    {
        "Effect": "Allow",
        "Action": [
            "organizations:DescribeAccount",
            "organizations:DescribeOrganization",
            "organizations:DescribeOrganizationalUnit",
            "organizations>ListAccounts",
            "organizations>ListAccountsForParent",
            "organizations>ListAWSAccessForOrganization",
            "organizations>ListChildren",
            "organizations>ListDelegatedAdministrators",
            "organizations>ListOrganizationalUnitsForParent",
            "organizations>ListParents",
            "organizations>ListRoots"
        ],
        "Resource": "*"
    }
]
}

```

A custom policy for managing Access Analyzer must include the following permissions:

- access-analyzer: *
- iam:CreateServiceLinkedRole

To allow read-only access to Access Analyzer, use the IAMAccessAnalyzerReadOnlyAccess managed policy. This policy grants the following permissions:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "access-analyzer:Get*",
                "access-analyzer>List*"
            ],
            "Resource": "*"
        }
    ]
}
```

If you plan to use Access Analyzer for an organization in AWS Organizations, you need to [enable trusted access](#) for Access Analyzer in AWS Organizations. You also need the following permissions:

- organizations:DescribeAccount
- organizations:DescribeOrganization
- organizations:DescribeOrganizationalUnit
- organizations>ListAccounts
- organizations>ListAccountsForParent
- organizations>ListAWSAccessForOrganization
- organizations>ListChildren
- organizations>ListDelegatedAdministrators
- organizations>ListOrganizationalUnitsForParent
- organizations>ListParents
- organizations>ListRoots

Resources defined by AWS IAM Access Analyzer

Access Analyzer defines the following resources:

Resource	ARN
analyzer	arn:\${Partition}:access-analyzer:\${Region}:\${Account}:analyzer/\${analyzerName}
archive-rule	arn:\${Partition}:access-analyzer:\${Region}:\${Account}:analyzer/\${analyzerName}/archive-rule/\${ruleName}

Required Access Analyzer service permissions

Access Analyzer uses a service-linked role named `AWSServiceRoleForAccessAnalyzer` to grant the service read-only access to analyze AWS resources with resource-based policies on your behalf. When you create an analyzer with your account as the zone of trust, the service creates the role your account. When you create an analyzer with your organization as the zone of trust, the service creates a role in each account that belongs to your organization. For more information, see [Using service-linked roles for AWS IAM Access Analyzer \(p. 545\)](#).

Note

Access Analyzer is Regional. You must enable Access Analyzer in each Region independently.

In some cases, after you enable Access Analyzer, the **Findings** page loads with no findings. This might be due to a delay in the console for populating your findings. You need to manually refresh the browser to view your findings. If you still don't see any findings, it's because you have no supported resources in your account that can be accessed by an external entity. If a policy that grants access to an external entity is applied to a resource, Access Analyzer generates a finding.

Note

It may take up to 30 minutes after a policy is modified for Access Analyzer to analyze the resource and then either generate a new finding or update an existing finding for the access to the resource.

Enabling Access Analyzer

To enable Access Analyzer in a Region, you must create an analyzer in that Region. You must create an analyzer in each Region in which you want to monitor access to your resources.

To create an analyzer with the account as the zone of trust

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. Choose **Access analyzer**.
3. Choose **Create analyzer**.
4. On the **Create analyzer** page, confirm that the Region displayed is the Region where you want to enable Access Analyzer.
5. Enter a name for the analyzer.
6. Choose the account as the zone of trust for the analyzer.

Note

If your account is not the AWS Organizations management account or [delegated administrator \(p. 548\)](#) account, you can create only one analyzer with your account as the zone of trust.

7. Optional. Add any tags that you want to apply to the analyzer.
8. Choose **Create Analyzer**.

When you create an analyzer to enable Access Analyzer, a service-linked role named `AWSServiceRoleForAccessAnalyzer` is created in your account.

To create an analyzer with the organization as the zone of trust

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. Choose **Access analyzer**.
3. Choose **Create analyzer**.
4. On the **Create analyzer** page, confirm that the Region displayed is the Region where you want to enable Access Analyzer.
5. Enter a name for the analyzer.
6. Choose your organization as the zone of trust for the analyzer.
7. Optional. Add any tags that you want to apply to the analyzer.
8. Choose **Create Analyzer**.

When you create an analyzer with the organization as the zone of trust, a service-linked role named `AWSServiceRoleForAccessAnalyzer` is created in each account of your organization.

Access Analyzer status

To view the status of your analyzers, choose **Analyzers**. Analyzers created for an organization or account can have the following status:

Status	Description
Active	The analyzer is actively monitoring resources within its zone of trust. The analyzer actively generates new findings and updates existing findings.
Creating	The creation of the analyzer is still in progress. The analyzer becomes active once creation is complete.
Disabled	The analyzer is disabled due to an action taken by the AWS Organizations administrator. For

Status	Description
	example, removing the analyzer's account as the delegated administrator for IAM Access Analyzer. When the analyzer is in a disabled state it does not generate new findings or update existing findings.
Failed	The creation of the analyzer failed due to a configuration issue. The analyzer won't generate any findings. Delete the analyzer and create a new analyzer.

Access Analyzer quotas

Access Analyzer has the following quotas:

Resource	Default quota	Maximum quota
Maximum analyzers with an account zone of trust	1	1
Maximum analyzers with an organization zone of trust	5	20*
Maximum archive rules per analyzer	100 Each archive rule can have up to 20 values per criterion.	1000*

* Some quotas are customer-configurable using [Service Quotas](#).

Using service-linked roles for AWS IAM Access Analyzer

AWS IAM Access Analyzer uses an IAM [service-linked role](#). A service-linked role is a unique type of IAM role that is linked directly to Access Analyzer. Service-linked roles are predefined by Access Analyzer and include all the permissions that the feature requires to call other AWS services on your behalf.

A service-linked role makes setting up Access Analyzer easier because you don't have to manually add the necessary permissions. Access Analyzer defines the permissions of its service-linked roles, and unless defined otherwise, only Access Analyzer can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

For information about other services that support service-linked roles, see [AWS Services That Work with IAM](#) and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for AWS IAM Access Analyzer

AWS IAM Access Analyzer uses the service-linked role named **AWSServiceRoleForAccessAnalyzer – Allow Access Analyzer to analyze resource metadata**.

The AWSServiceRoleForAccessAnalyzer service-linked role trusts the following services to assume the role:

- access-analyzer.amazonaws.com

The role permissions policy allows Access Analyzer to complete the following actions on the specified resources:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:GetBucketPublicAccessBlock",  
                "s3:GetBucketPolicyStatus",  
                "s3:GetAccountPublicAccessBlock",  
                "s3>ListAllMyBuckets",  
                "s3:GetBucketAcl",  
                "s3:GetBucketLocation",  
                "s3:GetBucketPolicy",  
                "s3>ListAccessPoints",  
                "s3:GetAccessPoint",  
                "s3:GetAccessPointPolicy",  
                "s3:GetAccessPointPolicyStatus",  
                "iam:GetRole",  
                "iam>ListRoles",  
                "kms:DescribeKey",  
                "kms:GetKeyPolicy",  
                "kms>ListGrants",  
                "kms>ListKeyPolicies",  
                "kms>ListKeys",  
                "ec2:DescribeVpcs",  
                "ec2:DescribeVpcEndpoints",  
                "ec2:DescribeByoipCidrs",  
                "ec2:DescribeAddresses",  
                "lambda>ListFunctions",  
                "lambda:GetPolicy",  
                "lambda>ListLayers",  
                "lambda>ListLayerVersions",  
                "lambda:GetLayerVersionPolicy",  
                "sns:GetQueueAttributes",  
                "sns>ListQueues",  
                "organizations>ListAWSServiceAccessForOrganization",  
                "organizations>ListDelegatedAdministrators",  
                "organizations>ListRoots",  
                "organizations>ListParents",  
                "organizations>ListChildren",  
                "organizations>ListOrganizationalUnitsForParent",  
                "organizations>ListAccountsForParent",  
                "organizations>ListAccounts",  
                "organizations:DescribeAccount",  
                "organizations:DescribeOrganization",  
                "organizations:DescribeOrganizationalUnit"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see [Service-Linked Role Permissions](#) in the *IAM User Guide*.

Creating a service-linked role for Access Analyzer

You don't need to manually create a service-linked role. When you enable Access Analyzer in the AWS Management Console or the AWS API, Access Analyzer creates the service-linked role for you. The same service-linked role is used in all Regions in which you enable Access Analyzer.

Note

Access Analyzer is Regional. You must enable Access Analyzer in each Region independently.

If you delete this service-linked role, Access Analyzer recreates the role when you next create an analyzer.

You can also use the IAM console to create a service-linked role with the **Access Analyzer** use case. In the AWS CLI or the AWS API, create a service-linked role with the `access-analyzer.amazonaws.com` service name. For more information, see [Creating a Service-Linked Role](#) in the *IAM User Guide*. If you delete this service-linked role, you can use this same process to create the role again.

Editing a service-linked role for Access Analyzer

Access Analyzer does not allow you to edit the `AWSServiceRoleForAccessAnalyzer` service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see [Editing a Service-Linked Role](#) in the *IAM User Guide*.

Deleting a service-linked role for Access Analyzer

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that isn't actively monitored or maintained. However, you must clean up the resources for your service-linked role before you can manually delete it.

Note

If Access Analyzer is using the role when you try to delete the resources, then the deletion might fail. If that happens, wait for a few minutes and try the operation again.

To delete Access Analyzer resources used by the `AWSServiceRoleForAccessAnalyzer`

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the **Access reports** section, under **Access analyzer**, choose **Analyzers**.
3. Choose the check box on the top left above the list of analyzers in the **Analyzers** table to select all analyzers.
4. Choose **Delete**.
5. To confirm that you want to delete the analyzers, enter **delete**, and then choose **Delete**.

To manually delete the service-linked role using IAM

Use the IAM console, the AWS CLI, or the AWS API to delete the `AWSServiceRoleForAccessAnalyzer` service-linked role. For more information, see [Deleting a Service-Linked Role](#) in the *IAM User Guide*.

Supported Regions for Access Analyzer service-linked roles

Access Analyzer supports using service-linked roles in all of the Regions where the service is available. For more information, see [AWS Regions and Endpoints](#).

Settings for Access Analyzer

If you're configuring AWS IAM Access Analyzer in your AWS Organizations management account, you can add a member account in the organization as the delegated administrator to manage Access Analyzer for your organization. The delegated administrator has permissions to create and manage analyzers with the organization as the zone of trust. Only the management account can add a delegated administrator.

Delegated administrator for Access Analyzer

The delegated administrator for Access Analyzer is a member account within the organization that has permissions to create and manage analyzers with the organization as the zone of trust. Only the management account can add, remove, or change a delegated administrator.

If you add a delegated administrator, you can later change to a different account for the delegated administrator. When you do, the former delegated administrator account loses permission to all analyzers with organization as the zone of trust that were created using that account. These analyzers move to a disabled state and no longer generate new or update existing findings. The existing findings for these analyzers are also no longer accessible. You can access them again in the future by configuring the account as the delegated administrator. If you know that you won't use the same account as a delegated administrator, consider deleting the analyzers before changing the delegated administrator. This deletes all findings generated. When the new delegated administrator creates new analyzers, new instances of the same findings are generated. You don't lose any findings, they just get generated for the new analyzer in a different account. And you can continue to access findings for the organization using the organization management account, which also has administrator permissions. The new delegated administrator must create new analyzers for Access Analyzer to start monitoring resources in your organization.

If the delegated administrator leaves the AWS organization, the delegated administration privileges are removed from the account. All analyzers in the account with the organization as the zone of trust move to a disabled state. The existing findings for these analyzers are also no longer accessible.

The first time that you configure analyzers in the management account, you can choose the option to **Add delegated administrator** that is displayed on the Access Analyzer homepage in the IAM console.

To add a delegated administrator using the console

1. Log in to the AWS console using the management account for your organization.
2. Open the IAM console at <https://console.aws.amazon.com/iam/>.
3. Under **Access Analyzer**, choose **Settings**.
4. Choose **Add delegated administrator**.
5. Enter the account number of an organization member account to make the delegated administrator.

The account must be a member of your organization.
6. Choose **Save changes**.

To add a delegated administrator using the AWS CLI or the AWS SDKs

When you create an analyzer with organization as the zone of trust in a delegated administrator account using the AWS CLI, AWS API (using the AWS SDKs) or AWS CloudFormation, you must use AWS Organizations APIs to enable service access for Access Analyzer and register the member account as a delegated administrator.

1. Enable trusted service access for Access Analyzer in AWS Organizations. See [How to Enable or Disable Trusted Access](#) in the AWS Organizations User Guide.

2. Register a valid member account of your AWS organization as a delegated administrator using the AWS Organizations [RegisterDelegatedAdministrator](#) API operation or the `register-delegated-administrator` AWS CLI command.

After you change the delegated administrator, the new administrator must create analyzers to start monitoring access to the resources in your organization.

Access Analyzer findings

Access Analyzer generates a finding for each instance of a resource-based policy that grants access to a resource within your zone of trust to a principal that is not within your zone of trust. When you create an analyzer, you choose an organization or AWS account to analyze. Any principal in the organization or account that you choose for the analyzer is considered trusted. Because principals in the same organization or account are trusted, the resources and principals within the organization or account comprise the zone of trust for the analyzer. Any sharing that is within the zone of trust is considered safe, so Access Analyzer does not generate a finding. For example, if you select an organization as the zone of trust for an analyzer, all resources and principals in the organization are within the zone of trust. If you grant permissions to an S3 bucket in one of your organization member accounts to a principal in another organization member account, Access Analyzer does not generate a finding. But if you grant permission to a principal in an account that is not a member of the organization, Access Analyzer generates a finding.

Topics

- [Working with findings \(p. 549\)](#)
- [Reviewing findings \(p. 550\)](#)
- [Filtering findings \(p. 551\)](#)
- [Archiving findings \(p. 553\)](#)
- [Resolving findings \(p. 554\)](#)

Working with findings

Findings are generated only once for each instance of a resource that is shared outside of your zone of trust. Each time a resource-based policy is modified, Access Analyzer analyzes the policy. If the updated policy shares a resource that is already identified in a finding, but with different permissions or conditions, a new finding is generated for that instance of the resource sharing. If the access in the first finding is removed, that finding is updated to a status of Resolved.

The status of all findings remains Active until you archive them or remove the access that generated the finding. When you remove the access, the finding status is updated to Resolved.

Note

It may take up to 30 minutes after a policy is modified for Access Analyzer to analyze the resource and then update the finding.

You should review all of the findings in your account to determine whether the sharing is expected and approved. If the sharing identified in the finding is expected, you can archive the finding. When you archive a finding, the status is changed to Archived, and the finding is removed from the Active findings list. The finding is not deleted. You can view your archived findings at any time. Work through all of the findings in your account until you have zero active findings. After you get to zero findings, you know that any new Active findings that are generated are from a recent change in your environment.

Reviewing findings

After you [enable Access Analyzer \(p. 543\)](#), the next step is to review any findings to determine whether the access identified in the finding is intentional or unintentional. You can also review findings to determine common findings for access that is intended, and then [create an archive rule \(p. 554\)](#) to automatically archive those findings. You can also review archived and resolved findings.

To review findings

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. Choose **Access analyzer**.

Note

Findings are displayed only if you have permission to view findings for the analyzer.

All Active findings are displayed for the analyzer. To view other findings generated by the analyzer, choose the appropriate tab:

- Choose **Active** to view all active findings that were generated by the analyzer.
- Choose **Archived** to view only findings generated by the analyzer that have been archived. To learn more, see [Archiving findings \(p. 553\)](#).
- Choose **Resolved** to view only findings that were generated by the analyzer that have been resolved. When you remediate the issue that generated the finding, the finding status is changed to Resolved.

Important

Resolved findings are deleted 90 days after the last update to the finding. Active and archived findings are not deleted unless you delete the analyzer that generated them.

- Choose **All** to view all findings with any status that were generated by the analyzer.

The **Findings** page displays the following details about the shared resource and policy statement that generated the finding:

Finding ID

The unique ID assigned to the finding. Choose the finding ID to display additional details about the resource and policy statement that generated the finding.

Resource

The type and partial name of the resource that has a policy applied to it that grants access to an external entity not within your zone of trust.

Resource owner account

This column is displayed only if you are using an organization as the zone of trust. The account in the organization that owns the resource reported in the finding.

External principal

The principal, not within your zone of trust, that the analyzed policy grants access to. Valid values include:

- **AWS account** – All principals in the listed AWS account with permissions from that account's administrator can access the resource.
- **Any principal** – All principals in any AWS account that meet the conditions included in the **Conditions** column have permission to access the resource. For example, if a VPC is listed, it means that any principal in any account that has permission to access the listed VPC can access the resource.

- **Canonical user** – All principals in the AWS account with the listed canonical user ID have permission to access the resource.
- **IAM role** – The listed IAM role has permission to access the resource.
- **IAM user** – The listed IAM user has permission to access the resource.

Condition

The condition from the policy statement that grants the access. For example, if the **Condition** field includes **Source VPC**, it means that the resource is shared with a principal that has access to the VPC listed. Conditions can be global or service-specific. [Global condition keys](#) have the `aws:` prefix.

Shared through

The **Shared through** field indicates how the access that generated the finding is granted. Valid values include:

- **Bucket policy** – The bucket policy that is attached to the Amazon S3 bucket.
- **Access control list** – The access control list (ACL) that is attached to the Amazon S3 bucket.
- **Access point** – An access point that is associated with the Amazon S3 bucket. The ARN of the access point is displayed in the **Findings** details.

Access level

The level of access granted to the external entity by the actions in the resource-based policy. View the details of the finding for more information. Access level values include the following:

- **List** – Permission to list resources within the service to determine whether an object exists. Actions with this level of access can list objects but cannot see the contents of a resource.
- **Read** – Permission to read but not edit the contents and attributes of resources in the service.
- **Write** – Permission to create, delete, or modify resources in the service.
- **Permissions** – Permission to grant or modify resource permissions in the service.
- **Tagging** – Permission to perform actions that only change the state of resource tags.

Updated

A timestamp for the most recent update to the finding status, or the time and date at which the finding was generated if no updates have been made.

Note

It may take up to 30 minutes after a policy is modified for Access Analyzer to again analyze the resource and then update the finding.

Status

The status of the finding, one of **Active**, **Archived**, or **Resolved**.

Filtering findings

The default filtering for the page is to display all active findings. To view archived findings, choose the **Archived** tab. When you first start using Access Analyzer, there are no archived findings.

Use filters to display only the findings for a specific resource, account, principal, or other value. To create a filter, select the property to filter on, then choose a property value to filter on. For example, to create a filter that displays only findings for a specific AWS account, choose **AWS Account** for the property, then enter the account number for the AWS account that you want to view findings for. To create a filter that displays only findings for resources that allow public access, you can choose the **Public access** property, then choose **Public access: true**.

For a list of filter keys that you can use to create or update an archive rule, see [Access Analyzer filter keys \(p. 556\)](#).

To filter the findings displayed

1. Choose the **Filter active findings** field.
2. Choose the property to use to filter the findings displayed.
3. Choose the value to match for the property. Only findings with that value in the finding are displayed.

For example, if you choose **Resource** as the property, type part or all of the name of a bucket, then press Enter. Only findings for the bucket that matches the filter criteria are displayed.

You can add additional properties to further filter the findings displayed. When you add additional properties, only findings that match all conditions in the filter are displayed. Defining a filter to display findings that match one property OR another property is not supported.

Some fields are displayed only when you are viewing findings for an analyzer with an organization as its zone of trust.

The following properties are available for defining filters:

- **Public access** – To filter by findings for resources that allow public access, filter by **Public access** then choose **Public access: true**.
- **Resource** – To filter by resource, type all or part of the name of the resource.
- **Resource Type** – To filter by resource type, choose the type from the list displayed.
- **AWS Account** – Use this property to filter by AWS account that is granted access in the **Principal** section of a policy statement. To filter by AWS account, type all or part of the 12-digit AWS account ID, or all or part of the full account ARN of the external AWS user or role that has access to resources in the current account.
- **Canonical User** – To filter by canonical user, type the canonical user ID as defined for S3 buckets. To learn more, see [AWS Account Identifiers](#).
- **Federated User** – To filter by federated user, type all or part of the ARN of the federated identity. To learn more, see [Identity Providers and Federation](#).
- **Principal ARN** – Use this property to filter on the ARN of the principal (IAM user, role, or group) used in an **aws:PrincipalArn** condition key. To filter by Principal ARN, type all or part of the ARN of the IAM user, role, or group from an external AWS account reported in a finding.
- **Principal OrgID** – To filter by Principal OrgID, type all or part of the organization ID associated with the external principals that belong to the AWS organization specified as a condition in the finding. To learn more, see [AWS Global Condition Context Keys](#).
- **Principal Org Paths** – To filter by Principal Org Paths, type all or part of the ID for the AWS organization or organizational unit (OU) that allows access to all external principals that are account members of the specified organization or OU as a condition in the policy. To learn more, see [AWS Global Condition Context Keys](#).
- **Source Account** – To filter on source account, type all or part of the AWS account ID associated with the resources, as used in some cross-service permissions in AWS.
- **Source ARN** – To filter by Source ARN, type all or part of the ARN specified as a condition in the finding. To learn more, see [To filter by Principal Org Paths](#), type all or part of the ID for the AWS organization or organizational unit (OU) that allows access to all external principals that are account members of the specified organization or OU as a condition in the policy. To learn more, see [AWS Global Condition Context Keys](#).
- **Source IP** – To filter by Source IP, type all or part of the IP address that allows external entities access to resources in the current account when using the specified IP address. To learn more, see [AWS Global Condition Context Keys](#).
- **Source VPC** – To filter by Source VPC, type all or part of the VPC ID that allows external entities access to resources in the current account when using the specified VPC. To learn more, see [AWS Global Condition Context Keys](#).

- **Source VPCE** – filter by Source VPCE, type all or part of the VPC endpoint ID that allows external entities access to resources in the current account when using the specified VPC endpoint. To learn more, see [AWS Global Condition Context Keys](#).
- **User ID** – To filter by User ID, type all or part of the user ID of the IAM user from an external AWS account who is allowed access to resource in the current account. To learn more, see [AWS Global Condition Context Keys](#).
- **KMS Key ID** – To filter by KMS Key ID, type all or part of the key ID for the KMS key specified as a condition for KMS-encrypted S3 object access in your current account.
- **Google Audience** – To filter by Google Audience, type all or part of the Google application ID specified as a condition for IAM role access in your current account. To learn more, see [IAM and AWS STS Condition Context Keys](#).
- **Cognito Audience** – To filter by Cognito Audience, type all or part of the Amazon Cognito identity pool ID specified as a condition for IAM role access in your current account. To learn more, see [IAM and AWS STS Condition Context Keys](#).
- **Caller Account** – The AWS account ID of the account that owns or contains the calling entity, such as an IAM role, user, or account root user. This is used by services calling KMS. To filter by caller account, type all or part of the AWS account ID.
- **Facebook App ID** – To filter by Facebook App ID, type all or part of the Facebook application ID (or site ID) specified as a condition to allow Login with Facebook federation access to an IAM role in your current account. To learn more, see [IAM and AWS STS Condition Context Keys](#).
- **Amazon App ID** – To filter by Amazon App ID, type all or part of the Amazon application ID (or site ID) specified as a condition to allow Login with Amazon federation access to an IAM role in your current account. To learn more, see [IAM and AWS STS Condition Context Keys](#).
- **Lambda Event Source Token** – To filter on Lambda Event Source Token passed in with Alexa integrations, type all or part of the token string.

Archiving findings

When you get a finding for access to a resource that is intentional, such as an IAM role that is used by multiple users for approved workflows, you can archive the finding. When you archive a finding it is cleared from Active findings list, letting you focus on the findings you need to resolve. Archived findings aren't deleted. You can filter the Findings page to display your archived findings, and unarchive them at any time.

To archive findings from the **Findings** page

1. Select the check box next to one or more findings to archive.
2. Choose **Archive**.

A confirmation is displayed at the top of the screen.

To archive findings from the **Findings Details** page.

1. Choose the **Finding ID** for the finding to archive.
2. Choose **Archive**.

A confirmation is displayed at the top of the screen.

To unarchive findings, repeat the preceding steps, but choose **Unarchive** instead of **Archive**. When you unarchive a finding, the status is set to Active.

Resolving findings

To resolve findings generated from access that you did not intend to allow, modify the policy statement to remove the permissions that allow access to the identified resource. For example, for findings on S3 buckets, use the Amazon S3 console to configure the permissions on the bucket. For IAM roles, use the IAM console to [modify the trust policy](#) for the listed IAM role. Use the console for the other supported resources to modify the policy statements that resulted in a generated finding.

After you make a change to resolve a finding, such as modifying a policy applied to an IAM role, Access Analyzer scans the resource again. If the resource is no longer shared outside of your zone of trust, the status of the finding is changed to Resolved. The finding is no longer displayed in the **Active findings** table, and instead is displayed in the **Resolved findings** table.

Note

This does not apply to Error findings. When Access Analyzer is not able to access a resource, it generates an error finding. If you resolve the issue that prevented Access Analyzer from accessing the resource, the error finding is removed completely rather than changing to a resolved finding.

If the changes you made resulted in the resource being shared outside of your zone of trust, but in a different way, such as with a different principal or for a different permission, Access Analyzer generates a new Active finding.

Note

It may take up to 30 minutes after a policy is modified for Access Analyzer to again analyze the resource and then update the finding. Resolved findings are deleted 90 days after the last update to the finding status.

Archive rules

Archive rules automatically archive new findings that meet the criteria you define when you create the rule. You can also apply archive rules retroactively to archive existing findings that meet the archive rule criteria. For example, you can create an archive rule to automatically archive any findings for a specific S3 bucket that you regularly grant access to. Or if you grant access to multiple resources to a specific principal, you can create a rule that automatically archives any new finding generated for access granted to that principal. This lets you focus only on active findings that may indicate a security risk.

Use the information provided in the finding details to identify the specific resource and external entity to use when creating or editing a rule. When you create an archive rule, only new findings that match the rule criteria are automatically archived. Existing findings are not automatically archived. When you create a rule, you can include up to 20 values per criterion in the rule. For a list of filter keys that you can use to create or update an archive rule, see [Access Analyzer filter keys \(p. 556\)](#).

Note

When you create or edit an archive rule, Access Analyzer does not validate the values you include in the filter for the rule. For example, if you add a rule to match an AWS Account, Access Analyzer accepts any value in the field, even if it is not a valid AWS account number.

To create an archive rule

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. Choose **Access analyzer**, then choose **Archive rules**.
3. Choose **Create archive rule**.
4. Enter a name for the rule if you want to change the default name.
5. In the **Rule** section, under **Criteria**, select a property to match for the rule.

6. Choose an operator for the property value, such as **contains**.

The operators available depend on the property you choose.

7. Optionally, add additional values for the property, or add additional criteria for the rule. To ensure your rule won't archive new findings for public access, you can also include the criterion **Public access** and set it to **false**.

To add another value for a criterion, choose **Add another value**. To add another criterion for the rule, choose the **Add** button.

8. When you finish adding criteria and values, choose **Create rule** to apply the rule to new findings only. Choose **Create and archive active findings** to archive new and existing findings based on the rule criteria. In the **Results** section, you can review the list of active findings the archive rule applies to.

For example, to create a rule that automatically archives any findings for S3 buckets: choose **Resource type**, and then choose **is** for the operator. Next choose **S3 bucket** from the **Select resource type** list, and then choose **Add**.

Continue to define criteria to customize the rule as appropriate for your environment, and then choose **Create archive rule**.

If you are creating a new rule and add multiple criteria, you can remove a single criterion from the rule by choosing **Remove this criterion**. You can remove a value added for a criterion by choosing **Remove value**.

To edit an archive rule

1. Choose name of the rule to edit in the **Name**.

You can edit only one archive rule at a time.

2. Add new or remove the existing criteria and values for each criterion. To ensure your rule won't archive new findings for public access, you can also include the criterion **Public access** and set it to **false**.
3. Choose **Save changes** to apply the rule to new findings only. Choose **Save and archive active findings** to archive new and existing findings based on the rule criteria.

To delete an archive rule

1. Select the check box for the rules to delete.

You can delete one, many, or all rules at the same time.

2. Choose **Delete**.
3. Type **delete** in the **Delete archive rule** confirmation dialog, and then choose **Delete**.

The rules are deleted only from the analyzer in the current Region. You must delete archive rules separately for each analyzer that you created in other Regions.

Reference information for AWS IAM Access Analyzer

Use the topics in this section to find detailed reference material for various aspects of AWS IAM Access Analyzer.

Topics

- [Access Analyzer filter keys \(p. 556\)](#)

Access Analyzer filter keys

You can use the filter keys below to define an archive rule [CreateArchiveRule](#), update an archive rule [UpdateArchiveRule](#), or retrieve a list of findings [ListFindings](#).

Criterion	Description	Type	Archive rule	List findings
resource	The ARN uniquely identifying the resource that the external principal has access to. To learn more, see Amazon resource names (ARNs) .	String	✓ Yes	✓ Yes
resourceType	The type of resource that the external principal has access to. AWS::IAM::Role AWS::KMS::Key AWS::Lambda::Function AWS::Lambda::LayerVersion AWS::S3::Bucket AWS::SQS::Queue	String	✓ Yes	✓ Yes
resourceOwnerArn	12 digit AWS account ID that owns the resource. To learn more, see AWS account identifiers .	String	✓ Yes	✓ Yes
isPublic	Indicates whether the finding reports a resource that has a policy that allows public access.	Boolean	✓ Yes	✓ Yes
status	The current status of the finding. ACTIVE ARCHIVED RESOLVED	String	✗ No	✓ Yes
error	Indicates the error reported for the finding.	String	✓ Yes	✓ Yes
principal.AWS	The ARN of the account granted access to the resource in the Principal field of the finding. To learn more, see AWS account identifiers .	String	✓ Yes	✓ Yes
principal.Federated	The ARN of the federated identity that has access to the resource in the finding. To learn more, see AWS Federated identities .	String	✓ Yes	✓ Yes

Criterion	Description	Type	Archive rule	List findings
	more, see Identity providers and federation			
condition.aws:PrincipalARN	The ARN of the principal (IAM user, role, or group) indicated as the condition for resource access. To learn more, see AWS global condition context keys .	String	✓ Yes	✓ Yes
condition.aws:PrincipalOrgID	The organizational identifier of the principal indicated as the condition for resource access. To learn more, see AWS global condition context keys .	String	✓ Yes	✓ Yes
condition.aws:PrincipalOrgPaths	The principal org paths or organizational unit (OU) ID indicated as the condition for resource access. To learn more, see AWS global condition context keys .	String	✓ Yes	✓ Yes
condition.aws:SourceIP	The IP address that allows the principal access to the resource when using the specified IP address. To learn more, see AWS global condition context keys .	IP address	✓ Yes	✓ Yes
condition.aws:SourceVPC	The VPC ID that allows the principal access to the resource when using the specified VPC. To learn more, see AWS global condition context keys .	String	✓ Yes	✓ Yes
condition.aws:UserId	The user ID of the IAM user from an external account indicated as the condition for access to the resource. To learn more, see AWS global condition context keys .	String	✓ Yes	✓ Yes
condition.cognitoIdentity.amazonawsRegion	The Amazon Cognito identity identity.amazonawsRegion specified as a condition for IAM role access in the finding. To learn more, see IAM and AWS STS condition context keys .	String	✓ Yes	✓ Yes
condition.graph.facebookComApplication	The Facebook application ID (or site ID) specified as a condition to allow Login with Facebook federation access to the IAM role in the finding. To learn more, see IAM and AWS STS condition context keys .	String	✓ Yes	✓ Yes

Criterion	Description	Type	Archive rule	List findings
condition.account	The Google application ID specified as a condition for access to the IAM role. To learn more, see IAM and AWS STS condition context keys .	String	✓ Yes	✓ Yes
condition.kms:CallerAccount	The AWS account ID that owns the calling entity (IAM user, role or account root user) used by services calling AWS KMS. To learn more, see Condition keys for AWS Key Management Service .	String	✓ Yes	✓ Yes
condition.www.amazonaws.com:application	The Amazon application ID (or site ID) specified as a condition to allow Login with Amazon federation access to the role. To learn more, see	String	✓ Yes	✓ Yes
id	The ID of the finding.	String	✗ No	✓ Yes

Monitoring AWS IAM Access Analyzer with Amazon EventBridge

Use the information in this topic to learn how to monitor Access Analyzer findings with Amazon EventBridge. EventBridge is the new version of Amazon CloudWatch Events.

Findings events

Access Analyzer sends an event to EventBridge for each generated finding, for a change to the status of an existing finding, and when a finding is deleted. To receive findings and notifications about findings, you must create an event rule in Amazon EventBridge. When you create an event rule, you can also specify a target action to trigger based on the rule. For example, you could create an event rule that triggers an Amazon SNS topic when an event for a new finding is received from Access Analyzer.

Event notification frequency

Access Analyzer sends events for new findings and findings with status updates to EventBridge within about an hour from when the event occurs in your account. Access Analyzer also sends events to EventBridge when a resolved finding is deleted because the retention period has expired. For findings that are deleted because the analyzer that generated them is deleted, the event is sent to EventBridge approximately 24 hours after the analyzer was deleted. When a finding is deleted, the finding status is not changed. Instead, the `isDeleted` attribute is set to `true`.

Example event

The following is an example Access Analyzer event sent to EventBridge. The `id` listed is the ID for the event in EventBridge. To learn more, see [Events and Event Patterns in EventBridge](#).

In the `detail` object, the values for the `accountId` and `region` attributes refer to the account and Region reported in the finding. The `isDeleted` attribute indicates whether the event was from the finding being deleted.

```
{  
    "id": "22222222-dcba-4444-dcba-333333333333",  
    "detail-type": "Access Analyzer Finding",  
    "source": "aws.access-analyzer",  
    "account": "111122223333",  
    "time": "2019-11-21T01:22:33Z",  
    "region": "us-west-2",  
    "resources": [  
        "arn:aws:access-analyzer:us-west-2:111122223333:analyzer/MyAnalyzer"  
    ],  
    "detail": {  
        "version": "1.0",  
        "accountId": "111122223333",  
        "region": "us-west-2",  
        "isDeleted": false,  
        COMPLETE_ACCESS_ANALYZER_GET_FINDING_RESPONSE  
    }  
}
```

The `"id"` is the finding ID. The `"resources"` array is a singleton with the ARN of the analyzer that generated the finding.

The following example shows data for an event that is sent to EventBridge from the `GetFinding` operation of the Access Analyzer API.

```
"version": "0",  
"id": "22222222-dcba-4444-dcba-333333333333",  
"status": "ACTIVE",  
"resourceType": "AWS::S3::Bucket",  
"resource": "arn:aws:s3:::my-bucket",  
"createdAt": "2019-11-20T04:58:50Z",  
"analyzedAt": "2019-11-21T01:22:22Z",  
"updatedAt": "2019-11-21T01:14:07Z",  
"principal": {"AWS": "999988887777"},  
"action": ["s3:GetObject"],  
"condition": {},  
"isPublic": false
```

Access Analyzer also sends events to EventBridge for error findings. An error finding is a finding generated when Access Analyzer can't access a resource it tries to analyze. Events for error findings include an `error` attribute as shown in the following example.

```
"id": "22222222-dcba-4444-dcba-333333333333",  
"status": "ACTIVE",  
"resourceType": "AWS::S3::Bucket",  
"resource": "arn:aws:s3:::my-bucket",  
"error": "ACCESS_DENIED",  
"createdAt": "2019-10-16T19:21:44.244Z",  
"analyzedAt": "2019-10-16T19:21:44.244Z",  
"updatedAt": "2019-10-16T19:21:44.244Z"
```

Creating an event rule using the console

The following procedure describes how to create an event rule using the console.

Open the Amazon EventBridge console at <https://console.aws.amazon.com/events/>.

1. Choose **Create rule**.
2. Enter a **Name** and, optionally, a **Description**.
3. Under **Define pattern** choose **Event pattern**, then choose **Custom pattern**.
4. Copy the following example and then paste it into the **Event pattern** box.

```
{  
    "source": [  
        "aws.access-analyzer"  
    ],  
    "detail-type": [  
        "Access Analyzer Finding"  
    ]  
}
```

5. Choose **Save**.
6. Under **Select targets**, choose a **Target** action for the rule, such as an Amazon SNS topic or AWS Lambda function.
7. Choose the specific SNS topic or Lambda function to use when the target is triggered.

The target is triggered when an event is received that matches the event pattern defined in the rule.
8. Choose **Save** to create the rule.

To learn more about creating rules, see [Creating an EventBridge Rule That Triggers on an Event from an AWS Resource](#).

Creating an event rule using the CLI

1. Use the following to create a rule for Amazon EventBridge using the AWS CLI. Replace the rule name **TestRule** with the name for your rule.

```
aws events put-rule --name TestRule --event-pattern "{\"source\":[\"aws.access-analyzer\"]}"
```

2. You can customize the rule to trigger target actions only for a subset of generated findings, such as findings with specific attributes. The following example demonstrates how to create a rule that triggers a target action only for findings with a status of Active.

```
aws events put-rule --name TestRule --event-pattern "{\"source\":[\"aws.access-analyzer\"],\"detail-type\":[\"Access Analyzer Finding\"],\"detail\":{\"status\":[\"ACTIVE\"]}}"
```

3. To define a Lambda function as a target for the rule you created, use the following example command. Replace the Region and the function name in the ARN as appropriate for your environment.

```
aws events put-targets --rule TestRule --targets Id=1,Arn=arn:aws:lambda:us-east-1:111122223333:function:MyFunction
```

4. Add the permissions required to invoke the rule target. The following example demonstrates how to grant permissions to a Lambda function, following the preceding examples.

```
aws lambda add-permission --function-name MyFunction --statement-id 1 --action 'lambda:InvokeFunction' --principal events.amazonaws.com
```

Integration with AWS Security Hub

[AWS Security Hub](#) provides you with a comprehensive view of your security state in AWS and helps you to check your environment against security industry standards and best practices. Security Hub collects security data from across AWS accounts, services, and supported third-party partner products and helps you to analyze your security trends and identify the highest priority security issues.

The IAM Access Analyzer integration with Security Hub enables you to send findings from Access Analyzer to Security Hub. Security Hub can then include those findings in its analysis of your security posture.

Contents

- [How Access Analyzer sends findings to Security Hub \(p. 561\)](#)
 - [Types of findings that Access Analyzer sends \(p. 561\)](#)
 - [Latency for sending findings \(p. 562\)](#)
 - [Retrying when Security Hub is not available \(p. 562\)](#)
 - [Updating existing findings in Security Hub \(p. 562\)](#)
- [Viewing Access Analyzer findings in Security Hub \(p. 562\)](#)
 - [Interpreting Access Analyzer finding names in Security Hub \(p. 562\)](#)
- [Typical finding from Access Analyzer \(p. 563\)](#)
- [Enabling and configuring the integration \(p. 563\)](#)
- [How to stop sending findings \(p. 564\)](#)

How Access Analyzer sends findings to Security Hub

In Security Hub, security issues are tracked as findings. Some findings come from issues that are detected by other AWS services or by third-party partners. Security Hub also has a set of rules that it uses to detect security issues and generate findings.

Security Hub provides tools to manage findings from across all of these sources. You can view and filter lists of findings and view details for a finding. See [Viewing findings](#) in the [AWS Security Hub User Guide](#). You can also track the status of an investigation into a finding. See [Taking action on findings](#) in the [AWS Security Hub User Guide](#).

All findings in Security Hub use a standard JSON format called the AWS Security Finding Format (ASFF). The ASFF includes details about the source of the issue, the affected resources, and the current status of the finding. See [AWS Security Finding Format \(ASFF\)](#) in the [AWS Security Hub User Guide](#).

IAM Access Analyzer is one of the AWS services that sends findings to Security Hub. Access Analyzer generates a finding when it detects a policy statement that allows an external principal access to a [supported resource \(p. 539\)](#) in your organization or account. Access Analyzer groups all of its findings for a resource and sends a single finding to Security Hub.

Types of findings that Access Analyzer sends

Access Analyzer sends the findings to Security Hub using the [AWS Security Finding Format \(ASFF\)](#). In ASFF, the `Types` field provides the finding type. Findings from Access Analyzer can have the following values for `Types`.

- Effects/Data Exposure/External Access Granted
- Software and Configuration Checks/AWS Security Best Practices/External Access Granted

Latency for sending findings

When Access Analyzer creates a new finding, it is usually sent to Security Hub within 30 minutes. Rarely, and under certain conditions, Access Analyzer is not notified that a policy was added or updated. For example, a change to Amazon S3 account-level block public access settings can take up to 12 hours. Also, if there is a delivery issue with AWS CloudTrail log delivery, the policy change does not trigger a rescan of the resource that was reported in the finding. When this happens, Access Analyzer analyzes the new or updated policy during the next periodic scan, which is within 24 hours.

Retrying when Security Hub is not available

If Security Hub is not available, Access Analyzer retries sending the findings on a periodic basis.

Updating existing findings in Security Hub

After it sends a finding to Security Hub, IAM Access Analyzer sends updates to reflect additional observations of the finding activity to Security Hub. The updates are reflected within the same finding.

The finding for a resource in Security Hub is active if at least one of the findings for the resource in Access Analyzer is active. If all findings in Access Analyzer for a resource are archived or resolved, then the Security Hub finding is archived.

The Security Hub finding is updated when you change the policy access between public and cross-account access. This update can include changes to the type, title, description, and severity of the finding.

Viewing Access Analyzer findings in Security Hub

To view your Access Analyzer findings in Security Hub, choose **See findings** in the **AWS: IAM Access Analyzer** section of the summary page. Alternatively, you can choose **Findings** from the navigation panel. You can then filter the findings to display only IAM Access Analyzer findings by choosing the **Product name:** field with a value of **IAM Access Analyzer**.

Interpreting Access Analyzer finding names in Security Hub

IAM Access Analyzer sends the findings to Security Hub using the AWS Security Finding Format (ASFF). In ASFF, the **Types** field provides the finding type. ASFF types use a different naming scheme than IAM Access Analyzer. The following table includes details about all of the ASFF types associated with IAM Access Analyzer findings as they appear in Security Hub.

ASFF finding type	Security Hub finding title	Description
Effects/Data Exposure/External Access Granted	<resource ARN> allows public access	A resource-based policy attached to the resource allows public access on the resource to all external principals.
Software and Configuration Checks/AWS Security Best Practices/ External Access Granted	<resource ARN> allows cross-account access	A resource-based policy attached to the resource allows cross-account access to external principals outside the zone of trust for the analyzer.

Typical finding from Access Analyzer

Access Analyzer sends findings to Security Hub using the [AWS Security Finding Format \(ASFF\)](#).

Here is an example of a typical finding from Access Analyzer.

```
{  
    "SchemaVersion": "2018-10-08",  
    "Id": "arn:aws:access-analyzer:us-west-2:111122223333:analyzer/my-analyzer/  
arn:aws:s3:::my-bucket",  
    "ProductArn": "arn:aws:securityhub:us-west-2::product/aws/access-analyzer",  
    "GeneratorId": "aws/access-analyzer",  
    "AwsAccountId": "111122223333",  
    "Types": ["Software and Configuration Checks/AWS Security Best Practices/External  
Access Granted"],  
    "CreatedAt": "2020-11-10T16:17:47Z",  
    "UpdatedAt": "2020-11-10T16:43:49Z",  
    "Severity": {  
        "Product": 1,  
        "Label": "LOW",  
        "Normalized": 1  
    },  
    "Title": "AwsS3Bucket/arn:aws:s3:::my-bucket/ allows cross-account access",  
    "Description": "AWS::S3::Bucket/arn:aws:s3:::my-bucket/ allows cross-account access  
from AWS 444455556666",  
    "Remediation": {  
        "Recommendation": {"Text": "If the access isn't intended, it indicates a potential  
security risk. Use the console for the resource to modify or remove the policy that grants  
the unintended access. You can use the Rescan button on the Finding details page in the  
Access Analyzer console to confirm whether the change removed the access. If the access is  
removed, the status changes to Resolved."}  
    },  
    "SourceUrl": "https://console.aws.amazon.com/access-analyzer/home?region=us-west-2#/  
findings/details/dad90d5d-63b4-6575-b0fa-ef9c556ge798",  
    "Resources": [  
        {  
            "Type": "AwsS3Bucket",  
            "Id": "arn:aws:s3:::my-bucket",  
            "Details": {  
                "Other": {  
                    "External Principal Type": "AWS",  
                    "Condition": "none",  
                    "Action Granted": "s3:GetObject,s3:GetObjectVersion",  
                    "External Principal": "444455556666"  
                }  
            }  
        }  
    ],  
    "WorkflowState": "NEW",  
    "Workflow": {"Status": "NEW"},  
    "RecordState": "ACTIVE"  
}
```

Enabling and configuring the integration

To use the integration with Security Hub, you must enable Security Hub. For information on how to enable Security Hub, see [Setting up Security Hub](#) in the *AWS Security Hub User Guide*.

When you enable both Access Analyzer and Security Hub, the integration is enabled automatically. Access Analyzer immediately begins to send findings to Security Hub.

How to stop sending findings

To stop sending findings to Security Hub, you can use either the Security Hub console or the API.

See [Disabling and enabling the flow of findings from an integration \(console\)](#) or [Disabling the flow of findings from an integration \(Security Hub API, AWS CLI\)](#) in the *AWS Security Hub User Guide*.

Logging Access Analyzer API calls with AWS CloudTrail

Access Analyzer is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Access Analyzer. CloudTrail captures all API calls for Access Analyzer as events. The calls captured include calls from the Access Analyzer console and code calls to the Access Analyzer API operations.

If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Access Analyzer. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**.

Using the information collected by CloudTrail, you can determine the request that was made to Access Analyzer, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the [AWS CloudTrail User Guide](#).

Access Analyzer information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in Access Analyzer, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see [Viewing Events with CloudTrail Event History](#).

For an ongoing record of events in your AWS account, including events for Access Analyzer, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- [Overview for Creating a Trail](#)
- [CloudTrail Supported Services and Integrations](#)
- [Configuring Amazon SNS Notifications for CloudTrail](#)
- [Receiving CloudTrail Log Files from Multiple Regions](#) and [Receiving CloudTrail Log Files from Multiple Accounts](#)

All Access Analyzer actions are logged by CloudTrail and are documented in the [IAM Access Analyzer API Reference](#). For example, calls to the `CreateAnalyzer`, `CreateArchiveRule` and `ListFindings` actions generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the [CloudTrail userIdentity Element](#).

Understanding Access Analyzer log file entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the `CreateAnalyzer` operation made by a user named "Alice" on "June 14, 2018".

```
{  
    "eventVersion": "1.05",  
    "userIdentity": {  
        "type": "AssumedRole",  
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",  
        "arn": "arn:aws:iam::111122223333:user/Alice",  
        "accountId": "111122223333",  
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",  
        "sessionContext": {  
            "attributes": {  
                "mfaAuthenticated": "false",  
                "creationDate": "2018-06-14T22:54:20Z"  
            },  
            "sessionIssuer": {  
                "type": "Role",  
                "principalId": "AIDACKCEVSQ6C2EXAMPLE",  
                "arn": "arn:aws:iam::111122223333:user/Alice",  
                "accountId": "111122223333",  
                "userName": "Alice"  
            }  
        },  
        "eventTime": "2018-06-14T22:57:36Z",  
        "eventSource": "access-analyzer.amazonaws.com",  
        "eventName": "CreateAnalyzer",  
        "awsRegion": "us-west-2",  
        "sourceIPAddress": "198.51.100.179",  
        "userAgent": "aws-cli/1.16.205 Python/2.7.16 Darwin/17.7.0 botocore/1.12.195",  
        "requestParameters": {  
            "analyzerName": "test",  
            "type": "ACCOUNT",  
            "clientToken": "11111111-abcd-2222-abcd-222222222222"  
        },  
        "responseElements": {  
            "arn": "arn:aws:access-analyzer:us-west-2:111122223333:analyzer/test"  
        },  
        "requestID": "22222222-dcba-4444-dcba-333333333333",  
        "eventID": "33333333-bcde-5555-bcde-444444444444",  
        "readOnly": false,  
        "eventType": "AwsApiCall",  
        "recipientAccountId": "111122223333"  
    }  
}
```

Troubleshooting IAM

If you encounter access-denied issues or similar difficulties when working with AWS Identity and Access Management (IAM), consult the topics in this section.

Topics

- [Troubleshooting general IAM issues \(p. 566\)](#)
- [Troubleshooting IAM policies \(p. 570\)](#)
- [Troubleshooting U2F security keys \(p. 584\)](#)
- [Troubleshooting IAM roles \(p. 585\)](#)
- [Troubleshooting IAM and Amazon EC2 \(p. 590\)](#)
- [Troubleshooting IAM and Amazon S3 \(p. 593\)](#)
- [Troubleshooting SAML 2.0 federation with AWS \(p. 594\)](#)

Troubleshooting general IAM issues

Use the information here to help you diagnose and fix access-denied or other common issues when you work with AWS Identity and Access Management (IAM).

Issues

- [I can't sign in to my AWS account \(p. 566\)](#)
- [I lost my access keys \(p. 566\)](#)
- [I get "access denied" when I make a request to an AWS service \(p. 567\)](#)
- [I get "access denied" when I make a request with temporary security credentials \(p. 567\)](#)
- [Policy variables aren't working \(p. 568\)](#)
- [Changes that I make are not always immediately visible \(p. 568\)](#)
- [I am not authorized to perform: iam>DeleteVirtualMFADevice \(p. 569\)](#)
- [How do I securely create IAM users? \(p. 570\)](#)

I can't sign in to my AWS account

Verify that you have the correct credentials and that you are using the correct method to sign in. For more information, see [Troubleshooting AWS sign-in or account issues \(p. 69\)](#).

I lost my access keys

Access keys consist of two parts:

- **The access key identifier.** This is not a secret, and can be seen in the IAM console wherever access keys are listed, such as on the user summary page.
- **The secret access key.** This is provided when you initially create the access key pair. Just like a password, it **cannot be retrieved later**. If you lost your secret access key, then you must create a new access key pair. If you already have the [maximum number of access keys \(p. 607\)](#), you must delete an existing pair before you can create another.

For more information, see [Resetting lost or forgotten passwords or access keys for AWS \(p. 110\)](#).

I get "access denied" when I make a request to an AWS service

- Verify that you have the identity-based policy 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 \(p. 438\)](#).
- Are you trying to access a service that supports [resource-based policies \(p. 374\)](#), such as Amazon S3, Amazon SNS, or Amazon SQS? If so, verify that the policy specifies you as a principal and grants you access. If you make a request to a service within your account, either your identity-based policies or the resource-based policies can grant you permission. If you make a request to a service in a different account, then both your identity-based policies and the resource-based policies must grant you permission. To view the services that support resource-based policies, see [AWS services that work with IAM \(p. 611\)](#).
- If your policy includes a condition with a key–value pair, review it carefully. Examples include the `aws:RequestTag/tag-key` (p. 692) global condition key, the AWS KMS `kms:EncryptionContext:encryption_context_key`, and the `ResourceTag/tag-key` condition key supported by multiple services. Make sure that the key name does not match multiple results. Because condition key names are not case sensitive, a condition that checks for a key named `foo` matches `foo`, `Foo`, or `FOO`. If your request includes multiple key–value pairs with key names that differ only by case, then your access might be unexpectedly denied. For more information, see [IAM JSON policy elements: Condition \(p. 641\)](#).
- If you have a [permissions boundary \(p. 365\)](#), verify that the policy that is used for the permissions boundary allows your request. If your identity-based policies allow the request, but your permissions boundary does not, then the request is denied. A permissions boundary controls the maximum permissions that an IAM principal (user or role) can have. Resource-based policies are not limited by permissions boundaries. Permissions boundaries are not common. For more information about how AWS evaluates policies, see [Policy evaluation logic \(p. 666\)](#).
- If you are signing requests manually (without using the [AWS SDKs](#)), verify that you have correctly [signed the request](#).

I get "access denied" when I make a request with temporary security credentials

- First, make sure that you are not denied access for a reason that is unrelated to your temporary credentials. For more information, see [I get "access denied" when I make a request to an AWS service \(p. 567\)](#).
- Verify that the service accepts temporary security credentials, see [AWS services that work with IAM \(p. 611\)](#).
- Verify that your requests are being signed correctly and that the request is well-formed. For details, see your [toolkit](#) documentation or [Using temporary credentials with AWS resources \(p. 313\)](#).
- Verify that your temporary security credentials haven't expired. For more information, see [Temporary security credentials in IAM \(p. 301\)](#).
- Verify that the IAM user or role has the correct permissions. Permissions for temporary security credentials are derived from an IAM user or role. As a result, the permissions are limited to those that are granted to the role whose temporary credentials you have assumed. For more information about how permissions for temporary security credentials are determined, see [Controlling permissions for temporary security credentials \(p. 316\)](#).

- If you assumed a role, your role session might be limited by session policies. When you [request temporary security credentials \(p. 303\)](#) programmatically using AWS STS, you can optionally pass inline or managed [session policies \(p. 353\)](#). Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary credential session for a role. You can pass a single JSON inline session policy document using the `Policy` parameter. You can use the `PolicyArns` parameter to specify up to 10 managed session policies. The resulting session's permissions are the intersection of the role's identity-based policies and the session policies. Alternatively, if your administrator or a custom program provides you with temporary credentials, they might have included a session policy to limit your access.
- If you are a federated user, your session might be limited by session policies. You become a federated user by signing in to AWS as an IAM user and then requesting a federation token. For more information about federated users, see [GetFederationToken—federation through a custom identity broker \(p. 308\)](#). If you or your identity broker passed session policies while requesting a federation token, then your session is limited by those policies. The resulting session's permissions are the intersection of your IAM user identity-based policies and the session policies. For more information about session policies, see [Session policies \(p. 353\)](#).
- 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/*"]  
    }]  
}
```

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.

A `Version` policy element is different from a policy version. The `Version` policy element is used within a policy and defines the version of the policy language. A policy version, on the other hand, is created when you make changes to a customer managed policy in IAM. The changed policy doesn't overwrite the existing policy. Instead, IAM creates a new version of the managed policy. To learn more about the `Version` policy element see [IAM JSON policy elements: Version \(p. 629\)](#). To learn more about policy versions, see [the section called "Versioning IAM policies" \(p. 462\)](#).

- Verify that your policy variables are in the right case. For details, see [IAM policy elements: Variables and tags \(p. 658\)](#).

Changes that I make are not always immediately visible

As a service that is accessed through computers in data centers around the world, IAM uses a distributed computing model called [eventual consistency](#). Any change that you make in IAM (or other AWS services) takes time to become visible from all possible endpoints. Some of the delay results from the time it takes

to send the data from server to server, from replication zone to replication zone, and from Region to Region around the world. IAM also uses caching to improve performance, but in some cases this can add time: The change might not be visible until the previously cached data times out.

You must design your global applications to account for these potential delays. Ensure that they work as expected, even when a change made in one location is not instantly visible at another. Such changes include creating or updating users, groups, roles, or policies. We recommend that you do not include such IAM changes in the critical, high-availability code paths of your application. Instead, make IAM changes in a separate initialization or setup routine that you run less frequently. Also, be sure to verify that the changes have been propagated before production workflows depend on them.

For more information about how some other AWS services are affected by this, consult the following resources:

- **Amazon DynamoDB:** [What is the consistency model of Amazon DynamoDB?](#) in the *DynamoDB FAQ*, and [Read Consistency](#) in the Amazon DynamoDB Developer Guide.
- **Amazon EC2:** [EC2 Eventual Consistency](#) in the *Amazon EC2 API Reference*.
- **Amazon EMR:** [Ensuring Consistency When Using Amazon S3 and Amazon Elastic MapReduce for ETL Workflows](#) in the AWS Big Data Blog
- **Amazon Redshift:** [Managing Data Consistency](#) in the *Amazon Redshift Database Developer Guide*
- **Amazon S3:** [Amazon S3 Data Consistency Model](#) in the *Amazon Simple Storage Service Developer Guide*

I am not authorized to perform: iam:DeleteVirtualMFADevice

You might receive the following error when you attempt to assign or remove a virtual MFA device for yourself or others:

```
User: arn:aws:iam::123456789012:user/Diego is not authorized to perform:  
iam:DeleteVirtualMFADevice on resource: arn:aws:iam::123456789012:mfa/Diego with an  
explicit deny
```

This could happen if someone previously began assigning a virtual MFA device to a user in the IAM console and then cancelled the process. This creates an MFA device for the user in IAM but never activates it. You must delete the existing MFA device before you can associate a new device with the user.

AWS recommends a policy that allows a user to delete their own virtual MFA device only if they are authenticated using MFA. For more information, see [AWS: Allows MFA-authenticated IAM users to manage their own credentials on the My Security Credentials page \(p. 393\)](#).

To fix this issue, an administrator should **not** edit policy permissions. Instead, the administrator must use the AWS CLI or AWS API to remove the existing but deactivated device.

To delete an existing but deactivated MFA device

1. View the virtual MFA devices in your account.
 - AWS CLI: [aws iam list-virtual-mfa-devices](#)
 - AWS API: [ListVirtualMFADevices](#)
2. In the response, locate the ARN of the virtual device for the user you are trying to fix.
3. Delete the device.
 - AWS CLI: [aws iam delete-virtual-mfa-device](#)

- AWS API: [DeleteVirtualMFADevice](#)

How do I securely create IAM users?

If you have employees that require access to AWS, you might choose to create IAM users or [use AWS SSO for authentication](#). If you use IAM, AWS recommends that you create an IAM user and securely communicate the credentials to the employee. If you are not physically located next to your employee, use a secure workflow to communicate credentials to employees.

Use the following workflow to securely create a new user in IAM:

1. [Create a new user](#) using the AWS Management Console. Choose to grant AWS Management Console access with an auto-generated password. If necessary, select the **Users must create a new password at next sign-in** check box. Do not add a permissions policy to the user until after they have changed their password.
2. After the user is added, copy the sign-in URL, user name, and password for the new user. To view the password, choose **Show**.
3. Send the password to your employee using a secure communications method in your company, such as email, chat, or a ticketing system. Separately, provide your users with the IAM user console link and their user name. Tell the employee to confirm that they can sign in successfully before you will grant them permissions.
4. After the employee confirms, add the permissions that they need. As a security best practice, add a policy that requires the user to authenticate using MFA to manage their credentials. For an example policy, see [AWS: Allows MFA-authenticated IAM users to manage their own credentials on the My Security Credentials page](#) (p. 393).

Troubleshooting IAM policies

A [policy](#) (p. 351) is an entity in AWS that, when attached to an identity or resource, defines their permissions. AWS evaluates these policies when a principal, such as a user, makes a request. Permissions in the policies determine whether the request is allowed or denied. Policies are stored in AWS as JSON documents that are attached to principals as *identity-based policies* or to resources as *resource-based policies*. You can attach an identity-based policy to a principal (or identity), such as an IAM group, user, or role. Identity-based policies include AWS managed policies, customer managed policies, and inline policies. You can create and edit customer managed policies in the AWS Management Console using the **Visual editor** tab or the **JSON** tab. When you view a policy in the AWS Management Console, you can see a summary of the permissions that are granted by that policy. You can use the visual editor and policy summaries to help you diagnose and fix common errors encountered while managing IAM policies.

Keep in mind that all IAM policies are stored using syntax that begins with the rules of [JavaScript Object Notation](#) (JSON). You do not have to understand this syntax to create or manage your policies. You can create and edit a policy using the visual editor in the AWS Management Console. To learn more about JSON syntax in IAM policies, see [Grammar of the IAM JSON policy language](#) (p. 679).

Troubleshooting IAM Policy Topics

- [Troubleshoot using the visual editor](#) (p. 571)
 - [Policy restructuring](#) (p. 571)
 - [Choosing a resource ARN in the visual editor](#) (p. 572)
 - [Denying permissions in the visual editor](#) (p. 572)
 - [Specifying multiple services in the visual editor](#) (p. 572)
 - [Reducing the size of your policy in the visual editor](#) (p. 573)

- [Fixing unrecognized services, actions, or resource types in the visual editor \(p. 573\)](#)
- [Troubleshoot using policy summaries \(p. 574\)](#)
 - [Missing policy summary \(p. 574\)](#)
 - [Policy summary includes unrecognized services, actions, or resource types \(p. 575\)](#)
 - [Service does not support IAM policy summaries \(p. 575\)](#)
 - [My policy does not grant the expected permissions \(p. 576\)](#)
- [Troubleshoot policy management \(p. 580\)](#)
 - [Attaching or detaching a policy in an IAM account \(p. 580\)](#)
 - [Changing policies for your IAM identities based on their activity \(p. 580\)](#)
- [Troubleshoot JSON policy documents \(p. 580\)](#)
 - [More than one JSON policy object \(p. 580\)](#)
 - [More than one JSON statement element \(p. 581\)](#)
 - [More than one effect, action, or resource element in a JSON statement element \(p. 582\)](#)
 - [Missing JSON version element \(p. 583\)](#)

Troubleshoot using the visual editor

When you create or edit a customer managed policy, you can use information in the **Visual editor** tab to help you troubleshoot errors in your policy. To view an example of using the visual editor to create a policy, see [the section called “Controlling access to identities” \(p. 378\)](#).

Policy restructuring

When you create a policy, AWS validates, processes, and transforms the policy before storing it. When AWS returns the policy in response to a user query or displays it in the console, AWS transforms the policy back into a human-readable format without changing the permissions granted by the policy. This can result in differences in what you see in the policy visual editor or **JSON** tab: Visual editor permission blocks can be added, removed, or reordered, and content within a block can be optimized. In the **JSON** tab, insignificant white space can be removed, and elements within JSON maps can be reordered. In addition, AWS account IDs within the principal elements can be replaced by the ARN of the AWS account root user. Because of these possible changes, you should not compare JSON policy documents as strings.

When you create a customer managed policy in the AWS Management Console, you can choose to work entirely in the **JSON** tab. If you never make any changes in the **Visual editor** tab and choose **Review policy** from the **JSON** tab, the policy is less likely to be restructured. However, if you create a policy and use the **Visual editor** tab to make any modifications, or if you choose **Review policy** from the **Visual editor** tab, then IAM might restructure the policy to optimize its appearance in the visual editor.

This restructuring exists only in your editing session and is not saved automatically.

If your policy is restructured in your editing session, IAM determines whether to save the restructuring based on the following situations:

On this tab	If you edit your policy	And then choose Review policy from this tab	When you choose Save changes
Visual editor	Edited	Visual editor	The policy is restructured
Visual editor	Edited	JSON	The policy is restructured

On this tab	If you edit your policy	And then choose <i>Review policy</i> from this tab	When you choose <i>Save changes</i>
Visual editor	Not Edited	Visual editor	The policy is restructured
JSON	Edited	Visual editor	The policy is restructured
JSON	Edited	JSON	The policy structure is not changed
JSON	Not Edited	JSON	The policy structure is not changed

IAM might restructure complex policies or policies that have permission blocks or statements that allow multiple services, resource types, or condition keys.

Choosing a resource ARN in the visual editor

When you create or edit a policy using the visual editor, you must first choose a service, and then choose actions from that service. If the service and actions that you selected support choosing [specific resources \(p. 383\)](#), then the visual editor lists the supported resource types. You can then choose **Add ARN** to provide the details about your resource. You can choose from the following options for adding an ARN for a resource type.

- **Use the ARN builder** – Based on the resource type, you might see different fields to build your ARN. You can also choose **Any** to provide permissions for any value for the specified setting. For example, if you selected the Amazon EC2 **Read** access level group, then the actions in your policy support the **instance** resource type. You must provide the **Region**, **Account**, and **InstanceId** values for your resource. If you provide your account ID but choose **Any** for the Region and instance ID, then the policy grants permissions to any instance in your account.
- **Type or paste the ARN** – You can specify resources by their [Amazon Resource Name \(ARN\) \(p. 601\)](#). You can include a wildcard character (*) in any field of the ARN (between each pair of colons). For more information, see [IAM JSON policy elements: Resource \(p. 639\)](#).

Denying permissions in the visual editor

By default, the policy that you create using the visual editor allows the actions that you choose. To deny the chosen actions instead, choose **Switch to deny permissions**. Because requests are *denied by default*, we recommend as a security best practice that you allow permissions to only those actions and resources that a user needs. You should create a statement to deny permissions only if you want to override a permission separately that is allowed by another statement or policy. We recommend that you limit the number of deny permissions to a minimum because they can increase the difficulty of troubleshooting permissions. For more information about how IAM evaluates policy logic, see [Policy evaluation logic \(p. 666\)](#).

Note

By default, only the AWS account root user has access to all the resources in that account. So if you are not signed in as the root user, you must have permissions granted by a policy.

Specifying multiple services in the visual editor

When you use the visual editor to construct a policy, you can select only one service at a time. This is a best practice because the visual editor then allows you to choose from the actions for that one service.

You then choose from the resources supported by that service and the selected actions. This makes it easier to create and troubleshoot your policy.

If you are familiar with the JSON syntax, you can also use a wildcard character (*) to manually specify multiple services. For example, type `Code*` to provide permissions for all services beginning with `Code`, such as `CodeBuild` and `CodeCommit`. However, you must then type the actions and resource ARNs to complete your policy. Additionally, when you save your policy, it might be [restructured \(p. 571\)](#) to include each service in a separate permission block.

Alternatively, to use JSON syntax (such as wildcards) for services, create, edit, and save your policy using the **JSON** tab.

Reducing the size of your policy in the visual editor

When you use the visual editor to create a policy, IAM creates a JSON document to store your policy. You can view this document by switching to the **JSON** tab. If this JSON document exceeds the size limit of a policy, the visual editor displays an error message and does not allow you to review and save your policy. To view the IAM limitation on the size of a managed policy, see [IAM and STS character quotas \(p. 608\)](#).

To reduce the size of your policy in the visual editor, edit your policy or move permission blocks to another policy. The error message includes the number of characters that your policy document contains, and you can use this information to help you reduce the size of your policy.

Fixing unrecognized services, actions, or resource types in the visual editor

When you create or edit a policy in the visual editor, you might see a warning that your policy includes an unrecognized service, action, or resource type.

Note

IAM reviews service names, actions, and resource types for services that support policy summaries. However, your policy summary might include a resource value or condition that does not exist. Always test your policies with the [policy simulator \(p. 445\)](#).

If your policy includes unrecognized services, actions or resource types, one of the following errors has occurred:

- **Preview service** – Services that are in preview do not support the visual editor. If you are participating in the preview, you can ignore the warning and continue, though you must manually type the actions and resource ARNs to complete your policy. Alternatively, you can choose the **JSON** tab to type or paste a JSON policy document.
- **Custom service** – Custom services do not support the visual editor. If you are using a custom service, you can ignore the warning and continue, though you must manually type the actions and resource ARNs to complete your policy. Alternatively, you can choose the **JSON** tab to type or paste a JSON policy document.
- **Service does not support the visual editor** – If your policy includes a generally available (GA) service that does not support the visual editor, you can ignore the warning and continue, though you must manually type the actions and resource ARNs to complete your policy. Alternatively, you can choose the **JSON** tab to type or paste a JSON policy document.

Generally available services are services that are released publicly and are not preview or custom services. If an unrecognized service is generally available and the name is spelled correctly, then the service does not support the visual editor. To learn how to request visual editor or policy summary support for a GA service, see [Service does not support IAM policy summaries \(p. 575\)](#).

- **Action does not support the visual editor** – If your policy includes a supported service with an unsupported action, you can ignore the warning and continue, though you must manually type the

resource ARNs to complete your policy. Alternatively, you can choose the **JSON** tab to type or paste a JSON policy document.

If your policy includes a supported service with an unsupported action, then the service does not fully support the visual editor. To learn how to request visual editor or policy summary support for a GA service, see [Service does not support IAM policy summaries \(p. 575\)](#).

- **Resource type does not support the visual editor** – If your policy includes a supported action with an unsupported resource type, you can ignore the warning and continue. However, IAM cannot confirm that you have included resources for all of your selected actions, and you might see additional warnings.
- **Typo** – When you manually type a service, action, or resource in the visual editor, you can create a policy that includes a typo. As a best practice, use the visual editor by selecting from the list of services and actions, and then complete the resource section according to the prompts. However, if a service does not fully support the visual editor, you might have to manually type parts of your policy.

If you are certain that your policy contains none of the errors above, then your policy might include a typo. Check for misspelled service, action, and resource type names. For example, you might use `s2` instead of `s3` and `ListMyBuckets` instead of `ListAllMyBuckets`. Another common action typo is the inclusion of unnecessary text in ARNs, such as `arn:aws:s3:::*`, or missing colons in actions, such as `iam.CreateUser`. You can evaluate a policy that might include typos by choosing **Review policy** to review the policy summary and confirm whether the policy provides the permissions you intended.

Troubleshoot using policy summaries

You can diagnose and resolve issues related to policy summaries.

Missing policy summary

The IAM console includes *policy summary* tables that describe the access level, resources, and conditions that are allowed or denied for each service in a policy. Policies are summarized in three tables: the [policy summary \(p. 491\)](#), the [service summary \(p. 501\)](#), and the [action summary \(p. 506\)](#). The *policy summary* table includes a list of services and summaries of the permissions that are defined by the chosen policy. You can view the [policy summary \(p. 490\)](#) for any policies that are attached to a user on the **Users** page. You can view the policy summary for managed policies on the **Policies** page. If AWS is unable to render a summary for a policy, then you see the JSON policy document instead of the summary, and receive the following error:

A summary for this policy cannot be generated. You can still view or edit the JSON policy document.

If your policy does not include a summary, one of the following errors has occurred:

- **Unsupported policy element** – IAM does not support generating policy summaries for policies that include one of the following [policy elements \(p. 628\)](#):
 - `Principal`
 - `NotPrincipal`
 - `NotResource`
- **No policy permissions** – If a policy does not provide any effective permissions, then the policy summary cannot be generated. For example, if a policy includes a single statement with the element `"NotAction": "*"`, then it grants access to all actions except "all actions" (*). This means it grants Deny or Allow access to nothing.

Note

You must be careful when using these policy elements such as `NotPrincipal`, `NotAction`, and `NotResource`. For information about using policy elements, see [IAM JSON policy elements reference \(p. 628\)](#).

You can create a policy that does not provide effective permissions if you provide mismatched services and resources. This can occur if you specify actions in one service and resources from another service. In this case, the policy summary does appear. The only indication that there is a problem is that the resource column in the summary can include a resource from a different service. If this column includes a mismatched resource, then you should review your policy for errors. To better understand your policies, always test them with the [policy simulator \(p. 445\)](#).

Policy summary includes unrecognized services, actions, or resource types

In the IAM console, if a [policy summary \(p. 490\)](#) includes a warning symbol (), then the policy might include an unrecognized service, action or resource type. To learn about warnings within a policy summary, see [Policy summary \(list of services\) \(p. 491\)](#).

Note

IAM reviews service names, actions, and resource types for services that support policy summaries. However, your policy summary might include a resource value or condition that does not exist. Always test your policies with the [policy simulator \(p. 445\)](#).

If your policy includes unrecognized services, actions or resource types, one of the following errors has occurred:

- **Preview service** – Services that are in preview do not support policy summaries.
- **Custom service** – Custom services do not support policy summaries.
- **Service does not support summaries** – If your policy includes a generally available (GA) service that does not support policy summaries, then the service is included in the **Unrecognized services** section of the policy summary table. Generally available services are services that are released publicly and are not preview or custom services. If an unrecognized service is generally available and the name is spelled correctly, then the service does not support IAM policy summaries. To learn how to request policy summary support for a GA service, see [Service does not support IAM policy summaries \(p. 575\)](#).
- **Action does not support summaries** – If your policy includes a supported service with an unsupported action, then the action is included in the **Unrecognized actions** section of the service summary table. To learn about warnings within a service summary, see [Service summary \(list of actions\) \(p. 501\)](#).
- **Resource type does not support summaries** – If your policy includes a supported action with an unsupported resource type, then the resource is included in the **Unrecognized resource types** section of the service summary table. To learn about warnings within a service summary, see [Service summary \(list of actions\) \(p. 501\)](#).
- **Typo** – Because the policy validator in AWS checks only that the JSON is syntactically correct, you can create a policy that includes a typo. If you are certain that your policy contains none of the errors above, then your policy might include a typo. Check for misspelled service, action, and resource type names. For example, you might use s2 instead of s3 and ListMyBuckets instead of ListAllMyBuckets. Another common action typo is the inclusion of unnecessary text in ARNs, such as arn:aws:s3: : : *, or missing colons in actions, such as iam.CreateUser. You can evaluate a policy that might include typos by using the [policy simulator \(p. 445\)](#) to confirm whether the policy provides the permissions you intended.

Service does not support IAM policy summaries

When a generally available (GA) service or action is not recognized by IAM policy summaries or the visual editor, it is possible that the service does not support these features. Generally available services are services that are released publicly and are not previewed or custom services. If an unrecognized service is generally available and the name is spelled correctly, then the service does not support these features.

If your policy includes a supported service with an unsupported action, then the service does not fully support IAM policy summaries.

To request that a service add IAM policy summary or visual editor support

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. Locate the policy that includes the unsupported service:
 - If the policy is a managed policy, choose **Policies** in the navigation pane. In the list of policies, choose the name of the policy that you want to view.
 - If the policy is an inline policy attached to the user, choose **Users** in the navigation pane. In the list of users, choose the name of the user whose policy you want to view. In the table of policies for the user, expand the header for the policy summary that you want to view.
3. In the left side on the AWS Management Console footer, choose **Feedback**. In the **Tell us about your experience:** box, type **I request that the <ServiceName> service add support for IAM policy summaries and the visual editor.** If you want more than one service to support summaries, type **I request that the <ServiceName1>, <ServiceName2>, and <ServiceName3> services add support for IAM policy summaries and the visual editor.**

To request that a service add IAM policy summary support for a missing action

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. Locate the policy that includes the unsupported service:
 - If the policy is a managed policy, choose **Policies** in the navigation pane. In the list of policies, choose the name of the policy that you want to view.
 - If the policy is an inline policy attached to the user, choose **Users** in the navigation pane. In the list of users, choose the name of the user whose policy you want to view. In the table of policies for the user, choose the name of the policy that you want to view to expand the policy summary.
3. In the policy summary, choose the name of the service that includes an unsupported action.
4. In the left side on the AWS Management Console footer, choose **Feedback**. In the **Tell us about your experience:** box, type **I request that the <ServiceName> service add IAM policy summary and the visual editor support for the <ActionName> action.** If you want to report more than one unsupported action, type **I request that the <ServiceName> service add IAM policy summary and the visual editor support for the <ActionName1>, <ActionName2>, and <ActionName3> actions.**

To request that a different service includes missing actions, repeat the last three steps.

My policy does not grant the expected permissions

To assign permissions to a user, group, role, or resource, you create a *policy*, which is a document that defines permissions. The policy document includes the following elements:

- **Effect** – whether the policy allows or denies access
- **Action** – the list of actions that are allowed or denied by the policy
- **Resource** – the list of resources on which the actions can occur
- **Condition (Optional)** – the circumstances under which the policy grants permission

To learn about these and other policy elements, see [IAM JSON policy elements reference \(p. 628\)](#).

To grant access, your policy must define an action with a supported resource. If your policy also includes a condition, that condition must include a [global condition key \(p. 692\)](#) or must apply to the action.

To learn which resources are supported by an action, see the [AWS documentation](#) for your service. To learn which conditions are supported by an action, see [Actions, Resources, and Condition Keys for AWS Services](#).

To learn whether your policy defines an action, resource, or condition that does not grant permissions, you can view the [policy summary \(p. 491\)](#) for your policy using the IAM console at <https://console.aws.amazon.com/iam/>. You can use policy summaries to identify and correct problems in your policy.

There are several reasons why an element might not grant permissions despite being defined in the IAM policy:

- [An action is defined without an applicable resource \(p. 577\)](#)
- [A resource is defined without an applicable action \(p. 577\)](#)
- [A condition is defined without an applicable action \(p. 578\)](#)

To view examples of policy summaries that include warnings, see [the section called “Policy summary \(list of services\)” \(p. 491\)](#).

An action is defined without an applicable resource

The policy below defines all `ec2:Describe*` actions and defines a specific resource. None of the `ec2:Describe` actions are granted because none of these actions support resource-level permissions. Resource-level permissions mean that the action supports resources using [ARNs \(p. 601\)](#) in the policy's [Resource \(p. 639\)](#) element. If an action does not support resource-level permissions, then that statement in the policy must use a wildcard (*) in the Resource element. To learn which services support resource-level permissions, see [AWS services that work with IAM \(p. 611\)](#).

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {"Effect": "Allow",  
         "Action": "ec2:Describe*",  
         "Resource": "arn:aws:ec2:us-east-2:ACCOUNT-ID:instance/*"  
     }]  
}
```

This policy does not provide any permissions, and the policy summary includes the following error:

This policy does not grant any permissions. To grant access, policies must have an action that has an applicable resource or condition.

To fix this policy, you must use * in the Resource element.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {"Effect": "Allow",  
         "Action": "ec2:Describe*",  
         "Resource": "*"  
     }]  
}
```

A resource is defined without an applicable action

The policy below defines an Amazon S3 bucket resource but does not include an S3 action that can be performed on that resource. This policy also grants full access to all Amazon CloudFront actions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "cloudfront:*",
            "Resource": [
                "arn:aws:cloudfront:*,",
                "arn:aws:s3:::examplebucket"
            ]
        }
    ]
}
```

This policy provides permissions for all CloudFront actions. But because the policy defines the S3 examplebucket resource without defining any S3 actions, the policy summary includes the following warning:

This policy defines some actions, resources, or conditions that do not provide permissions. To grant access, policies must have an action that has an applicable resource or condition.

To fix this policy to provide S3 bucket permissions, you must define S3 actions that can be performed on a bucket resource.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudfront:*,",
                "s3:CreateBucket",
                "s3>ListBucket*",
                "s3:PutBucket*",
                "s3:GetBucket*"
            ],
            "Resource": [
                "arn:aws:cloudfront:*,",
                "arn:aws:s3:::examplebucket"
            ]
        }
    ]
}
```

Alternately, to fix this policy to provide only CloudFront permissions, remove the S3 resource.

A condition is defined without an applicable action

The policy below defines two Amazon S3 actions for all S3 resources, if the S3 prefix equals custom and the version ID equals 1234. However, the s3:VersionId condition key is used for object version tagging and is not supported by the defined bucket actions. To learn which conditions are supported by an action, see [Actions, Resources, and Condition Keys for AWS Services](#) and follow the link to the service documentation for condition keys.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3>ListBucketVersions",
                "s3>ListBucket"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "s3:VersionId": "1234"
                }
            }
        }
    ]
}
```

```
        "Condition": {
            "StringEquals": [
                "s3:prefix": [
                    "custom"
                ],
                "s3:VersionId": [
                    "1234"
                ]
            ]
        }
    }
}
```

This policy provides permissions for the `s3>ListBucketVersions` action and the `s3>ListBucket` action if the bucket name includes the custom prefix. But because the `s3:VersionId` condition is not supported by any of the defined actions, the policy summary includes the following error:

This policy does not grant any permissions. To grant access, policies must have an action that has an applicable resource or condition.

To fix this policy to use S3 object version tagging, you must define an S3 action that supports the s3:VersionId condition key.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3>ListBucketVersions",  
                "s3>ListBucket",  
                "s3GetObjectVersion"  
            ],  
            "Resource": "*",  
            "Condition": {  
                "StringEquals": {  
                    "s3:prefix": [  
                        "custom"  
                    ],  
                    "s3:VersionId": [  
                        "1234"  
                    ]  
                }  
            }  
        }  
    ]  
}
```

This policy provides permissions for every action and condition in the policy. However, the policy still does not provide any permissions because there is no case where a single action matches both conditions. Instead, you must create two separate statements that each include only actions with the conditions to which they apply.

To fix this policy, create two statements. The first statement includes the actions that support the `s3:prefix` condition, and the second statement includes the actions that support the `s3:VersionId` condition.

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

```
        "Effect": "Allow",
        "Action": [
            "s3>ListBucketVersions",
            "s3>ListBucket"
        ],
        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "s3:prefix": "custom"
            }
        }
    },
    {
        "Effect": "Allow",
        "Action": "s3GetObjectVersion",
        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "s3:VersionId": "1234"
            }
        }
    }
]
```

Troubleshoot policy management

You can diagnose and resolve issues relating to policy management.

Attaching or detaching a policy in an IAM account

Some AWS managed policies are linked to a service. These policies are used only with a [service-linked role \(p. 168\)](#) for that service. In the IAM console, when you view the **Summary** page for a policy, the page includes a banner to indicate that the policy is linked to a service. You cannot attach this policy to a user, group, or role within IAM. When you create a service-linked role for the service, this policy is automatically attached to your new role. Because the policy is required, you cannot detach the policy from the service-linked role.

Changing policies for your IAM identities based on their activity

You can update policies for your IAM identities (users, groups, and roles) based on their activity. To do this, view your account's events in CloudTrail **Event history**. CloudTrail event logs include detailed event information that you can use to change the policy's permissions. You might find that a user or role is attempting to perform an action in AWS and that request is denied. In that case, you can consider whether the user or role should have permission to perform the action. If so, you can add the action and even the ARN of the resource that they attempted to access to their policy. Alternatively, if the user or role has permissions that they are not using, you might consider removing those permissions from their policy. Make sure that your policies grant the [least privilege \(p. 529\)](#) that is needed to perform only the necessary actions. For more information about using CloudTrail, see [Viewing CloudTrail Events in the CloudTrail Console](#) in the *AWS CloudTrail User Guide*.

Troubleshoot JSON policy documents

You can diagnose and resolve issues relating to JSON policy documents.

More than one JSON policy object

An IAM policy must consist of one and only one JSON object. You denote an object by placing {} braces around it. Although you can nest other objects within a JSON object by embedding additional {} braces

within the outer pair, a policy can contain only one outermost pair of {} braces. The following example is incorrect because it contains two objects at the top level (called out in *red*):

```
{
    "Version": "2012-10-17",
    "Statement":
    {
        "Effect": "Allow",
        "Action": "ec2:Describe*",
        "Resource": "*"
    }
}

{
    "Statement": {
        "Effect": "Allow",
        "Action": "s3:*",
        "Resource": "arn:aws:s3:::my-bucket/*"
    }
}
```

You can, however, meet the intention of the previous example with the use of correct policy grammar. Instead of including two complete policy objects each with its own Statement element, you can combine the two blocks into a single Statement element. The Statement element has an array of two objects as its value, as shown in the following example (called out in **bold**):

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "ec2:Describe*",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "s3:*",
            "Resource": "arn:aws:s3:::my-bucket/*"
        }
    ]
}
```

More than one JSON statement element

This error might at first appear to be a variation on the previous section. However, syntactically it is a different type of error. The following example has only one policy object as denoted by a single pair of {} braces at the top level. However, that object contains two Statement elements within it.

An IAM policy must contain only one Statement element, consisting of the name (Statement) appearing to the left of a colon, followed by its value on the right. The value of a Statement element must be an object, denoted by {} braces, containing one Effect element, one Action element, and one Resource element. The following example is incorrect because it contains two Statement elements in the policy object (called out in *red*):

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": "ec2:Describe*",
        "Resource": "*"
    },
}
```

```
    "Statement": {
        "Effect": "Allow",
        "Action": "s3:*",
        "Resource": "arn:aws:s3:::my-bucket/*"
    }
}
```

A value object can be an array of multiple value objects. To solve this problem, combine the two Statement elements into one element with an object array, as shown in the following example (called out in **bold**):

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "ec2:Describe*",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "s3:*",
            "Resource": "arn:aws:s3:::my-bucket/*"
        }
    ]
}
```

The value of the Statement element is an object array. The array in this example consists of two objects, each of which is by itself a correct value for a Statement element. Each object in the array is separated by commas.

More than one effect, action, or resource element in a JSON statement element

On the value side of the Statement name/value pair, the object must consist of only one Effect element, one Action element, and one Resource element. The following policy is incorrect because it has two Effect elements in the value object of the Statement:

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Deny",
        "Effect": "Allow",
        "Action": "ec2:*",
        "Resource": "*"
    }
}
```

Note

The policy engine does not allow such errors in new or edited policies. However, the policy engine continues to permit policies that were saved before the engine was updated. The behavior of existing policies with the error is as follows:

- Multiple Effect elements: only the last Effect element is observed. The others are ignored.
- Multiple Action elements: all Action elements are combined internally and treated as if they were a single list.
- Multiple Resource elements: all Resource elements are combined internally and treated as if they were a single list.

The policy engine does not allow you to save any policy with syntax errors. You must correct the errors in the policy before you can save it. The [Policy Validator \(p. 444\)](#) tool can help you to find all older policies with errors and can recommend corrections for them.

In each case, the solution is to remove the incorrect extra element. For `Effect` elements, this is straightforward: if you want the previous example to *deny* permissions to Amazon EC2 instances, then you must remove the line `"Effect": "Allow"`, from the policy, as follows:

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Deny",
        "Action": "ec2:*",
        "Resource": "*"
    }
}
```

However, if the duplicate element is `Action` or `Resource`, then the resolution can be more complicated. You might have multiple actions that you want to allow (or deny) permission to, or you might want to control access to multiple resources. For example, the following example is incorrect because it has multiple `Resource` elements (called out in *red*):

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": "s3:*",
        "ResourceResource

```

Each of the required elements in a `Statement` element's value object can be present only once. The solution is to place each value in an array. The following example illustrates this by making the two separate resource elements into one `Resource` element with an array as the value object (called out in **bold**):

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": "s3:*",
        "Resource": [
            "arn:aws:s3:::my-bucket",
            "arn:aws:s3:::my-bucket/*"
        ]
    }
}
```

Missing JSON version element

A `Version` policy element is different from a policy version. The `Version` policy element is used within a policy and defines the version of the policy language. A policy version, on the other hand, is created when you make changes to a customer managed policy in IAM. The changed policy doesn't overwrite the existing policy. Instead, IAM creates a new version of the managed policy. To learn more about the `Version` policy element see [IAM JSON policy elements: Version \(p. 629\)](#). To learn more about policy versions, see [the section called "Versioning IAM policies" \(p. 462\)](#).

As AWS features evolve, new capabilities are added to IAM policies to support those features. Sometimes, an update to the policy syntax includes a new version number. If you use newer features

of the policy grammar in your policy, then you must tell the policy parsing engine which version you are using. The default policy version is "2008-10-17." If you want to use any policy feature that was introduced later, then you must specify the version number that supports the feature you want. We recommend that you *always* include the latest policy syntax version number, which is currently "Version": "2012-10-17". For example, the following policy is incorrect because it uses a policy variable \${...} in the ARN for a resource. But it fails to specify a policy syntax version that supports policy variables (called out in *red*):

```
{  
  "Statement":  
  {  
    "Action": "iam:*AccessKey*",  
    "Effect": "Allow",  
    "Resource": "arn:aws:iam::123456789012:user/${aws:username}"  
  }  
}
```

Adding a `Version` element at the top of the policy with the value 2012-10-17, the first IAM API version that supports policy variables, solves this problem (called out in **bold**):

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

Troubleshooting U2F security keys

Use the information here to help you diagnose common issues that you might encounter when working with U2F security keys.

Topics

- [I can't enable my U2F security key \(p. 584\)](#)
- [I can't sign in using my U2F security key \(p. 585\)](#)
- [I lost or broke my U2F key \(p. 585\)](#)
- [Other issues \(p. 585\)](#)

I can't enable my U2F security key

Consult the following solutions depending on your status as an IAM user or system administrator

IAM users

If you can't enable your U2F security key, check the following:

- Are you using a supported configuration?

For information on devices and browsers you can use with U2F and AWS, see [Supported configurations for using U2F security keys \(p. 121\)](#).

- Are you using Mozilla Firefox?

Most Firefox versions that support U2F do not enable support by default. To enable support for U2F in Firefox, do the following:

1. From the Firefox address bar, type **about:config**.
 2. In the Search bar of the screen that opens, type **u2f**.
 3. Choose **security.webauth.u2f** and change its value to **true**.
- Are you using any browser plugins?

AWS does not support the use of plugins to add U2F browser support. Instead, use a browser that offers native support of the U2F standard.

Even if you're using a supported browser, you may have a plugin that is incompatible with U2F. An incompatible plugin may prevent you from enabling and using your U2F security key. You should disable any plugins that might be incompatible and restart your browser. Then retry enabling the U2F security key.

- Do you have the appropriate permissions?

If you don't have any of the above compatibility issues, you may not have the appropriate permissions. Contact your system administrator.

System administrators

If you're an administrator and your IAM users can't enable their U2F security keys despite using a supported configuration, make sure they have the appropriate permissions. For a detailed example, see [IAM Tutorial: Enable users to manage their credentials and MFA settings \(p. 60\)](#).

I can't sign in using my U2F security key

If you're an IAM user and you can't sign in to the AWS Management Console using U2F, first see [Supported configurations for using U2F security keys \(p. 121\)](#). If you're using a supported configuration but cannot sign in, contact your system administrator for assistance.

I lost or broke my U2F key

Only *one* MFA device (virtual, U2F security key, or hardware) is assigned to a user at a time. Replacing a U2F security key is similar to replacing a hardware MFA device. For information on what to do if you lose or break any type of MFA device, see [What if an MFA device is lost or stops working? \(p. 136\)](#).

Other issues

If you have an issue with U2F security keys that is not covered here, do one of the following:

- IAM users: Contact your system administrator.
- AWS account root users: Contact [AWS Support](#).

Troubleshooting IAM roles

Use the information here to help you diagnose and fix common issues that you might encounter when working with IAM roles.

Topics

- [I can't assume a role \(p. 586\)](#)
- [A new role appeared in my AWS account \(p. 587\)](#)

- I can't edit or delete a role in my AWS account (p. 587)
- I'm not authorized to perform: iam:PassRole (p. 587)
- Why can't I assume a role with a 12-hour session? (AWS CLI, AWS API) (p. 588)
- I receive an error when I try to switch roles in the IAM console (p. 588)
- My role has a policy that allows me to perform an action, but I get "access denied" (p. 588)
- The service did not create the role's default policy version (p. 589)
- There is no use case for a service role in the console (p. 590)

I can't assume a role

Check the following:

- Make sure to use the exact name of your role, because role names are case sensitive.
- Verify that your IAM policy grants you permission 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 AssumeRole action. In addition, the Resource element of your IAM policy must specify the role that you want to assume. For example, the Resource element can specify a role by its Amazon Resource Name (ARN) or by a wildcard (*). For example, at least one policy applicable to you must grant permissions similar to the following:

```
"Effect": "Allow",
"Action": "sts:AssumeRole",
"Resource": "arn:aws:iam::account_id_number:role/role-name-you-want-to-assume"
```

- Verify that your IAM identity is tagged with any tags that the IAM policy requires. For example, in the following policy permissions, the Condition element requires that you, as the principal requesting to assume the role, must have a specific tag. You must be tagged with department = HR or department = CS. Otherwise, you cannot assume the role. To learn about tagging IAM users and roles, see the section called "Tagging users and roles" (p. 289).

```
"Effect": "Allow",
"Action": "sts:AssumeRole",
"Resource": "*",
"Condition": {"StringEquals": {"aws:PrincipalTag/department": [
    "HR",
    "CS"
]}}
```

- Verify that you meet all the conditions that are specified in the role's trust policy. A Condition can specify an expiration date, an external ID, or that a request must come only from specific IP addresses. Consider the following example: If the current date is any time after the specified date, then the policy never matches and cannot grant you the permission to assume the role.

```
"Effect": "Allow",
"Action": "sts:AssumeRole",
"Resource": "arn:aws:iam::account_id_number:role/role-name-you-want-to-assume"
"Condition": {
    "DateLessThan" : {
        "aws:CurrentTime" : "2016-05-01T12:00:00Z"
    }
}
```

- Verify that the AWS account from which you are calling AssumeRole is a trusted entity for the role that you are assuming. Trusted entities are defined as a Principal in a role's trust policy. The following example is a trust policy that is attached to the role that you want to assume. In this example, the account ID with the IAM user that you signed in with must be 123456789012. If your

account number is not listed in the **Principal** element of the role's trust policy, then you cannot assume the role. It does not matter what permissions are granted to you in access policies. Note that the example policy limits permissions to actions that occur between July 1, 2017 and December 31, 2017 (UTC), inclusive. If you log in before or after those dates, then the policy does not match, and you cannot assume the role.

```
"Effect": "Allow",
"Principal": { "AWS": "arn:aws:iam::123456789012:root" },
"Action": "sts:AssumeRole",
"Condition": {
    "DateGreaterThanOrEqualTo": {"aws:CurrentTime": "2017-07-01T00:00:00Z"},
    "DateLessThanOrEqualTo": {"aws:CurrentTime": "2017-12-31T23:59:59Z"}
}
```

A new role appeared in my AWS account

Some AWS services require that you use a unique type of service role that is linked directly to the service. This [service-linked role \(p. 168\)](#) is predefined by the service and includes all the permissions that the service requires. This makes setting up a service easier because you don't have to manually add the necessary permissions. For general information about service-linked roles, see [Using service-linked roles \(p. 213\)](#).

You might already be using a service when it begins supporting service-linked roles. If so, you might receive an email telling you about a new role in your account. This role includes all the permissions that the service needs to perform actions on your behalf. You don't need to take any action to support this role. However, you should not delete the role from your account. Doing so could remove permissions that the service needs to access AWS resources. You can view the service-linked roles in your account by going to the **IAM Roles** page of the IAM console. Service-linked roles appear with **(Service-linked role)** in the **Trusted entities** column of the table.

For information about which services support service-linked roles, see [AWS services that work with IAM \(p. 611\)](#) and look for the services that have **Yes** in the **Service-Linked Role** column. For information about using the service-linked role for a service, choose the **Yes** link.

I can't edit or delete a role in my AWS account

You cannot delete or edit the permissions for a [service-linked role \(p. 168\)](#) in IAM. These roles include predefined trusts and permissions that are required by the service in order to perform actions on your behalf. You can use the IAM console, AWS CLI, or API to edit only the description of a service-linked role. You can view the service-linked roles in your account by going to the **IAM Roles** page in the console. Service-linked roles appear with **(Service-linked role)** in the **Trusted entities** column of the table. A banner on the role's **Summary** page also indicates that the role is a service-linked role. You can manage and delete these roles only through the linked service, if that service supports the action. Be careful when modifying or deleting a service-linked role because doing so could remove permissions that the service needs to access AWS resources.

For information about which services support service-linked roles, see [AWS services that work with IAM \(p. 611\)](#) and look for the services that have **Yes** in the **Service-Linked Role** column.

I'm not authorized to perform: iam:PassRole

When you create a service-linked role, you must have permission to pass that role to the service. Some services automatically create a service-linked role in your account when you perform an action in that service. For example, Amazon EC2 Auto Scaling creates the `AWSServiceRoleForAutoScaling` service-linked role for you the first time that you create an Auto Scaling group. If you try to create an Auto Scaling group without the `PassRole` permission, you receive the following error:

`ClientError: An error occurred (AccessDenied) when calling the PutLifecycleHook operation: User: arn:aws:sts::111122223333:assumed-role/Testrole/Diego is not authorized to perform: iam:PassRole on resource: arn:aws:iam::111122223333:role/aws-service-role/autoscaling.amazonaws.com/AWSServiceRoleForAutoScaling`

To fix this error, ask your administrator to add the `iam:PassRole` permission for you.

To learn which services support service-linked roles, see [AWS services that work with IAM \(p. 611\)](#). To learn whether a service automatically creates a service-linked role for you, choose the **Yes** link to view the service-linked role documentation for the service.

Why can't I assume a role with a 12-hour session? (AWS CLI, AWS API)

When you use the AWS STS `AssumeRole*` API or `assume-role*` CLI operations to assume a role, you can specify a value for the `DurationSeconds` parameter. You can specify a value from 900 seconds (15 minutes) up to the **Maximum session duration** setting for the role. If you specify a value higher than this setting, the operation fails. This setting can have a maximum value of 12 hours. For example, if you specify a session duration of 12 hours, but your administrator set the maximum session duration to 6 hours, your operation fails. To learn how to view the maximum value for your role, see [View the maximum session duration setting for a role \(p. 248\)](#).

If you use [role chaining \(p. 169\)](#) (using a role to assume a second role), your session is limited to a maximum of one hour. If you then use the `DurationSeconds` parameter to provide a value greater than one hour, the operation fails.

I receive an error when I try to switch roles in the IAM console

The information you enter on the **Switch Role** page must match the information for the role. Otherwise, the operation fails and you receive the following error:

`Invalid information in one or more fields. Check your information or contact your administrator.`

If you receive this error, confirm that the following information is correct:

- **Account ID or alias** – The AWS account ID is a 12-digit number. Your account might have an alias, which is a friendly identifier such as your company name that can be used instead of your AWS account ID. You can use either the account ID or the alias in this field.
- **Role name** – Role names are case sensitive. The account ID and role name must match what is configured for the role.

If you continue to receive an error message, contact your administrator to verify the previous information. The role trust policy or the IAM user policy might limit your access. Your administrator can verify the permissions for these policies.

My role has a policy that allows me to perform an action, but I get "access denied"

Your role session might be limited by session policies. When you [request temporary security credentials \(p. 303\)](#) programmatically using AWS STS, you can optionally pass inline or managed

[session policies \(p. 353\)](#). Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary credential session for a role. You can pass a single JSON inline session policy document using the `Policy` parameter. You can use the `PolicyArns` parameter to specify up to 10 managed session policies. The resulting session's permissions are the intersection of the role's identity-based policies and the session policies. Alternatively, if your administrator or a custom program provides you with temporary credentials, they might have included a session policy to limit your access.

The service did not create the role's default policy version

A service role is a role that a service assumes to perform actions in your account on your behalf. When you set up some AWS service environments, you must define a role for the service to assume. In some cases, the service creates the service role and its policy in IAM for you. Although you can modify or delete the service role and its policy from within IAM, AWS does not recommend this. The role and policy are intended for use only by that service. If you edit the policy and set up another environment, when the service tries to use the same role and policy, the operation can fail.

For example, when you use AWS CodeBuild for the first time, the service creates a role named `codebuild-RWBCore-service-role`. That service role uses the policy named `codebuild-RWBCore-managed-policy`. If you edit the policy, it creates a new version and saves that version as the default version. If you perform a subsequent operation in AWS CodeBuild, the service might try to update the policy. If it does, you receive the following error:

```
codebuild.amazon.com did not create the default version (V2) of the codebuild-RWBCore-managed-policy policy that is attached to the codebuild-RWBCore-service-role role. To continue, detach the policy from any other identities and then delete the policy and the role.
```

If you receive this error, you must make changes in IAM before you can continue with your service operation. First, set the default policy version to V1 and try the operation again. If V1 was previously deleted, or if choosing V1 doesn't work, then clean up and delete the existing policy and role.

For more information on editing managed policies, see [Editing customer managed policies \(console\) \(p. 466\)](#). For more information about policy versions, see [Versioning IAM policies \(p. 462\)](#).

To delete a service role and its 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, choose **Policies**.
3. In the list of policies, choose the name of the policy that you want to delete.
4. Choose the **Policy usage** tab to view which IAM users, groups, or roles use this policy. If any of these identities use the policy, complete the following tasks:
 - a. Create a new managed policy with the necessary permissions. To ensure that the identities have the same permissions before and after your actions, copy the JSON policy document from the existing policy. Then create the new managed policy and paste the JSON document as described in [Creating Policies on the JSON Tab \(p. 440\)](#).
 - b. For each affected identity, attach the new policy and then detach the old one. For more information, see [Adding and removing IAM identity permissions \(p. 454\)](#).
5. In the navigation pane, choose **Roles**.
6. In the list of roles, choose the name of the role that you want to delete.
7. Choose the **Trust relationships** tab to view which entities can assume the role. If any entity other than the service is listed, complete the following tasks:

- a. [Create a new role \(p. 222\)](#) that trusts those entities.
 - b. The policy that you created in the previous step. If you skipped that step, create the new managed policy now.
 - c. Notify anyone who was assuming the role that they can no longer do so. Provide them with information about how to assume the new role and have the same permissions.
8. [Delete the policy \(p. 470\)](#).
 9. [Delete the role \(p. 284\)](#).

There is no use case for a service role in the console

Some services require that you manually create a service role to grant the service permissions to perform actions on your behalf. If the service is not listed in the IAM console, you must manually list the service as the trusted principal. If the documentation for the service or feature that you are using does not include instructions for listing the service as the trusted principal, provide feedback for the page.

To manually create a service role, you must know the [service principal \(p. 633\)](#) for the service that will assume the role. A service principal is an identifier that is used to grant permissions to a service. The service principal is defined by the service.

You can find the service principal for some services by checking the following:

1. Open [AWS services that work with IAM \(p. 611\)](#).
2. Check whether the service has **Yes** in the **Service-linked roles** column.
3. Choose the **Yes** link to view the service-linked role documentation for that service.
4. Find the Service-linked role permissions section for that service to view the [service principal \(p. 633\)](#).

You can manually create a service role using [AWS CLI commands \(p. 232\)](#) or [AWS API operations \(p. 234\)](#). To manually create a service role using the IAM console, complete the following tasks:

1. Create an IAM role using your account ID. Do not attach a policy or grant any permissions. For details, see [Creating a role to delegate permissions to an IAM user \(p. 221\)](#).
2. Open the role and edit the trust relationship. Instead of trusting the account, the role must trust the service. For example, update the following **Principal** element:

```
"Principal": { "AWS": "arn:aws:iam::123456789012:root" }
```

Change the principal to the value for your service, such as IAM.

```
"Principal": { "Service": "iam.amazonaws.com" }
```

3. Add the permissions that the service requires by attaching permissions policies to the role.
4. Return to the service that requires the permissions and use the documented method to notify the service about the new service role.

Troubleshooting IAM and Amazon EC2

Use the information here to help you troubleshoot and fix access denied or other issues that you might encounter when working with Amazon EC2 and IAM.

Topics

- When attempting to launch an instance, I don't see the role I expected to see in the Amazon EC2 console **IAM Role** list (p. 591)
- The credentials on my instance are for the wrong role (p. 591)
- When I attempt to call the `AddRoleToInstanceProfile`, I get an `AccessDenied` error (p. 591)
- Amazon EC2: When I attempt to launch an instance with a role, I get an `AccessDenied` error (p. 592)
- I can't access the temporary security credentials on my EC2 instance (p. 592)
- What do the errors from the info document in the IAM subtree mean? (p. 593)

When attempting to launch an instance, I don't see the role I expected to see in the Amazon EC2 console **IAM Role** list

Check the following:

- If you are signed in as an IAM user, verify that you have permission to call `ListInstanceProfiles`. For information about the permissions necessary to work with roles, see "Permissions Required for Using Roles with Amazon EC2" in [Using an IAM role to grant permissions to applications running on Amazon EC2 instances \(p. 263\)](#). For information about adding permissions to a user, see [Managing IAM policies \(p. 438\)](#).

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 [Using instance profiles \(p. 270\)](#).

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. An instance profile can contain only one IAM role, and that limit cannot be increased.

The credentials on my instance are for the wrong role

The role in the instance profile might have been replaced recently. If so, your application will need to wait for the next automatically scheduled credential rotation before credentials for your role become available.

When I attempt to call the `AddRoleToInstanceProfile`, I get an `AccessDenied` error

If you are making requests as an IAM user, verify that you have the following permissions:

- `iam:AddRoleToInstanceProfile` with the resource matching the instance profile ARN (for example, `arn:aws:iam::999999999999:instance-profile/ExampleInstanceProfile`).

For more information about the permissions necessary to work with roles, see "How Do I Get Started?" in [Using an IAM role to grant permissions to applications running on Amazon EC2 instances \(p. 263\)](#). For information about adding permissions to a user, see [Managing IAM policies \(p. 438\)](#).

Amazon EC2: When I attempt to launch an instance with a role, I get an AccessDenied error

Check the following:

- Launch an instance without an instance profile. This will help ensure that the problem is limited to IAM roles for Amazon EC2 instances.
- If you are making requests as an IAM user, verify that you have the following permissions:
 - `ec2:RunInstances` with a wildcard resource ("*")
 - `iam:PassRole` with the resource matching the role ARN (for example, `arn:aws:iam::999999999999:role/ExampleRoleName`)
- Call the IAM `GetInstanceProfile` action to ensure that you are using a valid instance profile name or a valid instance profile ARN. For more information, see [Using IAM roles with Amazon EC2 instances](#).
- Call the IAM `GetInstanceProfile` action to ensure that the instance profile has a role. Empty instance profiles will fail with an `AccessDenied` error. For more information about creating a role, see [Creating IAM roles \(p. 221\)](#).

For more information about the permissions necessary to work with roles, see "How Do I Get Started?" in [Using an IAM role to grant permissions to applications running on Amazon EC2 instances \(p. 263\)](#). For information about adding permissions to a user, see [Managing IAM policies \(p. 438\)](#).

I can't access the temporary security credentials on my EC2 instance

To access temporary security credentials on your EC2 instance, you must first use the IAM console to create a role. Then you launch an EC2 instance that uses that role and examine the running instance. For more information, see [How Do I Get Started?](#) in [Using an IAM role to grant permissions to applications running on Amazon EC2 instances \(p. 263\)](#).

If you still can't access your temporary security credentials on your EC2 instance, check the following:

- Can you 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 the EC2 `DescribeInstances` API operation or using the `aws ec2 describe-instances` CLI command.

```
[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 [What do the errors from the info document in the IAM subtree mean? \(p. 593\)](#) for more information.

```
[ec2-user@domU-12-31-39-0A-8D-DE ~]$ GET http://169.254.169.254/latest/meta-data/iam/info; echo
```

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 must attach a valid instance profile to your Amazon EC2 instance.

If an instance profile with that name exists, check that the instance profile wasn't deleted and another was created with the same name:

1. Call the IAM `GetInstanceProfile` operation to get the `InstanceProfileId`.
2. Call the Amazon EC2 `DescribeInstances` operation to get the `IamInstanceProfileId` for the instance.
3. Verify that the `InstanceProfileId` from the IAM operation matches the `IamInstanceProfileId` from the Amazon EC2 operation.

If the IDs are different, then the instance profile attached to your instances is no longer valid. You must attach a valid instance profile to the instance.

The `iam/info` document indicates a success but indicates `"Message": "Instance Profile does not contain a role..."`

The role has been removed from the instance profile by the IAM `RemoveRoleFromInstanceProfile` action. You can use the IAM `AddRoleToInstanceProfile` action to attach a role to the instance profile. Your application will need to wait until the next scheduled refresh to access the credentials for the role.

The `iam/security-credentials/[role-name]` document indicates `"Code": "AssumeRoleUnauthorizedAccess"`

Amazon EC2 does not have permission to assume the role. Permission to assume the role is controlled by the trust policy attached to the role, like the example that follows. Use the IAM `UpdateAssumeRolePolicy` API to update the trust policy.

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

Troubleshooting IAM and Amazon S3

Use the information here to help you troubleshoot and fix issues that you might encounter when working with Amazon S3 and IAM.

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*.

I'm signed in as an AWS account 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 AWS account 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. To do that, modify the bucket policy to allow root user access from the Amazon S3 console or the AWS CLI. Use the following principal, replacing `123456789012` with the ID of the AWS account.

```
"Principal": { "AWS": "arn:aws:iam::123456789012:root" }
```

Troubleshooting SAML 2.0 federation with AWS

Use the information here to help you diagnose and fix issues that you might encounter when working with SAML 2.0 and federation with IAM.

Topics

- [Error: Your request included an invalid SAML response. to logout, click here. \(p. 595\)](#)
- [Error: RoleSessionName is required in AuthnResponse \(service: AWSecurityTokenService; status code: 400; error code: InvalidIdentityToken\) \(p. 595\)](#)
- [Error: Not authorized to perform sts:AssumeRoleWithSAML \(service: AWSecurityTokenService; status code: 403; error code: AccessDenied\) \(p. 595\)](#)
- [Error: RoleSessionName in AuthnResponse must match \[a-zA-Z_0-9+=,.@-\]{2,64} \(service: AWSecurityTokenService; status code: 400; error code: InvalidIdentityToken\) \(p. 596\)](#)
- [Error: Response signature invalid \(service: AWSecurityTokenService; status code: 400; error code: InvalidIdentityToken\) \(p. 596\)](#)
- [Error: Failed to assume role: Issuer not present in specified provider \(service: AWSOpenIdDiscoveryService; status code: 400; error code: AuthSamlInvalidSamlResponseException\) \(p. 596\)](#)
- [Error: Could not parse metadata. \(p. 596\)](#)
- [Error: Specified provider doesn't exist. \(p. 597\)](#)
- [Error: Requested DurationSeconds exceeds MaxSessionDuration set for this role. \(p. 597\)](#)
- [How to view a SAML response in your browser for troubleshooting \(p. 597\)](#)

Error: Your request included an invalid SAML response. to logout, click here.

This error can occur when the SAML response from the identity provider does not include an attribute with the Name set to `https://aws.amazon.com/SAML/Attributes/Role`. The attribute must contain one or more `AttributeValue` elements, each containing a comma-separated pair of strings:

- The ARN of a role that the user can be mapped to
- The ARN of the SAML provider

For more information, see [Configuring SAML assertions for the authentication response \(p. 199\)](#). To view the SAML response in your browser, follow the steps listed in [How to view a SAML response in your browser for troubleshooting \(p. 597\)](#).

Error: RoleSessionName is required in AuthnResponse (service: AWSSecurityTokenService; status code: 400; error code: InvalidIdentityToken)

This error can occur when the SAML response from the identity provider does not include an attribute with the Name set to `https://aws.amazon.com/SAML/Attributes/RoleSessionName`. The attribute value is an identifier for the user and is typically a user ID or an email address.

For more information, see [Configuring SAML assertions for the authentication response \(p. 199\)](#). To view the SAML response in your browser, follow the steps listed in [How to view a SAML response in your browser for troubleshooting \(p. 597\)](#).

Error: Not authorized to perform sts:AssumeRoleWithSAML (service: AWSSecurityTokenService; status code: 403; error code: AccessDenied)

This error can occur if the IAM role specified in the SAML response is misspelled or does not exist. Make sure to use the exact name of your role, because role names are case sensitive. Correct the name of the role in the SAML service provider configuration.

You are allowed access only if your role trust policy includes the `sts:AssumeRoleWithSAML` action. If your SAML assertion is configured to use the [PrincipalTag attribute \(p. 201\)](#), your trust policy must also include the `sts:TagSession` action. For more information about session tags, see [Passing session tags in AWS STS \(p. 293\)](#).

This error can also occur if the federated users do not have permissions to assume the role. The role must have a trust policy that specifies the ARN of the IAM SAML identity provider as the `Principal`. The role also contains conditions that control which users can assume the role. Ensure that your users meet the requirements of the conditions.

This error can also occur if the SAML response does not include a `Subject` containing a `NameID`.

For more information, see [Establish Permissions in AWS for Federated Users](#) and [Configuring SAML assertions for the authentication response \(p. 199\)](#). To view the SAML response in your browser, follow the steps listed in [How to view a SAML response in your browser for troubleshooting \(p. 597\)](#).

Error: RoleSessionName in AuthnResponse must match [a-zA-Z_0-9+=,.@-]{2,64} (service: AWSSecurityTokenService; status code: 400; error code: InvalidIdentityToken)

This error can occur if the `RoleSessionName` attribute value is too long or contains invalid characters. The maximum valid length is 64 characters.

For more information, see [Configuring SAML assertions for the authentication response \(p. 199\)](#). To view the SAML response in your browser, follow the steps listed in [How to view a SAML response in your browser for troubleshooting \(p. 597\)](#).

Error: Response signature invalid (service: AWSSecurityTokenService; status code: 400; error code: InvalidIdentityToken)

This error can occur when federation metadata of the identity provider does not match the metadata of the IAM identity provider. For example, the metadata file for the identity service provider might have changed to update an expired certificate. Download the updated SAML metadata file from your identity service provider. Then update it in the AWS identity provider entity that you define in IAM with the `aws iam update-saml-provider` cross-platform CLI command or the `Update-IAMSSMIPProvider` PowerShell cmdlet.

Error: Failed to assume role: Issuer not present in specified provider (service: AWSOpenIdDiscoveryService; status code: 400; error code: AuthSamlInvalidSamlResponseException)

This error can occur if the issuer in the SAML response does not match the issuer declared in the federation metadata file. The metadata file was uploaded to AWS when you created the identity provider in IAM.

Error: Could not parse metadata.

This error can occur if your metadata file is not formatted properly.

When you [create or manage a SAML identity provider \(p. 194\)](#) in the AWS Management Console, you must retrieve the SAML metadata document from your identity provider. This metadata file includes the issuer's name, expiration information, and keys that can be used to validate the SAML authentication response (assertions) that are received from the IdP. The metadata file must be encoded in UTF-8 format without a byte order mark (BOM). Also, the x.509 certificate that is included as part of the SAML metadata document must use a key size of at least 1024 bits. If the key size is smaller, the IdP creation fails with an "Unable to parse metadata" error. To remove the BOM, you can encode the file as UTF-8 using a text editing tool, such as Notepad++.

Error: Specified provider doesn't exist.

This error can occur if the name of the provider that you specify in the SAML assertion does not match the name of the provider configured in IAM. For more information about viewing the provider name, see [Creating IAM SAML identity providers \(p. 193\)](#).

Error: Requested DurationSeconds exceeds MaxSessionDuration set for this role.

This error can occur if you assume a role from the AWS CLI or API.

When you use the [assume-role-with-saml](#) CLI or [AssumeRoleWithSAML](#) API operations to assume a role, you can specify a value for the `DurationSeconds` parameter. You can specify a value from 900 seconds (15 minutes) up to the maximum session duration setting for the role. If you specify a value higher than this setting, the operation fails. For example, if you specify a session duration of 12 hours, but your administrator set the maximum session duration to 6 hours, your operation fails. To learn how to view the maximum value for your role, see [View the maximum session duration setting for a role \(p. 248\)](#).

How to view a SAML response in your browser for troubleshooting

The following procedures describe how to view the SAML response from your service provider from in your browser when troubleshooting a SAML 2.0-related issue.

For all browsers, go to the page where you can reproduce the issue. Then follow the steps for the appropriate browser:

Topics

- [Google chrome \(p. 597\)](#)
- [Mozilla firefox \(p. 598\)](#)
- [Apple safari \(p. 598\)](#)
- [Microsoft Internet Explorer \(p. 598\)](#)
- [What to do with the Base64-encoded SAML response \(p. 598\)](#)

Google chrome

To view a SAML response in chrome

These steps were tested using version 54.0.2840.87m. If you use another version, you might need to adapt the steps accordingly.

1. Press **F12** to start the developer console.
2. Select the **Network** tab, and then select **Preserve log**.
3. Reproduce the issue.
4. Look for a **SAML Post** in the developer console pane. Select that row, and then view the **Headers** tab at the bottom. Look for the **SAMLResponse** attribute that contains the encoded request.

Mozilla firefox

To view a SAML response in firefox

This procedure was tested on version 37.0.2 of Mozilla Firefox. If you use another version, you might need to adapt the steps accordingly.

1. Press **F12** to start the developer console.
2. In the upper right of the developer tools window, click options (the small gear icon). Under **Common Preferences** select **Enable persistent logs**.
3. Select the **Network** tab.
4. Reproduce the issue.
5. Look for a **POST SAML** in the table. Select that row. In the **Form Data** window on the right, select the **Params** tab and find the **SAMLResponse** element.

Apple safari

To view a SAML response in safari

These steps were tested using version 8.0.6 (10600.6.3). If you use another version, you might need to adapt the steps accordingly.

1. Enable Web Inspector in Safari. Open the **Preferences** window, select the **Advanced** tab, and then select **Show Develop menu in the menu bar**.
2. Now you can open Web Inspector. Click **Develop**, then select **Show Web Inspector**.
3. Select the **Resources** tab.
4. Reproduce the issue.
5. Look for a **saml-signin.aws.amazon.com** request.
6. Scroll down to find Request Data with the name **SAMLResponse**. The associated value is the Base64-encoded response.

Microsoft Internet Explorer

To view a SAML response in Internet Explorer

The best way to analyze network traffic in Internet Explorer is through the use of a third-party tool.

- Follow the steps at <http://social.technet.microsoft.com/wiki/contents/articles/3286.ad-fs-2-0-how-to-use-fiddler-web-debugger-to-analyze-a-ws-federation-passive-sign-in.aspx> to download and install Fiddler and capture the data.

What to do with the Base64-encoded SAML response

Once you find the Base64-encoded SAML response element in your browser, copy it and use your favorite Base-64 decoding tool to extract the XML tagged response.

Security tip

Because the SAML response data that you are viewing might contain sensitive security data, we recommend that you do not use an *online* base64 decoder. Instead use a tool installed on your local computer that does not send your SAML data over the network.

Built-in option for Windows systems (PowerShell):

```
PS C:\> [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("base64encodedtext"))
```

Built-in option for MacOS and Linux systems:

```
$ echo "base64encodedtext" | base64 --decode
```

Reference information for AWS Identity and Access Management

Use the topics in this section to find detailed reference material for various aspects of IAM and AWS STS.

Topics

- [IAM identifiers \(p. 600\)](#)
- [IAM and STS quotas \(p. 606\)](#)
- [AWS services that work with IAM \(p. 611\)](#)
- [IAM JSON policy reference \(p. 627\)](#)

IAM identifiers

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

Topics

- [Friendly names and paths \(p. 600\)](#)
- [IAM ARNs \(p. 601\)](#)
- [Unique identifiers \(p. 604\)](#)

Friendly names and paths

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

If you are using the IAM API or AWS Command Line Interface (AWS CLI) to create IAM resources, you can also give some resources an optional path. You can use a single path, or nest multiple paths as if they were a folder structure. For example, you could use the nested path /division_abc/subdivision_xyz/product_1234/engineering/ to match your company's organizational structure. You could then create a policy to allow all users in that path to access the policy simulator API. To view this policy, see [IAM: Access the policy simulator API based on user path \(p. 425\)](#). For additional examples of how you might use paths, see [IAM ARNs \(p. 601\)](#).

When you use AWS CloudFormation to create resources, you can specify a path for users, groups, and roles, and customer managed policies.

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 those resources). The number and size of IAM resources in an AWS account are limited. For more information, see [IAM and STS quotas \(p. 606\)](#).

IAM ARNs

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

```
arn:partition:service:region:account:resource
```

Where:

- **partition** identifies the partition that the resource is in. For standard AWS Regions, the partition is `aws`. If you have resources in other partitions, the partition is `aws-partitionname`. For example, the partition for resources in the China (Beijing) Region is `aws-cn`. You cannot [delegate access \(p. 287\)](#) between accounts in different partitions.
- **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 kept blank.
- **account** is the AWS account ID with no hyphens (for example, `123456789012`).
- **resource** is the portion that identifies the specific resource by name.

You can specify IAM and AWS STS ARNs using the following syntax. The Region portion of the ARN is blank because IAM resources are global.

Syntax:

```
arn:aws:iam::account-id:root
arn:aws:iam::account-id:user/user-name-with-path
arn:aws:iam::account-id:group/group-name-with-path
arn:aws:iam::account-id:role/role-name-with-path
arn:aws:iam::account-id:policy/policy-name-with-path
arn:aws:iam::account-id:instance-profile/instance-profile-name-with-path
arn:aws:sts::account-id:federated-user/user-name
arn:aws:sts::account-id:assumed-role/role-name/role-session-name
arn:aws:iam::account-id:mfa/virtual-device-name-with-path
arn:aws:iam::account-id:u2f/u2f-token-id
arn:aws:iam::account-id:server-certificate/certificate-name-with-path
arn:aws:iam::account-id:saml-provider/provider-name
arn:aws:iam::account-id:oidc-provider/provider-name
```

Many of the following examples include paths in the resource part of the ARN. Paths cannot be created or manipulated in the AWS Management Console. To use paths you must work with the resource by using the AWS API, the AWS CLI, or the Tools for Windows PowerShell.

Examples:

```
arn:aws:iam::123456789012:root
arn:aws:iam::123456789012:user/JohnDoe
arn:aws:iam::123456789012:user/division_abc/subdivision_xyz/JaneDoe
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:role/aws-service-role/access-analyzer.amazonaws.com/
AWSServiceRoleForAccessAnalyzer
arn:aws:iam::123456789012:role/service-role/QuickSightAction
arn:aws:iam::123456789012:policy/UsersManageOwnCredentials
arn:aws:iam::123456789012:policy/division_abc/subdivision_xyz/UsersManageOwnCredentials
arn:aws:iam::123456789012:instance-profile/Webserver
arn:aws:sts::123456789012:federated-user/JohnDoe
```

```
arn:aws:sts::123456789012:assumed-role/Accounting-Role/JaneDoe
arn:aws:iam::123456789012:mfa/JaneDoeMFA
arn:aws:iam::123456789012:u2f/user/JohnDoe/default (U2F security key)
arn:aws:iam::123456789012:server-certificate/ProdServerCert
arn:aws:iam::123456789012:server-certificate/division_abc/subdivision_xyz/ProdServerCert
arn:aws:iam::123456789012:saml-provider/ADFSProvider
arn:aws:iam::123456789012:oidc-provider/GoogleProvider
```

The following examples provide more detail to help you understand the ARN format for different types of IAM and AWS STS resources.

- An IAM user in the account:

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

- Another user with a path reflecting an organization chart:

```
arn:aws:iam::123456789012:user/division_abc/subdivision_xyz/JaneDoe
```

- An IAM group:

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

- An IAM group with a path:

```
arn:aws:iam::123456789012:group/division_abc/subdivision_xyz/product_A/Developers
```

- An IAM role:

```
arn:aws:iam::123456789012:role/S3Access
```

- A [service-linked role \(p. 168\)](#):

```
arn:aws:iam::123456789012:role/aws-service-role/access-analyzer.amazonaws.com/
AWSServiceRoleForAccessAnalyzer
```

- A [service role \(p. 168\)](#):

```
arn:aws:iam::123456789012:role/service-role/QuickSightAction
```

- A managed policy:

```
arn:aws:iam::123456789012:policy/ManageCredentialsPermissions
```

- An instance profile that can be associated with an EC2 instance:

```
arn:aws:iam::123456789012:instance-profile/Webserver
```

- A federated user identified in IAM as "Paulo":

```
arn:aws:sts::123456789012:federated-user/Paulo
```

- The active session of someone assuming the role of "Accounting-Role", with a role session name of "Mary":

```
arn:aws:sts::123456789012:assumed-role/Accounting-Role/Mary
```

- The multi-factor authentication device assigned to the user named Jorge:

```
arn:aws:iam::123456789012:mfa/Jorge
```

- A server certificate:

```
arn:aws:iam::123456789012:server-certificate/ProdServerCert
```

- A server certificate with a path that reflects an organization chart:

```
arn:aws:iam::123456789012:server-certificate/division_abc/subdivision_xyz/ProdServerCert
```

- Identity providers (SAML and OIDC):

```
arn:aws:iam::123456789012:saml-provider/ADFSProvider  
arn:aws:iam::123456789012:oidc-provider/GoogleProvider
```

Another important ARN is the root user ARN. Although this is not an IAM resource, you should be familiar with the format of this ARN. It is often used in the [Principal element \(p. 631\)](#) of a policy.

- The AWS account - the account itself:

```
arn:aws:iam::123456789012:root
```

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

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "ManageRichardAccessKeys",  
            "Effect": "Allow",  
            "Action": [  
                "iam:*AccessKey*",  
                "iam:GetUser"  
            ],  
            "Resource": "arn:aws:iam::*:user/division_abc/subdivision_xyz/Richard"  
        },  
        {  
            "Sid": "ListForConsole",  
            "Effect": "Allow",  
            "Action": "iam>ListUsers",  
            "Resource": "*"  
        }  
    ]  
}
```

Note

When you use ARNs to identify resources in an IAM policy, you can include *policy variables*. Policy variables can include placeholders for runtime information (such as the user's name) as part of the ARN. For more information, see [IAM policy elements: Variables and tags \(p. 658\)](#)

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

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

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

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

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

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

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

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

Example Example use of paths and ARNs for a project-based group

Paths cannot be created or manipulated in the AWS Management Console. To use paths you must work with the resource by using the AWS API, the AWS CLI, or the Tools for Windows PowerShell.

In this example, Jules in the `Marketing_Admin` group creates a project-based group within the `/marketing/` path. Jules 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` that is designated to this particular launch.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:*",
      "Resource": "arn:aws:s3:::example_bucket/marketing/newproductlaunch/widget/*"
    },
    {
      "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 Patricia and Eli from the `/marketing/` path. It also includes Chris and Chloe from the `/sales/` path, and Alice and Jim from the `/legal/` path.

Unique identifiers

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

AIDAJQABLZS4A3QDU576Q

For the most part, you use friendly names and [ARNs \(p. 601\)](#) when you work with IAM resources. That way you don't need to know the unique ID for a specific resource. However, the unique ID can sometimes be useful when it isn't practical to use friendly names.

One example pertains to reusing friendly names in your AWS account. Within your account, a friendly name for a user, group, or policy must be unique. For example, you might create an IAM user named David. Your company uses Amazon S3 and has a bucket with folders for each employee. The bucket has a resource-based policy (a bucket policy) that lets users access only their own folders in the bucket. Suppose that the employee named David leaves your company and you delete the corresponding IAM user. But later another employee named David starts and you create a new IAM user named David. If the bucket policy specifies the David IAM user, the policy allows the new David to access information 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. You can create resource policies for Amazon S3 buckets that grant access by unique ID and not just by user name. Doing so reduces the chance that you could inadvertently grant access to information that an employee should not have.

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

Understanding unique ID prefixes

IAM uses the following prefixes to indicate what type of resource each unique ID applies to.

Prefix	Resource type
ABIA	AWS STS service bearer token (p. 332)
ACCA	Context-specific credential
AGPA	Group
AIDA	IAM user
AIPA	Amazon EC2 instance profile
AKIA	Access key
ANPA	Managed policy
ANVA	Version in a managed policy
APKA	Public key
AROA	Role
ASCA	Certificate
ASIA	Temporary (AWS STS) access key IDs use this prefix, but are unique only in combination with the secret access key and the session token.

Getting the unique identifier

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

AWS CLI:

- [get-caller-identity](#)
- [get-group](#)
- [get-role](#)
- [get-user](#)
- [get-policy](#)
- [get-instance-profile](#)
- [get-server-certificate](#)

IAM API:

- [GetCallerIdentity](#)
- [GetGroup](#)
- [GetRole](#)
- [GetUser](#)
- [GetPolicy](#)
- [GetInstanceProfile](#)
- [GetServerCertificate](#)

IAM and STS quotas

AWS Identity and Access Management (IAM) and AWS Security Token Service (STS) have quotas that limit the size of objects. This affects how you name an object, the number of objects you can create, and the number of characters you can use when you pass an object.

Note

To get account-level information about IAM usage and quotas, use the [GetAccountSummary](#) API operation or the [get-account-summary](#) AWS CLI command.

IAM name requirements

IAM names have the following requirements and restrictions:

- Policy documents can contain only the following Unicode characters: horizontal tab (U+0009), linefeed (U+000A), carriage return (U+000D), and characters in the range U+0020 to U+00FF.
- Names of users, groups, roles, policies, instance profiles, and server certificates must be alphanumeric, including the following common characters: plus (+), equal (=), comma (,), period (.), at (@), underscore (_), and hyphen (-).
- Names of users, groups, roles, and instance profiles must be unique within the account. They are not distinguished by case, for example, you cannot create groups named both **ADMINS** and **admins**.
- The external ID value that a third party uses to assume a role must have a minimum of 2 characters and a maximum of 1,224 characters. The value must be alphanumeric without white space. It can also include the following symbols: plus (+), equal (=), comma (,), period (.), at (@), colon (:), forward slash (/), and hyphen (-). For more information about the external ID, see [How to use an external ID when granting access to your AWS resources to a third party \(p. 225\)](#).
- Path names must begin and end with a forward slash (/).
- Policy names for [inline policies \(p. 359\)](#) must be unique to the user, group, or role they are embedded in. The names can contain any Basic Latin (ASCII) characters minus the following reserved characters:

backward slash (\), forward slash (/), asterisk (*), question mark (?), and white space. These characters are reserved according to [RFC 3986](#).

- User passwords (login profiles) can contain any Basic Latin (ASCII) characters.
- AWS account ID aliases must be unique across AWS products, and must be alphanumeric following DNS naming conventions. An alias must be lowercase, it must not start or end with a hyphen, it cannot contain two consecutive hyphens, and it cannot be a 12-digit number.

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

IAM object quotas

AWS allows you to request an increase to default quotas for IAM entities. Use Service Quotas to manage your IAM quotas. For adjustable IAM quotas, you can request a quota increase. Smaller increases are automatically approved in Service Quotas and are completed within a few minutes. Larger requests above the [maximum autoapproved increase](#) are submitted to AWS Support. Some adjustable quotas can't be increased above the maximum autoapproved increase amount. You can track your request case in the AWS Support console.

To request a quota increase, sign in to the AWS Management Console and open the Service Quotas console at <https://console.aws.amazon.com/servicequotas/>. In the navigation pane, choose **AWS services**. On the navigation bar, choose the **US East (N. Virginia)** Region. Then search for **IAM**. Choose **AWS Identity and Access Management (IAM)**, choose a quota, and follow the directions to request a quota increase. For more information, see [Requesting a Quota Increase](#) in the *Service Quotas User Guide*.

To see an example of how to request an IAM quota increase using the Service Quotas console, watch the following video.

[Request an IAM quota increase using the Service Quotas console](#).

The following quotas are adjustable.

Default quotas for IAM entities

Resource	Default quota	Maximum autoapproval
Role trust policy length	2048 characters	4096 characters
Customer managed policies in an AWS account	1500	5000
Groups in an AWS account	300	500
Roles in an AWS account	1000	5000
Managed policies attached to an IAM role	10	20
Managed policies attached to an IAM user	10	20
Virtual MFA devices (assigned or unassigned) in an AWS account	Equal to the user quota for the account	Not applicable
Instance profiles in an AWS account	1000	5000
Server certificates stored in an AWS account	20	1000

You cannot request an increase for the following quotas.

Quotas for IAM entities

Resource	Quota
Access keys assigned to an IAM user	2
Access keys assigned to the AWS account root user	2
Aliases for an AWS account	1
Domains for an AWS account	2
Groups an IAM user can be a member of	10
IAM users in a group	Equal to the user quota for the account
Identity providers (IdPs) associated with an IAM SAML provider object	10
Keys per SAML provider	10
Login profiles for an IAM user	1
Managed policies attached to an IAM group	10
OpenID Connect identity providers per AWS account	100
Permissions boundaries for an IAM user	1
Permissions boundaries for an IAM role	1
MFA devices in use by an IAM user	1
MFA devices in use by the AWS account root user	1
Roles in an instance profile	1
SAML providers in an AWS account	100
Signing certificates assigned to an IAM user	2
SSH public keys assigned to an IAM user	5
Tags that can be attached to an IAM role	50
Tags that can be attached to an IAM user	50
Users in an AWS account	5000 (If you need to add a large number of users, consider using temporary security credentials (p. 301) .)
Versions of a managed policy that can be stored	5

IAM and STS character quotas

The following are the maximum character counts and size quotas for IAM and AWS STS. You cannot request an increase for the following quotas.

Description	Quota
Path	512 characters
User name	64 characters
Group name	128 characters
Role name	64 characters Important If you intend to use a role with the Switch Role feature in the AWS console, then the combined Path and RoleName cannot exceed 64 characters.
Tag key	128 characters This character quota applies to user tags, role tags, and session tags (p. 293) .
Tag value	256 characters This character quota applies to user tags, role tags, and session tags (p. 293) . Tag values can be empty. That is, tag values can have a length of 0 characters.
Instance profile name	128 characters
Unique IDs created by IAM, for example:	128 characters <ul style="list-style-type: none"> • User IDs that begin with AIDA • Group IDs that begin with AGPA • Role IDs that begin with AROA • Managed policy IDs that begin with ANPA • Server certificate IDs that begin with ASCA <p>Note This is not intended to be an exhaustive list, nor is it a guarantee that IDs of a certain type begin only with the specified letter combination.</p>
Policy name	128 characters
Password for a login profile	1–128 characters
Alias for an AWS account ID	3–63 characters
Role session name	64 characters
Role session duration	12 hours When you assume a role from the AWS CLI or API, you can use the <code>duration-seconds</code> CLI parameter or the <code>DurationSeconds</code> API parameter to request a longer role session. You

Description	Quota
	<p>can specify a value from 900 seconds (15 minutes) up to the maximum session duration setting for the role, which can range 1–12 hours. If you don't specify a value for the <code>DurationSeconds</code> parameter, your security credentials are valid for one hour. IAM users who switch roles in the console are granted the maximum session duration, or the remaining time in the IAM user's session, whichever is less. The maximum session duration setting does not limit sessions assumed by AWS services. To learn how to view the maximum value for your role, see View the maximum session duration setting for a role (p. 248).</p>
Role session policies (p. 353)	<ul style="list-style-type: none"> • You can pass a maximum of 10 managed policy ARNs when you create a session. • You can pass only one JSON policy document when you programmatically create a temporary session for a role or federated user. • The size of the passed JSON policy document and all passed managed policy ARN characters combined cannot exceed 2,048 characters. • Additionally, an AWS conversion compresses the passed session policies and session tags into a packed binary format that has a separate quota. The <code>PackedPolicySize</code> response element indicates by percentage how close the policies and tags for your request are to the upper size quota. • We recommend that you pass session policies using the AWS CLI or AWS API. The AWS Management Console might add additional console session information to the packed policy.
Role session tags (p. 293)	<ul style="list-style-type: none"> • Session tags must meet the tag key quota of 128 characters and the tag value quota of 256 characters. • You can pass up to 50 session tags. • An AWS conversion compresses the passed session policies and session tags into a packed binary format that has a separate quota. You can pass session tags using the AWS CLI or AWS API. The <code>PackedPolicySize</code> response element indicates by percentage how close the policies and tags for your request are to the upper size quota.

Description	Quota
For inline policies (p. 359)	<p>You can add as many inline policies as you want to an IAM user, role, or group. But the total aggregate policy size (the sum size of all inline policies) per entity cannot exceed the following quotas:</p> <ul style="list-style-type: none"> • User policy size cannot exceed 2,048 characters. • Role policy size cannot exceed 10,240 characters. • Group policy size cannot exceed 5,120 characters. <p>Note IAM does not count white space when calculating the size of a policy against these quotas.</p>
For managed policies (p. 359)	<ul style="list-style-type: none"> • You can add up to 10 managed policies to an IAM user, role, or group. • The size of each managed policy cannot exceed 6,144 characters. <p>Note IAM does not count white space when calculating the size of a policy against this quota.</p>

AWS services that work with IAM

The AWS services listed below are grouped by their [AWS product categories](#) and include information about what IAM features they support:

- **Service** – You can choose the name of a service to view the AWS documentation about IAM authorization and access for that service.
- **Actions** – You can specify individual actions in a policy. If the service does not support this feature, then **All actions** is selected in the [visual editor \(p. 440\)](#). In a JSON policy document, you must use * in the Action element. For a list of actions in each service, see [Actions, Resources, and Condition Keys for AWS Services](#).
- **Resource-level permissions** – You can use [ARNs \(p. 601\)](#) to specify individual resources in the policy. If the service does not support this feature, then **All resources** is chosen in the [policy visual editor \(p. 440\)](#). In a JSON policy document, you must use * in the Resource element. Some actions, such as List* actions, do not support specifying an ARN because they are designed to return multiple resources. If a service supports this feature for some resources but not others, it is indicated by yellow cells in the table. See the documentation for that service for more information.
- **Resource-based policies** – You can attach resource-based policies to a resource within the service. Resource-based policies include a Principal element to specify which IAM identities can access that resource. For more information, see [Identity-based policies and resource-based policies \(p. 374\)](#).
- **Authorization based on tags** – You can use [resource tags](#) in the condition of a policy to control access to a resource in the service. You do this using the [aws:ResourceTag \(p. 703\)](#) global condition key

or service-specific tags, such as [ec2:ResourceTag](#). For more information about defining permissions based on attributes such as tags, see [What is ABAC for AWS? \(p. 13\)](#).

- **Temporary credentials** – You can use short-term credentials that you obtain when you sign in using SSO, switch roles in the console, or that you generate using AWS STS in the AWS CLI or AWS API. You can access services with a **No** value only while using your long-term IAM user credentials. This includes a user name and password or your user access keys. For more information, see [Temporary security credentials in IAM \(p. 301\)](#).
- **Service-linked roles** – A [service-linked role \(p. 168\)](#) is a special type of service role that gives the service permission to access resources in other services on your behalf. Choose the **Yes** link to see the documentation for services that support these roles. This column does not indicate if the service uses standard service roles. For more information, see [Using service-linked roles \(p. 213\)](#).
- **More information** – If a service doesn't fully support a feature, you can review the footnotes for an entry to view the limitations and links to related information.

Compute services

Service	Actions	Resource-level permission	Resource-based policies	Authorization based on tags	Temporary credentials	Service-linked roles
AWS Batch	✓ Yes	⚠ Partial	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Elastic Compute Cloud (Amazon EC2)	✓ Yes	⚠ Partial	✗ No	⚠ Partial	✓ Yes	⚠ Partial ¹
Amazon EC2 Auto Scaling	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
EC2 Image Builder	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Elastic Beanstalk	✓ Yes	⚠ Partial	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Elastic Inference	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
Elastic Load Balancing	✓ Yes	⚠ Partial	✗ No	⚠ Partial	✓ Yes	✓ Yes
AWS Lambda	✓ Yes	✓ Yes	✓ Yes	✗ No	✓ Yes	⚠ Partial ²
Amazon Lightsail	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Outposts	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✓ Yes
AWS Serverless Application Repository	✓ Yes	✓ Yes	✓ Yes	✗ No	✓ Yes	✗ No

¹ Amazon EC2 service-linked roles cannot be created using the AWS Management Console, and can be used only for the following features: [Scheduled Instances](#), [Spot Instance Requests](#), [Spot Fleet Requests](#).

² AWS Lambda doesn't have service-linked roles, but Lambda@Edge does. For more information, see [Service-Linked Roles for Lambda@Edge](#) in the [Amazon CloudFront Developer Guide](#).

Containers services

Service	Actions	Resource-level permission	Resource-based policies	Authorization based on tags	Temporary credentials	Service-linked roles
Amazon Elastic Container Registry (Amazon ECR)	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✗ No
Amazon Elastic Container Registry Public (Amazon ECR Public)	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
Amazon Elastic Container Service (Amazon ECS)	✓ Yes	⚠ Partial ¹	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Elastic Kubernetes Service (Amazon EKS)	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes

¹ Only some Amazon ECS actions support resource-level permissions.

Storage services

Service	Actions	Resource-level permission	Resource-based policies	Authorization based on tags	Temporary credentials	Service-linked roles
AWS Backup	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes
AWS Backup Storage	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
Amazon Elastic Block Store (Amazon EBS)	✓ Yes	⚠ Partial	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Elastic File System (Amazon EFS)	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes
Amazon FSx	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon S3 Glacier	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✗ No
AWS Import/Export	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
Amazon Simple Storage Service (Amazon S3)	✓ Yes	✓ Yes	✓ Yes	⚠ Partial ¹	✓ Yes	⚠ Partial ²
Amazon Simple Storage Service (Amazon S3) on AWS Outposts	✓ Yes	✓ Yes	✓ Yes	⚠ Partial ¹	✓ Yes	✗ No
AWS Snowball	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Snowball Edge	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Storage Gateway	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No

¹ Amazon S3 supports tag-based authorization for only object resources.

² Amazon S3 supports service-linked roles for Amazon S3 Storage Lens.

Database services

Service	Actions	Resource-level permission	Resource-based policies	Authorization based on tags	Temporary credentials	Service-linked roles
Amazon DynamoDB	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✓ Yes
Amazon ElastiCache	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✓ Yes
Amazon Keyspaces (for Apache Cassandra)	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Quantum Ledger Database (Amazon QLDB)	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Redshift	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Redshift Data API	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
Amazon Relational Database Service (Amazon RDS)	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon RDS Data API	✓ Yes	✗ No	✗ No	✓ Yes	✓ Yes	✗ No
Amazon SimpleDB	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
Amazon Timestream	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No

Developer tools services

Service	Actions	Resource-level permission	Resource-based policies	Authorization based on tags	Temporary credentials	Service-linked roles
AWS Cloud9	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes
AWS CloudShell	✓ Yes	✓ Yes	✗ No	✗ No	✗ No	✗ No
AWS CodeArtifact	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✗ No
CodeBuild	✓ Yes	✓ Yes	✓ Yes ¹	⚠ Partial ²	✓ Yes	✗ No
CodeCommit	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS CodeDeploy	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No

Service	Actions	Resource-level permission	Resource-based policies	Authorization based on tags	Temporary credentials	Service-linked roles
CodePipeline	✓ Yes	⚠ Partial	✗ No	✓ Yes	✓ Yes	✗ No
AWS CodeStar	✓ Yes	⚠ Partial ¹	✗ No	✓ Yes	✓ Yes	✗ No
AWS CodeStar Connections	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS CodeStar Notifications	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS X-Ray	✓ Yes	✓ Yes	✗ No	⚠ Partial ³	✓ Yes	✗ No

¹ CodeBuild supports cross-account resource sharing using AWS RAM.

² CodeBuild supports authorization based on tags for project-based actions.

³ X-Ray supports tag-based access control for groups and sampling rules.

Security, identity, and compliance services

Service	Actions	Resource-level permission	Resource-based policies	Authorization based on tags	Temporary credentials	Service-linked roles
AWS Artifact	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
AWS Audit Manager	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Cognito	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Detective	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
AWS Directory Service	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Firewall Manager	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	⚠ Partial
Amazon GuardDuty	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	⚠ Partial
AWS Identity and Access Management (IAM)	✓ Yes	✓ Yes	⚠ Partial ² (p. 384)	⚠ Partial ³	⚠ Partial	✗ No
IAM Access Analyzer	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	⚠ Partial
Amazon Inspector	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✓ Yes
Amazon Macie	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Macie Classic	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✓ Yes

Service	Actions	Resource-level permission	Resource-based policies	Authorization based on tags	Temporary credentials	Service-linked roles
AWS Network Firewall	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Resource Access Manager (AWS RAM)	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Secrets Manager	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✗ No
AWS Security Hub	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Single Sign-On (AWS SSO)	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS SSO Directory	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS SSO Identity Store	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Security Token Service (AWS STS)	✓ Yes	⚠ Partial ⁴	✗ No	✓ Yes	⚠ Partial ⁵	✗ No
AWS Shield Advanced	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
AWS WAF	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS WAF Classic	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes

¹ IAM supports only one type of resource-based policy called a role *trust policy*, which is attached to an IAM role. For more information, see [Granting a user permissions to switch roles \(p. 248\)](#).

² IAM supports tag-based access control for only user and role resources.

³ Only some of the API actions for IAM can be called with temporary credentials. For more information, see [Comparing your API options](#).

⁴ AWS STS does not have "resources," but does allow restricting access in a similar way to users. For more information, see [Denying Access to Temporary Security Credentials by Name](#).

⁵ Only some of the API operations for AWS STS support calling with temporary credentials. For more information, see [Comparing your API options](#).

Cryptography and PKI services

Service	Actions	Resource-level permission	Resource-based policies	Authorization based on tags	Temporary credentials	Service-linked roles
AWS Certificate Manager Private Certificate Authority (ACM)	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Certificate Manager (ACM)	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS CloudHSM	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes

Service	Actions	Resource-level permission	Resource-based policies	Authorization based on tags	Temporary credentials	Service-linked roles
AWS Key Management Service (AWS KMS)	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes
AWS Signer	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No

Machine learning services

Service	Actions	Resource-level permission	Resource-based policies	Authorization based on tags	Temporary credentials	Service-linked roles
Amazon CodeGuru	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
Amazon CodeGuru Profiler	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon CodeGuru Reviewer	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Comprehend	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS DeepComposer	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS DeepRacer	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✓ Yes
AWS Panorama	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon DevOps Guru	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
Amazon Forecast	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Fraud Detector	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Ground Truth Labeling	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
Amazon Kendra	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Lex	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Lookout for Vision	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
Amazon Monitron	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Machine Learning	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Personalize	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
Amazon Polly	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
Amazon Rekognition	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No

Service	Actions	Resource-level permission	Resource-based policies	Authorization based on tags	Temporary credentials	Service-linked roles
Amazon SageMaker	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Textract	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
Amazon Transcribe	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
Amazon Translate	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No

Management and governance services

Service	Actions	Resource-level permission	Resource-based policies	Authorization based on tags	Temporary credentials	Service-linked roles
Application Auto Scaling	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✓ Yes
AWS AppConfig	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Auto Scaling	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✓ Yes
AWS Chatbot	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✓ Yes
AWS CloudFormation	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS CloudTrail	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✓ Yes
Amazon CloudWatch	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	⚠ Partial ¹
Amazon CloudWatch Application Insights	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
Amazon CloudWatch Events	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon CloudWatch Logs	✓ Yes	✓ Yes	✓ Yes	✗ No	✓ Yes	✗ No
Amazon CloudWatch Synthetics	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
AWS Compute Optimizer	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✓ Yes
AWS Config	✓ Yes	⚠ Partial ²	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Data Lifecycle Manager	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Health	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
AWS License Manager	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes

Service	Actions	Resource-level permission	Resource-based policies	Authorization based on tags	Temporary credentials	Service-linked roles
Amazon Managed Service for Prometheus (AMP)	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
AWS OpsWorks	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS OpsWorks for Chef Automate	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS OpsWorks Configuration Management	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Organizations	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Proton	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
AWS Resource Groups	✓ Yes	✓ Yes	✗ No	✓ Yes	⚠ Partial ³	✗ No
Resource Groups Tagging API	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Service Catalog	✓ Yes	✓ Yes	✗ No	⚠ Partial ⁴	✓ Yes	✗ No
AWS Systems Manager	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Tag Editor	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Trusted Advisor	⚠ Partial ⁵	✓ Yes	✗ No	✗ No	⚠ Partial	✓ Yes
AWS Well-Architected Tool	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
Service Quotas	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No

¹ Amazon CloudWatch service-linked roles cannot be created using the AWS Management Console, and support only the [Alarm Actions](#) feature.

² AWS Config supports resource-level permissions for multi-account multi-Region data aggregation and AWS Config Rules. For a list of supported resources, see the **Multi-Account Multi-Region Data Aggregation** section and **AWS Config Rules** section of [AWS Config API Guide](#).

³ Users can assume a role with a policy that allows AWS Resource Groups operations.

⁴ AWS Service Catalog supports tag-based access control for only actions that match API operations with one resource in the input.

⁵ API access to Trusted Advisor is through the AWS Support API and is controlled by AWS Support IAM policies.

Migration and transfer services

Service	Actions	Resource-level permission	Resource-based policies	Authorization based on tags	Temporary credentials	Service-linked roles
AWS Application Discovery Service	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✓ Yes
AWS Connector Service	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Transfer for SFTP	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Database Migration Service	✓ Yes	✓ Yes	✓ Yes ¹	✓ Yes	✓ Yes	✗ No
AWS DataSync	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Migration Hub	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
AWS Server Migration Service	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✓ Yes

¹ You can create and modify policies that are attached to AWS KMS encryption keys you create to encrypt data migrated to supported target endpoints. The supported target endpoints include Amazon Redshift and Amazon S3. For more information, see [Creating and Using AWS KMS Keys to Encrypt Amazon Redshift Target Data](#) and [Creating AWS KMS Keys to Encrypt Amazon S3 Target Objects](#) in the [AWS Database Migration Service User Guide](#).

Mobile services

Service	Actions	Resource-level permission	Resource-based policies	Authorization based on tags	Temporary credentials	Service-linked roles
AWS Amplify	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Amplify Admin	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
AWS AppSync	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Device Farm	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Location	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No

Networking and content delivery services

Service	Actions	Resource-level permission	Resource-based policies	Authorization based on tags	Temporary credentials	Service-linked roles
Amazon API Gateway	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes

Service	Actions	Resource-level permission	Resource-based policies	Authorization based on tags	Temporary credentials	Service-linked roles
AWS App Mesh	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon CloudFront	⚠ Partial ¹	✓ Yes	✗ No	✓ Yes	✓ Yes	⚠ Partial ⁴
AWS Cloud Map	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Direct Connect	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Global Accelerator	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Network Manager	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes
Amazon Route 53	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
Amazon Route 53 Resolver	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Virtual Private Cloud (Amazon VPC)	✓ Yes	⚠ Partial ²	⚠ Partial ³	✗ No	✓ Yes	✗ No

¹ CloudFront does not support action-level permissions for creating CloudFront key pairs. You must use an AWS account root user to create a CloudFront key pair. For more information, see [Creating CloudFront Key Pairs for Your Trusted Signers](#) in the *Amazon CloudFront Developer Guide*.

² In an IAM user policy, you cannot restrict permissions to a specific Amazon VPC endpoint. Any Action element that includes the ec2 : *VpcEndpoint* or ec2 : DescribePrefixLists API actions must specify "Resource": "*". For more information, see [Controlling the Use of Endpoints](#) in the *Amazon VPC User Guide*.

³ Amazon VPC supports attaching a single resource policy to a VPC endpoint to restrict what can be accessed through that endpoint. For more information about using resource-based policies to control access to resources from specific Amazon VPC endpoints, see [Using Endpoint Policies](#) in the *Amazon VPC User Guide*.

⁴ Amazon CloudFront doesn't have service-linked roles, but Lambda@Edge does. For more information, see [Service-Linked Roles for Lambda@Edge](#) in the *Amazon CloudFront Developer Guide*.

Media services

Service	Actions	Resource-level permission	Resource-based policies	Authorization based on tags	Temporary credentials	Service-linked roles
Amazon Elastic Transcoder	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
AWS Elemental Activations	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Elemental Appliances and Software	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No

Service	Actions	Resource-level permission	Resource-based policies	Authorization based on tags	Temporary credentials	Service-linked roles
AWS Elemental MediaConnect	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
AWS Elemental MediaConvert	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Elemental MediaLive	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Elemental MediaPackage	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Elemental MediaPackage VOD	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Elemental MediaStore	✓ Yes	✓ Yes	✓ Yes	✗ No	✓ Yes	✗ No
AWS Elemental MediaTailor	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Elemental Support Cases	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Elemental Support Content	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
Amazon Interactive Video Service	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Kinesis Video Streams	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No

Analytics services

Service	Actions	Resource-level permission	Resource-based policies	Authorization based on tags	Temporary credentials	Service-linked roles
Amazon Athena	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon CloudSearch	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
AWS Data Exchange	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Data Pipeline	✓ Yes	✗ No	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Elasticsearch Service	✓ Yes	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes
Amazon EMR	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon EMR on EKS (EMR Containers)	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Glue	✓ Yes	✓ Yes	✓ Yes	⚠ Partial	✓ Yes	✗ No
AWS Glue DataBrew	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No

Service	Actions	Resource-level permission	Resource-based policies	Authorization based on tags	Temporary credentials	Service-linked roles
Amazon Kinesis Data Analytics	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Kinesis Data Firehose	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Kinesis Data Streams	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
AWS Lake Formation	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✓ Yes
Amazon Managed Streaming for Apache Kafka (MSK)	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Managed Workflows for Apache Airflow	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon QuickSight	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No

Application integration services

Service	Actions	Resource-level permission	Resource-based policies	Authorization based on tags	Temporary credentials	Service-linked roles
Amazon AppFlow	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon EventBridge	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✗ No
Amazon EventBridge Schemas	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✗ No
Amazon MQ	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Simple Notification Service (Amazon SNS)	✓ Yes	✓ Yes	✓ Yes	✗ No	✓ Yes	✗ No
Amazon Simple Queue Service (Amazon SQS)	✓ Yes	✓ Yes	✓ Yes	✗ No	✓ Yes	✗ No
AWS Step Functions	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Simple Workflow Service (Amazon SWF)	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No

Business applications services

Service	Actions	Resource-level permission	Resource-based policies	Authorization based on tags	Temporary credentials	Service-linked roles
Alexa for Business	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
Amazon Chime	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Honeycode	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
Amazon WorkMail	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes

Satellite services

Service	Actions	Resource-level permission	Resource-based policies	Authorization based on tags	Temporary credentials	Service-linked roles
AWS Ground Station	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No

Internet of Things services

Service	Actions	Resource-level permission	Resource-based policies	Authorization based on tags	Temporary credentials	Service-linked roles
AWS IoT Greengrass	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS IoT Greengrass V2	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS IoT	✓ Yes	✓ Yes	⚠ Partial ¹	✓ Yes	✓ Yes	✗ No
AWS IoT Analytics	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS IoT Core Device Advisor	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS IoT Core for LoRaWAN	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS IoT Device Tester	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS IoT Events	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS IoT SiteWise	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS IoT Things Graph	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No

Service	Actions	Resource-level permission	Resource-based policies	Authorization based on tags	Temporary credentials	Service-linked roles
Fleet Hub for AWS IoT Device Management	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
FreeRTOS	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No

¹ Devices connected to AWS IoT are authenticated by using X.509 certificates or using Amazon Cognito Identities. You can attach AWS IoT policies to an X.509 certificate or Amazon Cognito Identity to control what the device is authorized to do. For more information, see [Security and Identity for AWS IoT](#) in the [AWS IoT Developer Guide](#).

Robotics services

Service	Actions	Resource-level permission	Resource-based policies	Authorization based on tags	Temporary credentials	Service-linked roles
RoboMaker	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes

Quantum Computing Services

Service	Actions	Resource-level permission	Resource-based policies	Authorization based on tags	Temporary credentials	Service-linked roles
Amazon Braket	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes

Blockchain services

Service	Actions	Resource-level permission	Resource-based policies	Authorization based on tags	Temporary credentials	Service-linked roles
Amazon Managed Blockchain	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No

Game development services

Service	Actions	Resource-level permission	Resource-based policies	Authorization based on tags	Temporary credentials	Service-linked roles
Amazon GameLift	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No

AR & VR services

Service	Actions	Resource-level permission	Resource-based policies	Authorization based on tags	Temporary credentials	Service-linked roles
Amazon Sumerian	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No

Customer enablement services

Service	Actions	Resource-level permission	Resource-based policies	Authorization based on tags	Temporary credentials	Service-linked roles
AWS IQ	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS IQ Permissions	✗ No	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Support	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✓ Yes

Customer engagement services

Service	Actions	Resource-level permission	Resource-based policies	Authorization based on tags	Temporary credentials	Service-linked roles
Amazon Connect	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Connect Customer Profiles	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Pinpoint	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Pinpoint Email Service	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Pinpoint SMS and Voice Service	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
Amazon Simple Email Service (Amazon SES)	✓ Yes	✓ Partial ¹	✓ Yes	✓ Yes	⚠ Partial ²	✗ No

¹ You can only use resource-level permissions in policy statements that refer to actions related to sending email, such as ses:SendEmail or ses:SendRawEmail. For policy statements that refer to any other actions, the Resource element can only contain *.

² Only the Amazon SES API supports temporary security credentials. The Amazon SES SMTP interface does not support SMTP credentials that are derived from temporary security credentials.

End user computing services

Service	Actions	Resource-level permission	Resource-based policies	Authorization based on tags	Temporary credentials	Service-linked roles
Amazon AppStream	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
Amazon AppStream 2.0	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon WAM	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
Amazon WorkDocs	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
Amazon WorkLink	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon WorkSpaces	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No

Additional resources

Service	Actions	Resource-level permission	Resource-based policies	Authorization based on tags	Temporary credentials	Service-linked roles
AWS Activate	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Billing and Cost Management	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Budget Service	✓ Yes	✓ Yes	✗ No	✗ No	✗ No	✗ No
AWS Cost and Usage Report	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
AWS Cost Explorer	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Marketplace	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Marketplace Catalog	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
AWS Marketplace Commerce Analytics Service	✓ Yes	✗ No	✗ No	✗ No	✗ No	✗ No
AWS Private Marketplace	✓ Yes	✗ No	✗ No	✗ No	✗ No	✗ No
AWS Savings Plans	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No

IAM JSON policy reference

This section presents detailed syntax, descriptions, and examples of the elements, variables, and evaluation logic of JSON policies in IAM. For more general information, see [Overview of JSON policies \(p. 356\)](#).

This reference includes the following sections.

- [IAM JSON policy elements reference \(p. 628\)](#) — Learn more about the elements that you can use when you create a policy. View additional policy examples and learn about conditions, supported data types, and how they are used in various services.
- [Policy evaluation logic \(p. 666\)](#) — This section describes AWS requests, how they are authenticated, and how AWS uses policies to determine access to resources.
- [Grammar of the IAM JSON policy language \(p. 679\)](#) — This section presents a formal grammar for the language that is used to create policies in IAM.
- [AWS managed policies for job functions \(p. 684\)](#) — This section lists all the AWS managed policies that directly map to common job functions in the IT industry. Use these policies to grant the permissions that are needed to carry out the tasks expected of someone in a specific job function. These policies consolidate permissions for many services into a single policy.
- [AWS global condition context keys \(p. 692\)](#) — This section includes a list of all the AWS global condition keys that you can use to limit permissions in an IAM policy.
- [IAM and AWS STS condition context keys \(p. 708\)](#) — This section includes a list of all the IAM and AWS STS condition keys that you can use to limit permissions in an IAM policy.
- [Actions, Resources, and Condition Keys for AWS Services](#) — This section presents a list of all the AWS API operations that you can use as permissions in an IAM policy. It also includes the service-specific condition keys that can be used to further refine the request.

IAM JSON policy elements reference

JSON policy documents are made up of elements. 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. To learn more about the general structure and purpose of a JSON policy document, see [Overview of JSON policies \(p. 356\)](#).

Some JSON policy elements are mutually exclusive. This means that you cannot create a policy that uses both. For example, you cannot use both `Action` and `NotAction` in the same policy statement. Other pairs that are mutually exclusive include `Principal/NotPrincipal` and `Resource/NotResource`.

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 work with IAM \(p. 611\)](#).

Topics

- [IAM JSON policy elements: Version \(p. 629\)](#)
- [IAM JSON policy elements: Id \(p. 629\)](#)
- [IAM JSON policy elements: Statement \(p. 629\)](#)
- [IAM JSON policy elements: Sid \(p. 630\)](#)
- [IAM JSON policy elements: Effect \(p. 631\)](#)
- [AWS JSON policy elements: Principal \(p. 631\)](#)
- [AWS JSON policy elements: NotPrincipal \(p. 635\)](#)
- [IAM JSON policy elements: Action \(p. 637\)](#)
- [IAM JSON policy elements: NotAction \(p. 638\)](#)
- [IAM JSON policy elements: Resource \(p. 639\)](#)

- [IAM JSON policy elements: NotResource \(p. 640\)](#)
- [IAM JSON policy elements: Condition \(p. 641\)](#)
- [IAM policy elements: Variables and tags \(p. 658\)](#)
- [IAM JSON policy elements: Supported data types \(p. 666\)](#)

IAM JSON policy elements: Version

Disambiguation note

This `Version` JSON policy element is different from a *policy version*. The `Version` policy element is used within a policy and defines the version of the policy language. A policy version, on the other hand, is created when you make changes to a customer managed policy in IAM. The changed policy doesn't overwrite the existing policy. Instead, IAM creates a new version of the managed policy. If you were searching for information about the multiple version support available for managed policies, see [the section called "Versioning IAM policies" \(p. 462\)](#).

The `Version` policy element specifies the language syntax rules that are to be used to process a policy. To use all of the available policy features, include the following `Version` element before the `Statement` element in all of your policies.

```
"Version": "2012-10-17"
```

IAM supports the following `Version` element values:

- 2012-10-17. This is the current version of the policy language, and you should always include a `Version` element and set it to 2012-10-17. Otherwise, you cannot use features such as [policy variables \(p. 658\)](#) that were introduced with this version.
- 2008-10-17. This was an earlier version of the policy language. You might see this version on older existing policies. Do not use this version for any new policies or when you update any existing policies.

If you do not include a `Version` element, the value defaults to 2008-10-17, but newer features, such as policy variables, will not work with your policy. For example, variables such as `#{aws:username}` aren't recognized as variables and are instead treated as literal strings in the policy.

IAM JSON policy elements: Id

The `Id` element specifies an optional identifier for the policy. The ID is used differently in different services.

For services that let you set an `ID` element, we recommend you use a UUID (GUID) for the value, or incorporate a UUID as part of the ID to ensure uniqueness.

```
"Id": "cd3ad3d9-2776-4ef1-a904-4c229d1642ee"
```

Note

Some AWS services (for example, Amazon SQS or Amazon SNS) might require this element and have uniqueness requirements for it. For service-specific information about writing policies, refer to the documentation for the service you're working with.

IAM JSON policy elements: Statement

The `Statement` element is the main element for a policy. This element is required. The `Statement` element can contain a single statement or an array of individual statements. Each individual statement

block must be enclosed in curly braces { }. For multiple statements, the array must be enclosed in square brackets [].

```
"Statement": [{...},{...},{...}]
```

The following example shows a policy that contains an array of three statements inside a single Statement element. (The policy allows you to access your own "home folder" in the Amazon S3 console.) The policy includes the aws:username variable, which is replaced during policy evaluation with the user name from the request. For more information, see [Introduction \(p. 659\)](#).

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

IAM JSON policy elements: 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 JSON policy.

```
"Sid": "1"
```

The `Sid` element supports uppercase letters, lowercase letters, and numbers.

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.

IAM JSON policy elements: Effect

The `Effect` element is required and specifies whether the statement results 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 [Policy evaluation logic \(p. 666\)](#).

AWS JSON policy elements: Principal

Use the `Principal` element in a policy to specify the principal that is allowed or denied access to a resource. You cannot use the `Principal` element in an IAM identity-based policy. You can use it in the trust policies for IAM roles and in resource-based policies. Resource-based policies are policies that you embed directly in a resource. For example, you can embed policies in an Amazon S3 bucket or an AWS KMS customer master key (CMK).

You can specify any of the following principals in a policy:

- AWS account and root user
- IAM users
- Federated users (using web identity or SAML federation)
- IAM roles
- Assumed-role sessions
- AWS services
- Anonymous users (not recommended)

Use the `Principal` element in these ways:

- In IAM roles, use the `Principal` element in the role's trust policy to specify who can assume the role. For cross-account access, you must specify the 12-digit identifier of the trusted account. To learn whether principals in accounts outside of your zone of trust (trusted organization or account) have access to assume your roles, see [What is IAM Access Analyzer?](#).

Note

After you create the role, you can change the account to `"*"` to allow everyone to assume the role. If you do this, we strongly recommend that you limit who can access the role through other means, such as a `Condition` element that limits access to only certain IP addresses. Do not leave your role accessible to everyone!

- In resource-based policies, use the `Principal` element to specify the accounts or users who are allowed to access the resource.

Do not use the `Principal` element in policies that you attach to IAM users and groups. Similarly, you do not specify a principal in the permission policy for an IAM role. In those cases, the principal is implicitly the user that the policy is attached to (for IAM users) or the user who assumes the role (for role access policies). When the policy is attached to an IAM group, the principal is the IAM user in that group who is making the request.

Specifying a principal

You specify a principal using the [Amazon Resource Name \(ARN\) \(p. 601\)](#) or other identifier of the principal. You cannot specify IAM groups and instance profiles as principals.

The following examples show different methods for specifying principals.

Specific AWS accounts

When you use an AWS account identifier as the principal in a policy, you delegate authority to the account. All identities inside the account can access the resource if they have the appropriate IAM permissions attached to explicitly allow access. This includes IAM users and roles in that account. When you specify an AWS account, you can use the account ARN (`arn:aws:iam::AWS-account-ID:root`), or a shortened form that consists of the `AWS:` prefix followed by the account ID.

For example, given an account ID of `123456789012`, you can use either of the following methods to specify that account in the `Principal` element:

```
"Principal": { "AWS": "arn:aws:iam::123456789012:root" }
```

```
"Principal": { "AWS": "123456789012" }
```

You can also specify more than one AWS account as a principal using an array, with any combination of the methods that we previously mentioned.

```
"Principal": {
  "AWS": [
    "arn:aws:iam::123456789012:root",
    "999999999999"
  ]
}
```

Some AWS services support additional options for specifying an account principal. For example, Amazon S3 lets you specify a [canonical user ID](#) using the following format:

```
"Principal": { "CanonicalUser":
  "79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be" }
```

Individual IAM users

You can specify an individual IAM user (or array of users) as the principal, as in the following examples. When you specify more than one principal in the `Principal` element, you grant permissions to each principal. This is a logical OR and not a logical AND, because you are authenticated as one principal at a time.

Note

In a `Principal` element, the user name is case sensitive.

```
"Principal": { "AWS": "arn:aws:iam::AWS-account-ID:user/user-name" }
```

```
"Principal": {
  "AWS": [
    "arn:aws:iam::AWS-account-ID:user/user-name-1",
    "arn:aws:iam::AWS-account-ID:user/UserName2"
  ]
}
```

When you specify users in a `Principal` element, you cannot use a wildcard (*) to mean "all users". Principals must always name a specific user or users.

Important

If your `Principal` element in a role trust policy contains an ARN that points to a specific IAM user, then that ARN is transformed to the user's unique principal ID when the policy is saved. This helps mitigate the risk of someone escalating their privileges by removing and

recreating the user. You don't normally see this ID in the console, because there is also a reverse transformation back to the user's ARN when the trust policy is displayed. However, if you delete the user, then the relationship is broken. The policy no longer applies, even if you recreate the user. That's because the new user has a new principal ID that does not match the ID stored in the trust policy. When this happens, the principal ID shows up in the console because AWS can no longer map it back to a valid ARN. The result is that if you delete and recreate a user referenced in a trust policy's `Principal` element, you must edit the role to replace the now incorrect principal ID with the correct ARN. The ARN is once again transformed into the user's new principal ID when you save the policy.

Federated web identity users

```
"Principal": { "Federated": "cognito-identity.amazonaws.com" }
```

```
"Principal": { "Federated": "www.amazon.com" }
```

```
"Principal": { "Federated": "graph.facebook.com" }
```

```
"Principal": { "Federated": "accounts.google.com" }
```

Federated SAML users

```
"Principal": { "Federated": "arn:aws:iam::AWS-account-ID:saml-provider/provider-name" }
```

IAM roles

```
"Principal": { "AWS": "arn:aws:iam::AWS-account-ID:role/role-name" }
```

Important

If your `Principal` element in a role trust policy contains an ARN that points to a specific IAM role, then that ARN is transformed to the role's unique principal ID when the policy is saved. This helps mitigate the risk of someone escalating their privileges by removing and recreating the role. You don't normally see this ID in the console, because there is also a reverse transformation back to the role's ARN when the trust policy is displayed. However, if you delete the role, then the relationship is broken. The policy no longer applies, even if you recreate the role because the new role has a new principal ID that does not match the ID stored in the trust policy. When this happens, the principal ID shows up in the console because AWS can no longer map it back to a valid ARN. The end result is that if you delete and recreate a role referenced in a trust policy's `Principal` element, you must edit the role to replace the now incorrect principal ID with the correct ARN. The ARN will once again be transformed into the role's new principal ID when you save the policy.

Specific assumed-role sessions

```
"Principal": { "AWS": "arn:aws:sts::AWS-account-ID:assumed-role/role-name/role-session-name" }
```

When you specify an assumed-role session in a `Principal` element, you cannot use a wildcard (*) to mean "all sessions". Principals must always name a specific session.

AWS services

IAM roles that can be assumed by an AWS service are called [service roles \(p. 168\)](#). Service roles must include a trust policy. *Trust policies* are resource-based policies that are attached to a role that define

which principals can assume the role. Some service roles have predefined trust policies. However, in some cases, you must specify the service principal in the trust policy. A *service principal* is an identifier that is used to grant permissions to a service.

The identifier for a service principal includes the service name, and is usually in the following format:

`service-name.amazonaws.com`

The service principal is defined by the service. You can find the service principal for some services by opening [AWS services that work with IAM \(p. 611\)](#), checking whether the service has **Yes** in the **Service-linked role** column, and opening the **Yes** link to view the service-linked role documentation for that service. Find the **Service-Linked Role Permissions** section for that service to view the service principal.

The following example shows a policy that can be attached to a service role. The policy enables two services, Amazon ECS and Elastic Load Balancing, to assume the role. The services can then perform any tasks granted by the permissions policy assigned to the role (not shown). To specify multiple service principals, you do not specify two `Service` elements; you can have only one. Instead, you use an array of multiple service principals as the value of a single `Service` element.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "Service": [  
                    "ecs.amazonaws.com",  
                    "elasticloadbalancing.amazonaws.com"  
                ]  
            },  
            "Action": "sts:AssumeRole"  
        }  
    ]  
}
```

Anonymous users (public)

For resource-based policies, such as Amazon S3 bucket policies, a wildcard (*) in the `Principal` element specifies all users or public access. The following elements are equivalent:

```
"Principal": "*"
```

```
"Principal" : { "AWS" : "*" }
```

You cannot use a wildcard to match part of a name or an ARN.

We strongly recommend that you do not use a wildcard in the `Principal` element in a role's trust policy unless you otherwise restrict access through a `Condition` element in the policy. Otherwise, any IAM user in any account in your [partition \(p. 601\)](#) can access the role.

More information

For more information, see the following:

- [Bucket Policy Examples](#) in the *Amazon Simple Storage Service Developer Guide*
- [Example Policies for Amazon SNS](#) in the *Amazon Simple Notification Service Developer Guide*
- [Amazon SQS Policy Examples](#) in the *Amazon Simple Queue Service Developer Guide*

- [Key Policies](#) in the *AWS Key Management Service Developer Guide*
- [Account Identifiers](#) in the *AWS General Reference*
- [About web identity federation \(p. 177\)](#)

AWS JSON policy elements: NotPrincipal

Use the `NotPrincipal` element to specify the IAM user, federated user, IAM role, AWS account, AWS service, or other principal that is *not* allowed or denied access to a resource. The `NotPrincipal` element enables you to specify an exception to a list of principals. Use this element to deny access to all principals *except* the one named in the `NotPrincipal` element. The syntax for specifying `NotPrincipal` is the same as for specifying [AWS JSON policy elements: Principal \(p. 631\)](#).

You cannot use the `NotPrincipal` element in an IAM identity-based policy. You can use it in the trust policies for IAM roles and in resource-based policies. Resource-based policies are policies that you embed directly in an IAM resource.

Important

Very few scenarios require the use of `NotPrincipal`, and we recommend that you explore other authorization options before you decide to use `NotPrincipal`.

NotPrincipal With Allow

We strongly recommend that you do not use `NotPrincipal` in the same policy statement as "`Effect`": "Allow". Doing so allows all principals *except* the one named in the `NotPrincipal` element. We do not recommend this because the permissions specified in the policy statement will be granted to *all* principals except for the ones specified. By doing this, you might grant access to anonymous (unauthenticated) users.

NotPrincipal With Deny

When you use `NotPrincipal` in the same policy statement as "`Effect`": "Deny", the actions specified in the policy statement are explicitly denied to all principals *except* for the ones specified. When you use `NotPrincipal` with Deny, you must also specify the account ARN of the not-denied principal. Otherwise, the policy might deny access to the entire account containing the principal. Depending on the service that you include in your policy, AWS might validate the account first and then the user. If an assumed-role user (someone who is using a role) is being evaluated, AWS might validate the account first, then the role, and then the assumed-role user. The assumed-role user is identified by the role session name that is specified when they assumed the role. Therefore, we strongly recommend that you explicitly include the ARN for a user's account, or include both the ARN for a role and the ARN for the account containing that role.

Note

As a best practice, you should include the ARNs for the account in your policy. Some services require the account ARN, although this is not required in all cases. Any existing policies without the required ARN will continue to work, but new policies that include these services must meet this requirement. IAM does not track these services, and therefore recommends that you always include the account ARN.

The following examples show how to use `NotPrincipal` and "`Effect`": "Deny" in the same policy statement effectively.

Example Example IAM user in the same or a different account

In the following example, all principals *except* the user named Bob in AWS account 444455556666 are explicitly denied access to a resource. Note that as a best practice, the `NotPrincipal` element contains the ARN of both the user Bob and the AWS account that Bob belongs to

(`arn:aws:iam::444455556666:root`). If the `NotPrincipal` element contained only Bob's ARN, the effect of the policy might be to explicitly deny access to the AWS account that contains the user Bob. In some cases, a user cannot have more permissions than its parent account, so if Bob's account is explicitly denied access then Bob might be unable to access the resource.

This example works as intended when it is part of a policy statement in a resource-based policy that is attached to a resource in either the same or a different AWS account (not 444455556666). This example by itself does not grant access to Bob, it only omits Bob from the list of principals that are explicitly denied. To allow Bob access to the resource, another policy statement must explicitly allow access using `"Effect": "Allow"`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "NotPrincipal": {"AWS": [
                "arn:aws:iam::444455556666:user/Bob",
                "arn:aws:iam::444455556666:root"
            ]},
            "Action": "s3:*",
            "Resource": [
                "arn:aws:s3:::BUCKETNAME",
                "arn:aws:s3:::BUCKETNAME/*"
            ]
        }
    ]
}
```

Example Example IAM role in the same or different account

In the following example, all principals *except* the assumed-role user named cross-account-audit-app in AWS account 444455556666 are explicitly denied access to a resource. As a best practice, the `NotPrincipal` element contains the ARN of the assumed-role user (cross-account-audit-app), the role (cross-account-read-only-role), and the AWS account that the role belongs to (444455556666). If the `NotPrincipal` element was missing the ARN of the role, the effect of the policy might be to explicitly deny access to the role. Similarly, if the `NotPrincipal` element was missing the ARN of the AWS account that the role belongs to, the effect of the policy might be to explicitly deny access to the AWS account and all entities in that account. In some cases, assumed-role users cannot have more permissions than their parent role, and roles cannot have more permissions than their parent AWS account, so when the role or the account is explicitly denied access, the assumed role user might be unable to access the resource.

This example works as intended when it is part of a policy statement in a resource-based policy that is attached to a resource in a different AWS account (not 444455556666). This example by itself does not allow access to the assumed-role user cross-account-audit-app, it only omits cross-account-audit-app from the list of principals that are explicitly denied. To give cross-account-audit-app access to the resource, another policy statement must explicitly allow access using `"Effect": "Allow"`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "NotPrincipal": {"AWS": [
                "arn:aws:sts::444455556666:assumed-role/cross-account-read-only-role/cross-
account-audit-app",
                "arn:aws:iam::444455556666:role/cross-account-read-only-role",
                "arn:aws:iam::444455556666:root"
            ]},
            "Action": "s3:*",
            "Resource": [
                "arn:aws:s3:::Bucket_AccountAudit",
                "arn:aws:s3:::Bucket_AccountAudit/*"
            ]
        }
    ]
}
```

```
        "arn:aws:s3:::Bucket_AccountAudit/*"
    }]
}
```

When you specify an assumed-role session in a `NotPrincipal` element, you cannot use a wildcard (*) to mean "all sessions". Principals must always name a specific session.

IAM JSON policy elements: Action

The `Action` element describes the specific action or actions that will be allowed or denied. Statements must include either an `Action` or `NotAction` element. Each AWS service has its own set of actions that describe tasks that you can perform with that service. For example, the list of actions for Amazon S3 can be found at [Specifying Permissions in a Policy](#) in the *Amazon Simple Storage Service Developer Guide*, the list of actions for Amazon EC2 can be found in the [Amazon EC2 API Reference](#), and the list of actions for AWS Identity and Access Management can be found in the [IAM API Reference](#). To find the list of actions for other services, consult the API reference [documentation](#) for the service.

You specify a value using a service namespace as an action prefix (`iam`, `ec2`, `sqs`, `sns`, `s3`, etc.) followed by the name of the action to allow or deny. The name must match an action that is supported by the service. The prefix and the action name are case insensitive. For example, `iam>ListAccessKeys` is the same as `IAM:listaccesskeys`. The following examples show `Action` elements for different services.

Amazon SQS action

```
"Action": "sqs:SendMessage"
```

Amazon EC2 action

```
"Action": "ec2:StartInstances"
```

IAM action

```
"Action": "iam:ChangePassword"
```

Amazon S3 action

```
"Action": "s3:GetObject"
```

You can specify multiple values for the `Action` element.

```
"Action": [ "sqs:SendMessage", "sqs:ReceiveMessage", "ec2:StartInstances",
  "iam:ChangePassword", "s3:GetObject" ]
```

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 S3 actions.

```
"Action": "s3:*
```

You can also use wildcards (*) as part of the action name. For example, the following `Action` element applies to all IAM actions that include the string `AccessKey`, including `CreateAccessKey`, `DeleteAccessKey`, `ListAccessKeys`, and `UpdateAccessKey`.

```
"Action": "iam:*AccessKey*"
```

Some services let you limit the actions that are available. For example, Amazon SQS lets you make available just a subset of all the possible Amazon SQS actions. In that case, the `*` wildcard doesn't allow complete control of the queue; it allows only the subset of actions that you've shared. For more information, see [Understanding Permissions](#) in the *Amazon Simple Queue Service Developer Guide*.

IAM JSON policy elements: NotAction

`NotAction` is an advanced policy element that explicitly matches everything *except* the specified list of actions. Using `NotAction` can result in a shorter policy by listing only a few actions that should not match, rather than including a long list of actions that will match. When using `NotAction`, you should keep in mind that actions specified in this element are the *only* actions in that are limited. This, in turn, means that all of the applicable actions or services that are not listed are allowed if you use the `Allow` effect. In addition, such unlisted actions or services are denied if you use the `Deny` effect. When you use `NotAction` with the `Resource` element, you provide scope for the policy. This is how AWS determines which actions or services are applicable. For more information, see the following example policy.

NotAction with Allow

You can use the `NotAction` element in a statement with "`Effect`": "Allow" to provide access to all of the actions in an AWS service, except for the actions specified in `NotAction`. You can use it with the `Resource` element to provide scope for the policy, limiting the allowed actions to the actions that can be performed on the specified resource.

The following example allows users to access all of the Amazon S3 actions that can be performed on any S3 resource *except* for deleting a bucket. This does not allow users to use the `ListAllMyBuckets` S3 API operation, because that action requires the `"*"` resource. This policy also does not allow actions in other services, because other service actions are not applicable to the S3 resources.

```
"Effect": "Allow",
"NotAction": "s3:DeleteBucket",
"Resource": "arn:aws:s3:::*
```

Sometimes, you might want to allow access to a large number of actions. By using the `NotAction` element you effectively reverse the statement, resulting in a shorter list of actions. For example, because AWS has so many services, you might want to create a policy that allows the user to do everything except access IAM actions.

The following example allows users to access every action in every AWS service except for IAM.

```
"Effect": "Allow",
"NotAction": "iam:*",
"Resource": "*"
```

Be careful using the `NotAction` element and "`Effect`": "Allow" in the same statement or in a different statement within a policy. `NotAction` matches all services and actions that are not explicitly listed or applicable to the specified resource, and could result in granting users more permissions than you intended.

NotAction with Deny

You can use the `NotAction` element in a statement with "`Effect`": "Deny" to deny access to all of the listed resources except for the actions specified in the `NotAction` element. This combination does not allow the listed items, but instead explicitly denies the actions not listed. You must still allow actions that you want to allow.

The following conditional example denies access to non-IAM actions if the user is not signed in using MFA. If the user is signed in with MFA, then the "`Condition`" test fails and the final "Deny" statement

has no effect. Note, however, that this would not grant the user access to any actions; it would only explicitly deny all other actions except IAM actions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DenyAllUsersNotUsingMFA",
            "Effect": "Deny",
            "NotAction": "iam:*",
            "Resource": "*",
            "Condition": {"BoolIfExists": {"aws:MultiFactorAuthPresent": "false"}}
        }
    ]
}
```

For an example policy that denies access to actions outside of specific Regions, except for actions from specific services, see [AWS: Denies access to AWS based on the requested Region \(p. 403\)](#).

IAM JSON policy elements: 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 [IAM ARNs \(p. 601\)](#).

Each service has its own set of resources. Although you always use an ARN to specify a resource, the details of the ARN for a resource depend on the service and the resource. For information about how to specify a resource, refer to the documentation for the service whose resources you're writing a statement for.

Note

Some services do not let you specify actions for individual resources; instead, any actions that you list in the **Action** or **NotAction** element apply to all resources in that service. In these cases, you use the wildcard ***** in the **Resource** element.

The following example refers to a specific Amazon SQS queue.

```
"Resource": "arn:aws:sqs:us-east-2:account-ID-without-hyphens:queue1"
```

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

```
"Resource": "arn:aws:iam::account-ID-without-hyphens:user/Bob"
```

You can use wildcards as part of the resource ARN. You can use wildcard characters (***** and **?**) within any ARN segment (the parts separated by colons). An asterisk (*****) represents any combination of characters and a question mark (**?**) represents any single character. You can use multiple ***** or **?** characters in each segment, but a wildcard cannot span segments. The following example refers to all IAM users whose path is `/accounting`.

```
"Resource": "arn:aws:iam::account-ID-without-hyphens:user/accounting/*"
```

The following example refers to all items within a specific Amazon S3 bucket.

```
"Resource": "arn:aws:s3:::my_corporate_bucket/*"
```

You can specify multiple resources. The following example refers to two DynamoDB tables.

```
"Resource": [
```

```
[ "arn:aws:dynamodb:us-east-2:account-ID-without-hyphens:table/books_table",
  "arn:aws:dynamodb:us-east-2:account-ID-without-hyphens:table/magazines_table"
]
```

In the Resource element, you can use JSON policy variables (p. 658) 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-2:ACCOUNT-ID-WITHOUT-HYPHENNS:table/
${aws:username}"
  }
}
```

For more information about JSON policy variables, see [IAM policy elements: Variables and tags \(p. 658\)](#).

IAM JSON policy elements: NotResource

NotResource is an advanced policy element that explicitly matches every resource except those specified. Using NotResource can result in a shorter policy by listing only a few resources that should not match, rather than including a long list of resources that will match. This is particularly useful for policies that apply within a single AWS service.

For example, imagine you have a group named `HRPayroll`. Members of `HRPayroll` should not be allowed to access any Amazon S3 resources except the `Payroll` folder in the `HRBucket` bucket. The following policy explicitly denies access to all Amazon S3 resources other than the listed resources. Note, however, that this policy does not grant the user access to any resources.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Deny",
    "Action": "s3:*",
    "NotResource": [
      "arn:aws:s3:::HRBucket/Payroll",
      "arn:aws:s3:::HRBucket/Payroll/*"
    ]
  }
}
```

Normally, to explicitly deny access to a resource you would write a policy that uses "Effect": "Deny" and that includes a Resource element that lists each folder individually. However, in that case, each time you add a folder to `HRBucket`, or add a resource to Amazon S3 that should not be accessed, you must add its name to the list in Resource. If you use a NotResource element instead, users are automatically denied access to new folders unless you add the folder names to the NotResource element.

When using NotResource, you should keep in mind that resources specified in this element are the *only* resources that are not limited. This, in turn, limits all of the resources that would apply to the action. In the example above, the policy affects only Amazon S3 actions, and therefore only Amazon S3 resources. If the action also included Amazon EC2 actions, then the policy would not deny access to any

EC2 resources. To learn which actions in a service allow specifying the ARN of a resource, see [Actions, Resources, and Condition Keys for AWS Services](#).

NotResource with other elements

You should **never** use the "Effect": "Allow", "Action": "*", and "NotResource": "arn:aws:s3:::HRBucket" elements together. This statement is very dangerous, because it allows all actions in AWS on all resources except the HRBucket S3 bucket. This would even allow the user to add a policy to themselves that allows them to access HRBucket. Do not do this.

Be careful using the NotResource element and "Effect": "Allow" in the same statement or in a different statement within a policy. NotResource allows all services and resources that are not explicitly listed, and could result in granting users more permissions than you intended. Using the NotResource element and "Effect": "Deny" in the same statement denies services and resources that are not explicitly listed.

IAM JSON policy elements: Condition

The Condition element (or Condition *block*) lets you specify conditions for when a policy is in effect. The Condition element is optional. In the Condition element, you build expressions in which you use [condition operators \(p. 644\)](#) (equal, less than, etc.) to match the condition keys and values in the policy against keys and values in the request context. To learn more about the request context, see [Request \(p. 5\)](#).

```
"Condition" : { "{condition-operator}" : { "{condition-key}" : "{condition-value}" }}
```

The condition key that you specify can be a [global condition key \(p. 692\)](#) or a service-specific condition key. Global condition keys have the aws: prefix. Service-specific condition keys have the service's prefix. For example, Amazon EC2 lets you write a condition using the ec2:InstanceType key, which is unique to that service. To view service-specific IAM condition keys with the iam: prefix, see [IAM and AWS STS condition context keys \(p. 708\)](#).

Condition key *names* are not case-sensitive. For example, including the aws:SourceIP condition key is equivalent to testing for AWS:SourceIp. Case-sensitivity of condition key *values* depends on the [condition operator \(p. 644\)](#) that you use. For example, the following condition includes the StringEquals operator to ensure that only requests made by johndoe will match. Users named JohnDoe are denied access.

```
"Condition" : { "StringEquals" : { "aws:username" : "johndoe" }}
```

The following condition uses the [StringEqualsIgnoreCase \(p. 644\)](#) operator to match users named johndoe or JohnDoe.

```
"Condition" : { "StringEqualsIgnoreCase" : { "aws:username" : "johndoe" }}
```

Some condition keys support key-value pairs that allow you to specify part of the key name. Examples include the [aws:RequestTag/tag-key \(p. 692\)](#) global condition key, the AWS KMS [kms:EncryptionContext:encryption_context_key](#), and the [ResourceTag/tag-key](#) condition key supported by multiple services. If you use the [ResourceTag/tag-key](#) condition key for a service such as [Amazon EC2](#), then you must specify a key name for the tag-key. **Key names are not case-sensitive.** This means that if you specify "ec2:ResourceTag:TagKey1": "Value1" in the condition element of your policy, then the condition matches a resource tag key named either TagKey1 or tagkey1, but not both. AWS services that support these attributes might allow you to create multiple key names that differ only by case. An example is tagging an Amazon EC2 instance with foo=bar1 and Foo=bar2. When you use a condition such as "ec2:ResourceTag:Foo": "bar1" to allow

access to that resource, the key name matches both tags, but only one value matches. This can result in unexpected condition failures.

Important

As a best practice, make sure that members of your account follow a consistent naming convention when naming key-value pair attributes. Examples include tags or AWS KMS encryption contexts. You can enforce this using the [aws:TagKeys \(p. 705\)](#) condition key for tagging, or the [kms:EncryptionContextKeys](#) for the AWS KMS encryption context.

- For a list of all of the condition operators and a description of how each one works, see [Condition Operators \(p. 644\)](#)
- Unless otherwise specified, all keys can have multiple values. For a description of how to handle condition keys that have multiple values, see [Creating a condition with multiple keys or values \(p. 652\)](#)
- For a list of all of the globally available condition keys, see [AWS global condition context keys \(p. 692\)](#).
- For conditions keys that are defined by each service, see [Actions, Resources, and Condition Keys for AWS Services](#).

The request context

When a [principal \(p. 5\)](#) makes a [request \(p. 5\)](#) to AWS, AWS gathers the request information into a request context. The information is used to evaluate and authorize the request. You can use the Condition element of a JSON policy to test specific conditions against the request context. For example, you can create a policy that uses the [aws:CurrentTime \(p. 695\)](#) condition key to [allow a user to perform actions within only a specific range of dates \(p. 392\)](#).

When a request is submitted, AWS evaluates each condition key in the policy and returns a value of *true*, *false*, *not present*, and occasionally *null* (an empty data string). A key that is not present in the request is not considered a mismatch. For example, the following policy allows removing your own multifactor authentication (MFA) device, but only if you have signed in using MFA in the last hour (3,600 seconds).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowRemoveMfaOnlyIfRecentMfa",
            "Effect": "Allow",
            "Action": [
                "iam:DeactivateMFADevice",
                "iam:DeleteVirtualMFADevice"
            ],
            "Resource": "arn:aws:iam::*:user/${aws:username}",
            "Condition": {
                "NumericLessThanEquals": {"aws:MultiFactorAuthAge": "3600"}
            }
        }
    ]
}
```

The request context can return the following values:

- **True** – If the requester signed in using MFA in the last one hour or less, then the condition returns *true*.
- **False** – If the requester signed in using MFA more than one hour ago, then the condition returns *false*.
- **Not present** – If the requester made a request using their IAM user access keys in the AWS CLI or AWS API, the key is not present. In this case, the key is not present, and it won't match.
- **Null** – For condition keys that are defined by the user, such as passing tags in a request, it is possible to include an empty string. In this case, the value in the request context is *null*. A null value might return true in some cases. For example, if you use the multivalued [ForAllValues \(p. 653\)](#) condition

operator with the [aws:TagKeys \(p. 705\)](#) condition key, you can experience unexpected results if the request context returns `null`. For more information, see [aws:TagKeys \(p. 705\)](#) and [Using multiple keys and values \(p. 653\)](#).

The condition block

The following example shows the basic format of a Condition element:

```
"Condition": {"StringLike": {"s3:prefix": ["$janedoe/*"]}}
```

A value from the request is represented by a condition key, in this case `s3:prefix`. The context key value is compared to a value that you specify as a literal value, such as `janedoe/*`. The type of comparison to make is specified by the [condition operator \(p. 644\)](#) (here, `StringLike`). You can create conditions that compare strings, dates, numbers, and more using typical Boolean comparisons such as equals, greater than, and less than. When you use [string operators \(p. 644\)](#) or [ARN operators \(p. 649\)](#), you can also use a [policy variable \(p. 658\)](#) in the condition value. The following example includes the `aws:username` variable.

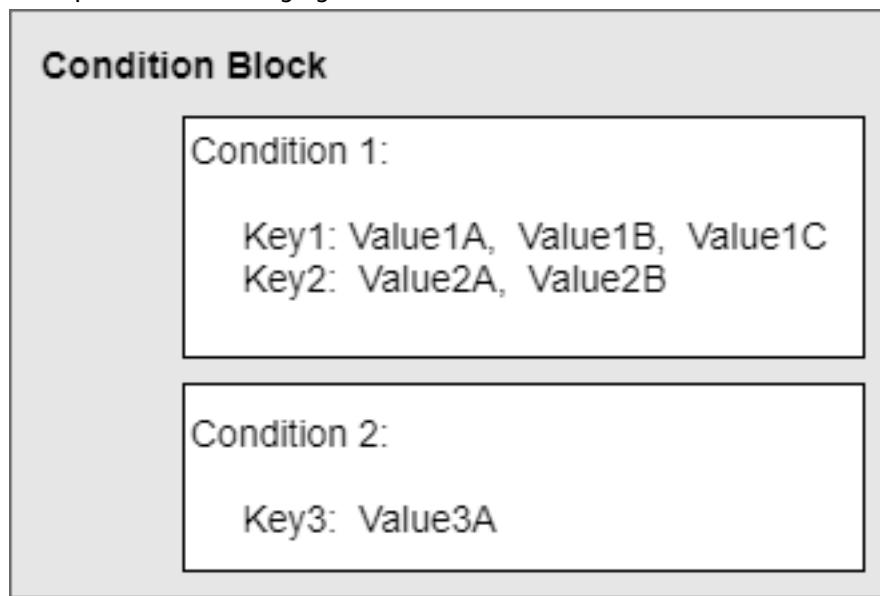
```
"Condition": {"StringLike": {"s3:prefix": ["${aws:username}/*"]}}
```

Under some circumstances, keys can contain multiple values. For example, a request to Amazon DynamoDB might ask to return or update multiple attributes from a table. A policy for access to DynamoDB tables can include the `dynamodb:Attributes` key, which contains all the attributes listed in the request. You can test the multiple attributes in the request against a list of allowed attributes in a policy by using set operators in the Condition element. For more information, see [Creating a condition with multiple keys or values \(p. 652\)](#).

When the policy is evaluated during a request, AWS replaces the key with the corresponding value from the request. (In this example, AWS would use the date and time of the request.) The condition is evaluated to return true or false, which is then factored into whether the policy as a whole allows or denies the request.

Multiple values in a condition

A Condition element can contain multiple conditions, and each condition can contain multiple key-value pairs. The following figure illustrates this.



For more information, see [Creating a condition with multiple keys or values \(p. 652\)](#).

IAM JSON policy elements: Condition operators

Use condition operators in the Condition element to match the condition key and value in the policy against values in the request context. For more information about the Condition element, see [IAM JSON policy elements: Condition \(p. 641\)](#).

The condition operator that you can use in a policy depends on the condition key you choose. You can choose a global condition key or a service-specific condition key. To learn which condition operator you can use for a global condition key, see [AWS global condition context keys \(p. 692\)](#). To learn which condition operator you can use for a service-specific condition key, see [Actions, Resources, and Condition Keys for AWS Services](#) and choose the service that you want to view.

Important

If the key that you specify in a policy condition is not present in the request context, the values do not match. This applies to all condition operators except [...IfExists \(p. 650\)](#) and [Null check \(p. 651\)](#). These operators test whether the key is present (exists) in the request context.

The condition operators can be grouped into the following categories:

- [String \(p. 644\)](#)
- [Numeric \(p. 646\)](#)
- [Date and time \(p. 646\)](#)
- [Boolean \(p. 647\)](#)
- [Binary \(p. 648\)](#)
- [IP address \(p. 648\)](#)
- [Amazon Resource Name \(ARN\) \(p. 649\)](#) (available for only some services.)
- [...IfExists \(p. 650\)](#) (checks if the key value exists as part of another check)
- [Null check \(p. 651\)](#) (checks if the key value exists as a standalone check)

String condition operators

String condition operators let you construct Condition elements that restrict access based on comparing a key to a string value.

Condition operator	Description
StringEquals	Exact matching, case sensitive
StringNotEquals	Negated matching
StringEqualsIgnoreCase	Exact matching, ignoring case
StringNotEqualsIgnoreCase	Negated matching, ignoring case
StringLike	Case-sensitive matching. The values can include a multi-character match wildcard (*) or a single-character match wildcard (?) anywhere in the string. Note If a key contains multiple values, StringLike can be qualified with set operators — ForAllValues:StringLike and ForAnyValue:StringLike . For more information,

Condition operator	Description
	see Creating a condition with multiple keys or values (p. 652) .
StringNotLike	Negated case-sensitive matching. The values can include a multi-character match wildcard (*) or a single-character match wildcard (?) anywhere in the string.

For example, the following statement contains a Condition element that uses the StringEquals condition operator with the aws:PrincipalTag key to specify that the principal making the request must be tagged with the iamuser-admin job category.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "iam:*AccessKey*",
    "Resource": "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:user/*",
    "Condition": {"StringEquals": {"aws:PrincipalTag/job-category": "iamuser-admin"}}
  }
}
```

If the key that you specify in a policy condition is not present in the request context, the values do not match. In this example, the aws:PrincipalTag/job-category key is present in the request context if the principal is using an IAM user with attached tags. It is also included for a principal using an IAM role with attached tags or session tags. If a user without the tag attempts to view or edit an access key, the condition returns false and the request is implicitly denied by this statement.

You can use a [policy variable \(p. 658\)](#) with the String condition operator.

The following example uses the StringLike condition operator to perform string matching with a [policy variable \(p. 658\)](#) 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. The policy allows the specified actions on an S3 bucket as long as the s3:prefix matches any one of the specified patterns.

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

```

        "Resource": [
            "arn:aws:s3:::BUCKET-NAME/home/${aws:username}",
            "arn:aws:s3:::BUCKET-NAME/home/${aws:username}/*"
        ]
    }
}

```

For an example of a policy that shows how to use the `Condition` element to restrict access to resources based on an application ID and a user ID for web identity federation, see [Amazon S3: Allows Amazon Cognito users to access objects in their bucket \(p. 432\)](#).

Numeric condition operators

Numeric condition operators let you construct `Condition` elements that restrict access based on comparing a key to an integer or decimal value.

Condition operator	Description
<code>NumericEquals</code>	Matching
<code>NumericNotEquals</code>	Negated matching
<code>NumericLessThan</code>	"Less than" matching
<code>NumericLessThanEquals</code>	"Less than or equals" matching
<code>NumericGreaterThan</code>	"Greater than" matching
<code>NumericGreaterThanEquals</code>	"Greater than or equals" matching

For example, the following statement contains a `Condition` element that uses the `NumericLessThanEquals` condition operator 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"}}
    }
}

```

If the key that you specify in a policy condition is not present in the request context, the values do not match. In this example, the `s3:max-keys` key is always present in the request when you perform the `ListBucket` operation. If this policy allowed all Amazon S3 operations, then only the operations that include the `max-keys` context key with a value of less than or equal to 10 would be allowed.

You can not use a [policy variable \(p. 658\)](#) with the `Numeric` condition operator.

Date condition operators

Date condition operators let you construct `Condition` elements that restrict access based on comparing a key to a date/time value. You use these condition operators 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 condition operators.

Condition operator	Description
DateEquals	Matching a specific date
DateNotEquals	Negated matching
DateLessThan	Matching before a specific date and time
DateLessThanEquals	Matching at or before a specific date and time
DateGreaterThan	Matching after a specific date and time
DateGreaterThanOrEqual	Matching at or after a specific date and time

For example, the following statement contains a Condition element that uses the DateGreaterThan condition operator with the aws:TokenIssueTime key. This condition specifies that the temporary security credentials used to make the request were issued in 2020. This policy can be updated programmatically every day to ensure that account members use fresh credentials.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "iam:*AccessKey*",
    "Resource": "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:user/*",
    "Condition": {"DateGreaterThan": {"aws:TokenIssueTime": "2020-01-01T00:00:01Z"}}
  }
}
```

If the key that you specify in a policy condition is not present in the request context, the values do not match. The aws:TokenIssueTime key is present in the request context only when the principal uses temporary credentials to make the request. They key is not present in AWS CLI, AWS API, or AWS SDK requests that are made using access keys. In this example, if an IAM user attempts to view or edit an access key, the request is denied.

You can not use a [policy variable \(p. 658\)](#) with the Date condition operator.

Boolean condition operators

Boolean conditions let you construct Condition elements that restrict access based on comparing a key to "true" or "false."

Condition operator	Description
Bool	Boolean matching

For example, the following statement uses the Bool condition operator with the aws:SecureTransport key to specify that the request must use SSL.

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

```

    "Effect": "Allow",
    "Action": "iam:*AccessKey*",
    "Resource": "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:user/*",
    "Condition": {"Bool": {"aws:SecureTransport": "true"}}
}
}

```

If the key that you specify in a policy condition is not present in the request context, the values do not match. The `aws:SecureTransport` key is always present in the request context.

You can not use a [policy variable \(p. 658\)](#) with the Boolean condition operator.

Binary condition operators

The `BinaryEquals` condition operator let you construct Condition elements that test key values that are in binary format. It compares the value of the specified key byte for byte against a [base-64](#) encoded representation of the binary value in the policy.

```

"Condition" : {
    "BinaryEquals": {
        "key" : "QmluYXJ5VmFsdVVJbkJhc2U2NA=="
    }
}

```

If the key that you specify in a policy condition is not present in the request context, the values do not match.

You can not use a [policy variable \(p. 658\)](#) with the `Binary` condition operator.

IP address condition operators

IP address condition operators let you construct Condition elements that restrict access based on comparing a key to an IPv4 or IPv6 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 or 2001:DB8:1234:5678::/64). If you specify an IP address without the associated routing prefix, IAM uses the default prefix value of /32.

Some AWS services support IPv6, using :: to represent a range of 0s. To learn whether a service supports IPv6, see the documentation for that service.

Condition operator	Description
<code>IpAddress</code>	The specified IP address or range
<code>NotIpAddress</code>	All IP addresses except the specified IP address or range

For example, the following statement uses the `IpAddress` condition operator with the `aws:SourceIp` key to specify that the request must come from the IP range 203.0.113.0 to 203.0.113.255.

```

{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": "iam:*AccessKey*",
        "Resource": "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:user/*",
        "Condition": {"IpAddress": {"aws:SourceIp": "203.0.113.0/24"}}
}

```

```
}
```

The `aws:SourceIp` condition key resolves to the IP address that the request originates from. If the requests originates from an Amazon EC2 instance, `aws:SourceIp` evaluates to the instance's public IP address.

If the key that you specify in a policy condition is not present in the request context, the values do not match. The `aws:SourceIp` key is always present in the request context, except when the requester uses a VPC endpoint to make the request. In this case, the condition returns `false` and the request is implicitly denied by this statement.

You can not use a [policy variable \(p. 658\)](#) with the `IP Address` condition operator.

The following example shows how to mix IPv4 and IPv6 addresses to cover all of your organization's valid IP addresses. We recommend that you augment your organization's policies with your IPv6 address ranges in addition to IPv4 ranges you already have to ensure the policies continue to work as you make the transition to IPv6.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "someService:*",
      "Resource": "*",
      "Condition": {
        "IpAddress": {
          "aws:SourceIp": [
            "203.0.113.0/24",
            "2001:DB8:1234:5678::/64"
          ]
        }
      }
    }
  ]
}
```

The `aws:SourceIp` condition key works only in a JSON policy if you are calling the tested API directly as a user. If you instead use a service to call the target service on your behalf, the target service sees the IP address of the calling service rather than the IP address of the originating user. This can happen, for example, if you use AWS CloudFormation to call Amazon EC2 to construct instances for you. There is currently no way to pass the originating IP address through a calling service to the target service for evaluation in a JSON policy. For these types of service API calls, do not use the `aws:SourceIp` condition key.

[Amazon Resource Name \(ARN\) condition operators](#)

Amazon Resource Name (ARN) condition operators let you construct Condition elements that restrict access based on comparing a key to an ARN. The ARN is considered a string. Not all services support comparing ARNs using this operator. If the ARN condition operator doesn't work, then try using [string condition operators \(p. 644\)](#).

Condition operator	Description
<code>ArnEquals</code> , <code>ArnLike</code>	Case-sensitive 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 (?). These behave identically.

Condition operator	Description
ArnNotEquals, ArnNotLike	Negated matching for ARN. These behave identically.

You can use a [policy variable \(p. 658\)](#) with the ARN condition operator.

The following resource-based policy example shows a policy attached to an Amazon SQS queue to which you want to send SNS messages. It gives Amazon SNS permission to send messages to the queue (or queues) of your choice, but only if the service is sending the messages on behalf of a particular Amazon SNS topic (or topics). You specify the queue in the `Resource` field, and the Amazon SNS topic as the value for the `SourceArn` key.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Principal": {"AWS": "123456789012"},
    "Action": "SQS:SendMessage",
    "Resource": "arn:aws:sqs:REGION:123456789012:QUEUE-ID",
    "Condition": {"ArnEquals": {"aws:SourceArn": "arn:aws:sns:REGION:123456789012:TOPIC-ID"}}
  }
}
```

If the key that you specify in a policy condition is not present in the request context, the values do not match. The `aws:SourceArn` key is present in the request context only if a resource triggers a service to call another service on behalf of the resource owner. If an IAM user attempts to perform this operation directly, the condition returns `false` and the request is implicitly denied by this statement.

[...IfExists condition operators](#)

You can add `IfExists` to the end of any condition operator name except the `Null` condition—for example, `StringLikeIfExists`. You do this to say "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, evaluate the condition element as true." Other condition elements in the statement can still result in a nonmatch, but not a missing key when checked with `...IfExists`.

[Example using IfExists](#)

Many condition keys describe information about a certain type of resource and only exist when accessing that type of resource. These condition keys are not present on other types of resources. This doesn't cause an issue when the policy statement applies to only one type of resource. However, there are cases where a single statement can apply to multiple types of resources, such as when the policy statement references actions from multiple services or when a given action within a service accesses several different resource types within the same service. In such cases, including a condition key that applies to only one of the resources in the policy statement can cause the `Condition` element in the policy statement to fail such that the statement's `"Effect"` does not apply.

For example, consider the following policy example:

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Sid": "THISPOLICYDOESNOTWORK",
    "Effect": "Allow",
    "Action": "ec2:RunInstances",
    "Resource": "*",
    "Condition": {"StringLike": {"ec2:InstanceType": [
      "t1.micro",
      "m1.small"
    ]}}
  }
}
```

```

        "t1.*",
        "t2.*",
        "m3.*"
    ]})
}
}

```

The *intent* of the preceding policy is to enable the user to launch any instance that is type t1, t2 or m3. However, launching an instance actually requires accessing many resources in addition to the instance itself; for example, images, key pairs, security groups, etc. The entire statement is evaluated against every resource that is required to launch the instance. These additional resources do not have the `ec2:InstanceType` condition key, so the `StringLike` check fails, and the user is not granted the ability to launch *any* instance type. To address this, use the `StringLikeIfExists` condition operator instead. This way, the test only happens if the condition key exists. You could read the following as: "If the resource being checked has an `ec2:InstanceType`" condition key, then allow the action only if the key value begins with `"t1.*"`, `"t2.*"`, or `"m3.*"`. If the resource being checked does not have that condition key, then don't worry about it." The `DescribeActions` statement includes the actions required to view the instance in the console.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "RunInstances",
      "Effect": "Allow",
      "Action": "ec2:RunInstances",
      "Resource": "*",
      "Condition": {
        "StringlikeIfExists": {
          "ec2:InstanceType": [
            "t1.*",
            "t2.*",
            "m3.*"
          ]
        }
      },
      {
        "Sid": "DescribeActions",
        "Effect": "Allow",
        "Action": [
          "ec2:DescribeImages",
          "ec2:DescribeInstances",
          "ec2:DescribeVpcs",
          "ec2:DescribeKeyPairs",
          "ec2:DescribeSubnets",
          "ec2:DescribeSecurityGroups"
        ],
        "Resource": "*"
      }
    ]
  }
}

```

Condition operator to check existence of condition keys

Use a `Null` condition operator 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 — it is null) or `false` (the key exists and its value is not null).

You can not use a [policy variable \(p. 658\)](#) with the `Null` condition operator.

For example, you can use this condition operator to determine whether a user is using their own credentials for the operation or temporary credentials. If the user is using temporary credentials, then the key `aws:TokenIssueTime` exists and has a value. The following example shows a condition that

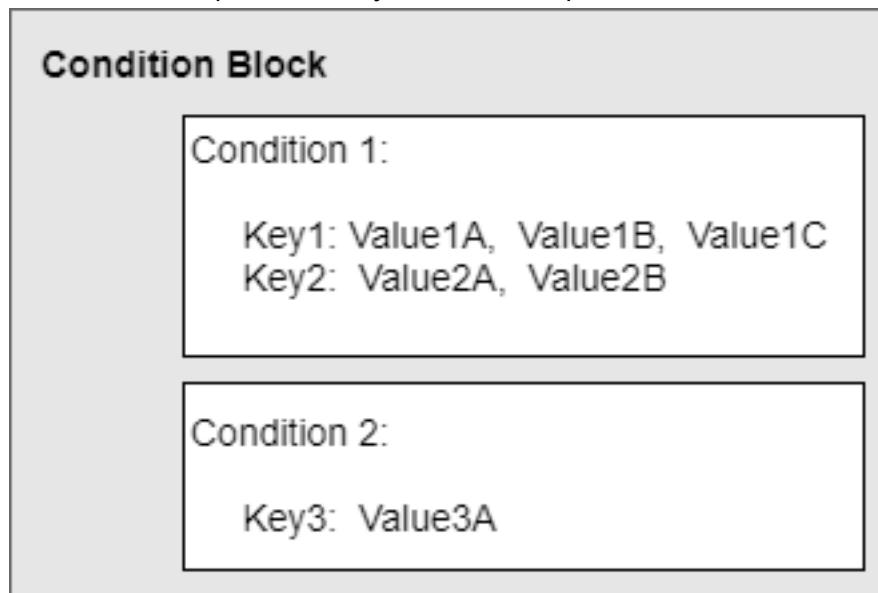
states that the user must not be using temporary credentials (the key must not exist) for the user to use the Amazon EC2 API.

```
{  
    "Version": "2012-10-17",  
    "Statement":{  
        "Action": "ec2:*",  
        "Effect": "Allow",  
        "Resource": "*"  
        "Condition": {"Null": {"aws:TokenIssueTime": "true"} }  
    }  
}
```

Creating a condition with multiple keys or values

You can use the `Condition` element of a policy to test multiple keys or multiple values for a single key in a request. When you make a request to AWS, either programmatically or through the AWS Management Console, your request includes information about your principal, operation, tags, and more. To learn about information and data included in a request, see [Request \(p. 5\)](#). You can use condition keys to test the values of the matching keys in the request. For example, you can use a condition key to control access to specific attributes of a DynamoDB table or to an Amazon EC2 instance based on tags.

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



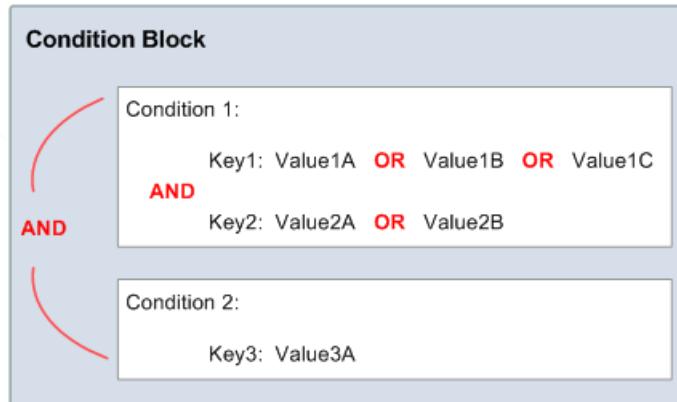
Topics

- [Evaluation logic for conditions with multiple keys or values \(p. 652\)](#)
- [Using multiple keys and values \(p. 653\)](#)
- [Examples of using multiple values with condition set operators \(p. 654\)](#)
- [Evaluation logic for multiple values with condition set operators \(p. 656\)](#)

Evaluation logic for conditions with multiple keys or values

If your policy has multiple condition operators or multiple keys attached to a single condition operator, the conditions are evaluated using a logical `AND`. If a single condition operator includes multiple values

for one key, that condition operator is evaluated using a logical OR. All conditions must resolve to true to trigger the desired Allow or Deny effect.

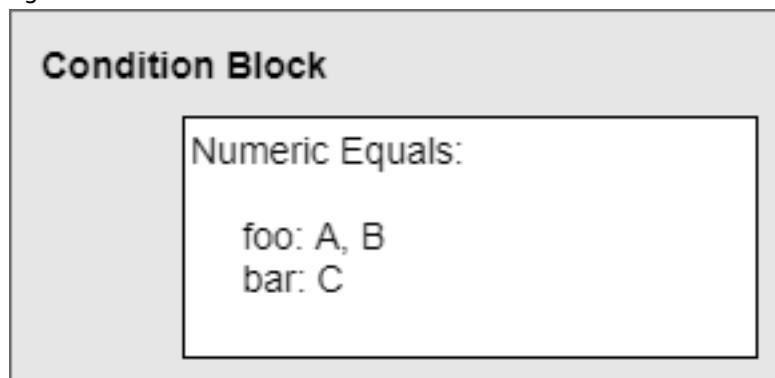


Using multiple keys and values

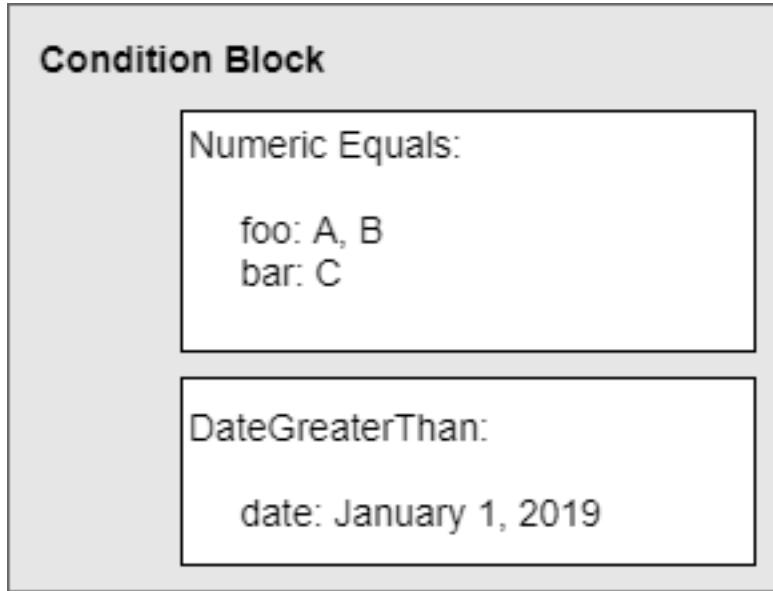
For requests that include multiple values for a single key, you must enclose the conditions within brackets like an array ("Key2":["Value2A", "Value2B"]). You must also use the `ForAllValues` or `ForAnyValue` set operators with the [StringLike condition operator \(p. 644\)](#). These qualifiers add set-operation functionality to the condition operator so that you can test multiple request values against multiple condition values.

- `ForAllValues` – Tests whether the value of every member of the request set is a subset of the condition key set. The condition returns true if every key value in the request matches at least one value in the policy. It also returns true if there are no keys in the request, or if the key values resolve to a null data set, such as an empty string.
- `ForAnyValue` – Tests whether at least one member of the set of request values matches at least one member of the set of condition key values. The condition returns true if any one of the key values in the request matches any one of the condition values in the policy. For no matching key or a null dataset, the condition returns false.

Assume that 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.



Assume that you also want to restrict John's access to after January 1, 2019. You would add another condition, `DateGreaterThanOrEqual`, with a date equal to January 1, 2019. The condition block would then look like the following figure.



AWS has predefined condition operators and keys (like `aws:CurrentTime`). Individual AWS services also define service-specific keys.

As an example, assume that you want to let user John access your Amazon SQS queue under the following conditions:

- The time is after 12:00 p.m. on 7/16/2019
- The time is before 3:00 p.m. on 7/16/2019
- The request 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 condition operators, 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 condition operators are evaluated using AND.

```
"Condition" : {
    "DateGreaterThanOrEqual" : {
        "aws:CurrentTime" : "2019-07-16T12:00:00Z"
    },
    "DateLessThan" : {
        "aws:CurrentTime" : "2019-07-16T15:00:00Z"
    },
    "IpAddress" : {
        "aws:SourceIp" : ["192.0.2.0/24", "203.0.113.0/24"]
    }
}
```

Examples of using multiple values with condition set operators

You can create a policy to test multiple values in a request against one or more values that you specify in the policy. Assume that you have an Amazon DynamoDB table named `Thread` that is used to store information about threads in a technical support forum. The table has attributes named `ID`, `UserName`, `PostDateTime`, `Message`, and `Tags`.

```
{
  ID=101
  UserName=Bob
  PostDateTime=20130930T231548Z
  Message="A good resource for this question is docs.aws.amazon.com"
  Tags=["AWS", "Database", "Security"]
}
```

For information about how set operators are used in DynamoDB to implement fine-grained access to individual data items and attributes, see [Fine-Grained Access Control for DynamoDB](#) in the *Amazon DynamoDB Developer Guide*.

You can create a policy that allows users to see only the `PostDateTime`, `Message`, and `Tags` attributes. If the user's request contains any of these attributes, it is allowed. But if the request contains any other attributes (for example, `ID`), the request is denied. Logically speaking, you want to create a list of allowed attributes (`PostDateTime`, `Message`, `Tags`). You also want to indicate in the policy that all of the user's requested attributes must be in that list of allowed attributes.

The following example policy shows how to use the `ForAllValues` qualifier with the `StringEquals` condition operator. The condition allows a user to request *only* the attributes `ID`, `Message`, or `Tags` from the DynamoDB table named `Thread`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "dynamodb:GetItem",
      "Resource": "arn:aws:dynamodb:*:*:table/Thread",
      "Condition": {
        "ForAllValues:StringEquals": {
          "dynamodb:Attributes": [
            "ID",
            "Message",
            "Tags"
          ]
        }
      }
    }
  ]
}
```

Assume the user makes a request to DynamoDB to get the attributes `Message` and `Tags` from the `Thread` table. In that case, the request is allowed because the user's requested attributes all match values specified in the policy. The `GetItem` operation requires the user to pass the `ID` attribute as the database table key, which is also allowed in the policy. However, if the user's request includes the `UserName` attribute, the request fails. The reason is that `UserName` is not within the list of allowed attributes and the `ForAllValues` qualifier requires all requested values to be listed in the policy.

Important

If you use `dynamodb:Attributes`, you must specify the names of all of the primary key and index key attributes for the table. You must also specify any secondary indexes that are listed in the policy. Otherwise, DynamoDB can't use these key attributes to perform the requested action.

Alternatively, you might want to make sure that users are explicitly forbidden to include some attributes in a request, such as the `ID` and `UserName` attributes. For example, you might exclude attributes when the user is updating the DynamoDB table, because an update (`PUT` operation) should not change certain attributes. In that case, you create a list of forbidden attributes (`ID`, `UserName`). If any of the user's requested attributes match any of the forbidden attributes, the request is denied.

The following example shows how to use the `ForAnyValue` qualifier to deny access to the `ID` and `PostDateTime` attributes if the user tries to perform the `PutItem` action. That is, if the user tries to update either of those attributes in the `Thread` table. Notice that the `Effect` element is set to `Deny`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": "dynamodb:PutItem",
            "Resource": "arn:aws:dynamodb:*:*:table/Thread",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "dynamodb:Attributes": [
                        "ID",
                        "PostDateTime"
                    ]
                }
            }
        }
    ]
}
```

Assume that the user makes a request to update the `PostDateTime` and `Message` attributes of the `Thread` table. The `ForAnyValue` qualifier determines whether any of the requested attributes appear in the list in the policy. In this case, there is one match (`PostDateTime`), so the condition is true. Assuming the other values in the request also match (for example, the resource), the overall policy evaluation returns true. Because the policy's effect is `Deny`, the request is denied.

Imagine the user instead makes a request to perform `PutItem` with just the `UserName` attribute. None of the attributes in the request (just `UserName`) match any of attributes listed in the policy (`ID`, `PostDateTime`). The condition returns false, so the effect of the policy (`Deny`) is also false, and the request is not denied by this policy. (For the request to succeed, it must be explicitly allowed by a different policy. It is not explicitly denied by this policy, but all requests are implicitly denied.)

Warning

When you use the `ForAllValues` condition operator, it returns true if there are no keys in the request, or if the key values resolve to a null data set, such as an empty string. To require that the request includes at least one value, you must use another condition in the policy. For an example, see [Controlling access during AWS requests \(p. 388\)](#).

Evaluation logic for multiple values with condition set operators

This section discusses the specifics of the evaluation logic used with the `ForAllValues` and `ForAnyValue` operators. The following table illustrates possible keys that might be included in a request (`PostDateTime` and `UserName`) and a policy condition that includes the values `PostDateTime`, `Message`, and `Tags`.

Key (in the request)	Condition value (in the policy)
<code>PostDateTime</code>	<code>PostDateTime</code>
<code>UserName</code>	<code>Message</code>
	<code>Tags</code>

The evaluation for the combination is this:

<code>PostDateTime</code> matches <code>PostDateTime</code> ?

PostDateTime matches Message?
PostDateTime matches Tags?
UserName matches PostDateTime?
UserName matches Message?
UserName matches Tags?

- **ForAllValues.** If every key in the request (PostDateTime or UserName) matches at least one condition value in the policy (PostDateTime, Message, Tags), the condition operator returns true. Stated another way, in order for the condition to be true, (PostDateTime must equal PostDateTime, Message, or Tags) *and* (UserName must equal PostDateTime, Message, or Tags).
- **ForAnyValue.** If any combination of request value and policy value (any one of the six in the example) returns true, the condition operator returns true.

The following policy includes a **ForAllValues** qualifier:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "dynamodb:GetItem",
            "Resource": "arn:aws:dynamodb:*:::table/Thread",
            "Condition": {
                "ForAllValues:StringEquals": {
                    "dynamodb:Attributes": [
                        "PostDateTime",
                        "Message",
                        "Tags"
                    ]
                }
            }
        }
    ]
}
```

Suppose that the user makes a request to DynamoDB to get the attributes PostDateTime and UserName. The evaluation for the combination is this:

PostDateTime matches PostDateTime?	True
PostDateTime matches Message?	False
PostDateTime matches Tags?	False
UserName matches PostDateTime?	False
UserName matches Message?	False
UserName matches Tags?	False

The policy includes the **ForAllValues** condition operator modifier, meaning that there must be at least one match for PostDateTime and one match for UserName. There's no match for UserName, so the condition operator returns false, and the policy does not allow the request.

The following policy includes a `ForAnyValue` qualifier:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": "dynamodb:PutItem",
            "Resource": "arn:aws:dynamodb:*:*:table/Thread",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "dynamodb:Attributes": [
                        "ID",
                        "PostDateTime"
                    ]
                }
            }
        }
    ]
}
```

Notice that the policy includes `"Effect": "Deny"` and the action is `PutItem`. Imagine that the user makes a `PutItem` request that includes the attributes `UserName`, `Message`, and `PostDateTime`. The evaluation is this:

<code>UserName</code> matches <code>ID</code> ?	<code>False</code>
<code>UserName</code> matches <code>PostDateTime</code> ?	<code>False</code>
<code>Messages</code> matches <code>ID</code> ?	<code>False</code>
<code>Message</code> matches <code>PostDateTime</code> ?	<code>False</code>
<code>PostDateTime</code> matches <code>ID</code> ?	<code>False</code>
<code>PostDateTime</code> matches <code>PostDateTime</code> ?	<code>True</code>

With the modifier `ForAnyValue`, if any one of these tests returns true, the condition returns true. The last test returns true, so the condition is true; because the `Effect` element is set to `Deny`, the request is denied.

Note

If the key values in the request resolve to an empty data set (for example, an empty string), a condition operator modified by `ForAllValues` returns true. In addition, a condition operator modified by `ForAnyValue` returns false.

IAM policy elements: Variables and tags

Use AWS Identity and Access Management (IAM) policy variables as placeholders when you don't know the exact value of a resource or condition key when you write the policy.

Note

If AWS cannot resolve a variable, this might cause the entire statement to be invalid. For example, if you use the `aws:TokenIssueTime` variable, the variable resolves to a value only when the requester authenticated using temporary credentials (an IAM role). To prevent variables from causing invalid statements, use the [...IfExists condition operator](#) (p. 650)

Topics

- [Introduction \(p. 659\)](#)
- [Tags as policy variables \(p. 660\)](#)
- [Where you can use policy variables \(p. 661\)](#)

- [Request information that you can use for policy variables \(p. 663\)](#)
- [For more information \(p. 665\)](#)

Introduction

In IAM policies, many actions allow you to provide a name for the specific resources that you want to control access to. For example, the following policy allows the user to 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. You might want to generalize the policy so it works for many users without having to make a unique copy of the policy for each user. For example, consider writing a policy to allow each user to have access to his or her own objects in an Amazon S3 bucket, as in the previous example. But don't create a separate policy for each user that explicitly specifies the user's name as part of the resource. Instead, create a single group policy that works for any user in that group.

You can do this by using *policy variables*, a feature that lets you specify placeholders in a policy. When the policy is evaluated, the policy variables are replaced with values that come from the context of 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 replaces the variable `#{aws:username}` with the [friendly name \(p. 600\)](#) of the actual current user. This means that a single policy applied to a group of users can control access to a bucket by using the user name as part of the resource's name.

The variable is marked using a \$ prefix followed by a pair of curly braces ({}). Inside the \${ } characters, you can include the name of the value from the request that you want to use in the policy. The values you can use are discussed later on this page.

Note

In order to use policy variables, you must include the `Version` element in a statement, and the version must be set to a version that supports policy variables. Variables were introduced in version 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 an appropriate version date, variables like `#{aws:username}` are treated as literal strings in the policy.

A `Version` policy element is different from a policy version. The `Version` policy element is used within a policy and defines the version of the policy language. A policy version, on the other hand, is created when you change a customer managed policy in IAM. The changed policy doesn't overwrite the existing policy. Instead, IAM creates a new version of the managed policy.

To learn more about the `Version` policy element see [the section called "Version" \(p. 629\)](#). To learn more about policy versions, see [the section called "Versioning IAM policies" \(p. 462\)](#).

You can use policy variables in a similar way to allow each user to manage his or her own access keys. A policy that allows a user to programmatically change the access key for user David looks like this:

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

If this policy is attached to user David, that user can change his own access key. As with the policies for accessing user-specific Amazon S3 objects, you would have to create a separate policy for each user that includes the user's name. You would then attach each policy to the individual users.

By using a policy variable, you can create a policy like this:

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

When you use a policy variable for the user name like this, you don't have to have a separate policy for each individual user. Instead, you can attach this new policy to an IAM group that includes everyone who should be allowed to manage their own access keys. When a user makes a request to modify his or her access key, IAM substitutes the user name from the current request for the `#{aws:username}` variable and evaluates the policy.

Tags as policy variables

In some AWS services you can attach your own custom attributes to resources that are created by those services. For example, you can apply tags to Amazon S3 buckets or to IAM users and roles. These tags are

key-value pairs. You define the tag key name and the value associated with that key name. For example, you might create a tag with a **department** key and a **Human Resources** value. For more information about tagging IAM entities, see [Tagging IAM users and roles \(p. 289\)](#). For information about tagging resources created by other AWS services, see the documentation for that service. For information about using Tag Editor, see [Working with Tag Editor](#) in the *AWS Management Console User Guide*.

You can tag IAM identities to simplify discovering, organizing, and tracking your IAM resources. You can also tag IAM identities to control access to resources or to tagging itself. To learn more about using tags to control access, see [Controlling access to and for IAM users and roles using IAM resource tags \(p. 384\)](#).

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

You can use a policy variable in the **Resource** element, but only in the resource portion of the ARN. This portion of the ARN appears after the 5th colon (:). You can't use a variable to replace parts of the ARN before the 5th colon, such as the service or account. For more information about the ARN format, see [IAM ARNs \(p. 601\)](#).

The following policy might be attached to a group. It gives each of the users in the group full programmatic access to a user-specific object (their own "home directory") in Amazon S3.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": ["s3>ListBucket"],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::mybucket"],
      "Condition": {"StringLike": {"s3:prefix": ["${aws:username}/*"]}}
    },
    {
      "Action": [
        "s3GetObject",
        "s3PutObject"
      ],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::mybucket/${aws:username}/*"]
    }
  ]
}
```

Note

This example uses the `aws:username` key, which returns the user's friendly name (like "Adele" or "David"). Under some circumstances, you might want to use the `aws:userId` key instead, which is a globally unique value. For more information, see [Unique identifiers \(p. 604\)](#).

The following policy might be used for an IAM group. It gives users in that group the ability to create, use, and delete queues that have their names and that are in the us-east-2 Region.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ListForConsole",
      "Effect": "Allow",
      "Action": ["SQSListQueues", "SQSGetQueueUrl"]
    }
  ]
}
```

```

        "Action": "sns:ListQueues",
        "Resource": "*"
    },
    {
        "Sid": "AllQueueActions",
        "Effect": "Allow",
        "Action": "sns:*",
        "Resource": "arn:aws:sns:us-east-2:*:${aws:username}-queue"
    }
]
}

```

To replace part of an ARN with a tag value, surround the prefix and key name with \${}. For example, the following Resource element refers to only a bucket that is named the same as the value in the requesting user's department tag.

```
"Resource": ["arn:aws:s3:::bucket/${aws:PrincipalTag/department}"]
```

Condition element

You can use a policy variable for Condition values in any condition that involves the string operators or the ARN operators. String operators include StringEquals, StringLike, and StringNotLike.. ARN operators include ArnEquals and ArnLike. You can't use a policy variable with other operators, such as Numeric, Date, Boolean, Binary, IP Address, or Null operators. For more information about condition operators, see [IAM JSON policy elements: Condition operators \(p. 644\)](#).

The following Amazon SNS topic policy gives users in AWS account 999999999999 the ability to manage (perform all actions for) the topic only if the URL matches their AWS user name.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Principal": {"AWS": "999999999999"},
            "Effect": "Allow",
            "Action": "sns:*",
            "Condition": {"StringLike": {"sns:endpoint": "https://example.com/${aws:username}/*"}}
        }
    ]
}
```

When referencing a tag in a Condition element expression, use the relevant prefix and key name as the condition key. Then use the value that you want to test in the condition value. For example, the following policy example allows full access to IAM resources, but only if the tag costCenter is attached to the resource. The tag must also have a value of either 12345 or 67890. If the tag has no value, or has any other value, then the request fails.

```
{
    "Version": "2015-01-01",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:*",
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "iam:ResourceTag/costCenter": [ "12345", "67890" ]
                }
            }
        }
    ]
}
```

}

Request information that you can use for policy variables

You can use the `Condition` element of a JSON policy to compare keys in the [request context \(p. 667\)](#) with key values that you specify in your policy. When you use a policy variable, AWS substitutes a value from the request context key in place of the variable in your policy.

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`** This can be used for conditions that check the date and time.
- **`aws:EpochTime`** This is the date in epoch or Unix time, for use with date/time conditions.
- **`aws:TokenIssueTime`** This is the date and time that temporary security credentials were issued and can be used with date/time conditions. **Note:** This key is only available in requests that are signed using temporary security credentials. For more information about temporary security credentials, see [Temporary security credentials in IAM \(p. 301\)](#).
- **`aws:PrincipalType`** This value indicates whether the principal is an account, user, federated, or assumed role—see the explanation that follows later.
- **`aws:SecureTransport`** This is a Boolean value that represents whether the request was sent using SSL.
- **`aws:SourceIp`** This is the requester's IP address, for use with IP address conditions. Refer to [IP address condition operators \(p. 648\)](#) for information about when `SourceIp` is valid and when you should use a VPC-specific key instead.
- **`aws:UserAgent`** This value is a string that contains information about the requester's client application. This string is generated by the client and can be unreliable. You can only use this context key from the AWS CLI.
- **`aws:userid`** This value is the unique ID for the current user—see the chart that follows.
- **`aws:username`** This is a string containing the [friendly name \(p. 600\)](#) of the current user—see the chart that follows.
- **`ec2:SourceInstanceARN`** This is the Amazon Resource Name (ARN) of the Amazon EC2 instance from which the request is made. This key is present only when the request comes from an Amazon EC2 instance using an IAM role associated with an EC2 instance profile.

Important

Key names are case-insensitive. For example, `aws:CurrentTime` is equivalent to `AWS:currenttime`.

Principal key values

The values for `aws:username`, `aws:userid`, and `aws:PrincipalType` depend on what type of principal initiated the request. For example, the request could be made using the credentials of an IAM user, an IAM role, or the AWS account root user. The following list shows values for these keys for different types of principals.

- **AWS account root user**
 - `aws:username`: (not present)
 - `aws:userid`: AWS account ID
 - `aws:PrincipalType`: Account
- **IAM user**

- aws:username: *IAM-user-name*
- aws:userid: *unique ID (p. 604)*
- aws:PrincipalType: User
- **Federated user**
 - aws:username: (not present)
 - aws:userid: *account:caller-specified-name*
 - aws:PrincipalType: FederatedUser
- **Web federated user and SAML federated user**

Note

For information about policy keys that are available when you use web identity federation, see [Identifying users with web identity federation \(p. 180\)](#).

- aws:username: (not present)
- aws:userid: (not present)
- aws:PrincipalType: AssumedRole
- **Assumed role**
 - aws:username: (not present)
 - aws:userid: *role-id:caller-specified-role-name*
 - aws:PrincipalType: Assumed role
- **Role assigned to Amazon EC2 instance**
 - aws:username: (not present)
 - aws:userid: *role-id:ec2-instance-id*
 - aws:PrincipalType: Assumed role
- **Anonymous caller** (Amazon SQS Amazon SNS and Amazon S3 only)
 - aws:username: (not present)
 - aws:userid: (not present)
 - aws:PrincipalType: Anonymous

For the items in this list, note the following:

- *not present* means that the value is not in the current request information, and any attempt to match it fails and causes the statement to be invalid.
- *role-id* is a unique identifier assigned to each role at creation. You can display the role ID with the AWS CLI command: `aws iam get-role --role-name rolename`
- *caller-specified-name* and *caller-specified-role-name* are names that are passed by the calling process (such as an application or service) when it makes a call to get temporary credentials.
- *ec2-instance-id* is a value assigned to the instance when it is launched and appears on the **Instances** page of the Amazon EC2 console. You can also display the instance ID by running the AWS CLI command: `aws ec2 describe-instances`

Information available in requests for federated users

Federated users are users who are authenticated using a system other than IAM. For example, a company might have an application for use in-house that makes calls to AWS. It might be impractical to give an IAM identity to every corporate user who uses the application. Instead, the company might use a proxy (middle-tier) application that has a single IAM identity, or the company might use a SAML identity provider (IdP). The proxy application or SAML IdP authenticates individual users using the corporate network. A proxy application can then use its IAM identity to get temporary security credentials for individual users. A SAML IdP can in effect exchange identity information for AWS temporary security credentials. The temporary credentials can then be used to access AWS resources.

Similarly, you might create an app for a mobile device in which the app needs to access AWS resources. In that case, you might use *web identity federation*, where the app authenticates the user using a well-known identity provider like Login with Amazon, Amazon Cognito, Facebook, or Google. The app can then use the user's authentication information from these providers to get temporary security credentials for accessing AWS resources.

The recommended way to use web identity federation is by taking advantage of Amazon Cognito and the AWS mobile SDKs. For more information, see the following:

- [Amazon Cognito Overview](#) in the *AWS Mobile SDK for Android Developer Guide*
- [Amazon Cognito Overview](#) in the *AWS Mobile SDK for iOS Developer Guide*
- [Common scenarios for temporary credentials \(p. 302\)](#).

Service-specific information

Requests can also include service-specific keys and values in its request context. Examples include the following:

- `s3:prefix`
- `s3:max-keys`
- `s3:x-amz-acl`
- `sns:Endpoint`
- `sns:Protocol`

For information about service-specific keys that you can use to get values for policy variables, refer to the documentation for the individual services. For example, see the following topics:

- [Bucket Keys in Amazon S3 Policies](#) in the *Amazon Simple Storage Service Developer Guide*.
- [Amazon SNS Keys](#) in the *Amazon Simple Notification Service Developer Guide*.

Special characters

There are a few special predefined policy variables that have fixed values that enable you to represent characters that otherwise have special meaning. If these special characters are part of the string, you are trying to match and you inserted them literally they would be misinterpreted. For example, inserting an * asterisk in the string would be interpreted as a wildcard, matching any characters, instead of as a literal *. In these cases, you can use the following predefined policy variables:

- `${*}` - use where you need an * asterisk character.
- `${?}` - use where you need a ? question mark character.
- `${$}` - use where you need a \$ dollar sign character.

These predefined policy variables can be used in any string where you can use regular policy variables.

For more information

For more information about policies, see the following:

- [Policies and permissions in IAM \(p. 351\)](#)
- [Example IAM identity-based policies \(p. 389\)](#)
- [IAM JSON policy elements reference \(p. 628\)](#)
- [Policy evaluation logic \(p. 666\)](#)

- About web identity federation (p. 177)

IAM JSON policy elements: Supported data types

This section lists the data types that are supported when you specify values in JSON policies. The policy language doesn't support all types for each policy element; for information about each element, see the preceding sections.

- Strings
- Numbers (Ints and Floats)
- Boolean
- Null
- Lists
- Maps
- Structs (which are just nested Maps)

The following table maps each data type to the serialization. Note that all policies must be in UTF-8. For information about the JSON data types, go to [RFC 4627](#).

Type	JSON
String	String
Integer	Number
Float	Number
Boolean	true false
Null	null
Date	String adhering to the W3C Profile of ISO 8601
IpAddress	String adhering to RFC 4632
List	Array
Object	Object

Policy evaluation logic

When a principal tries to use the AWS Management Console, the AWS API, or the AWS CLI, that principal sends a *request* to AWS. When an AWS service receives the request, AWS completes several steps to determine whether to allow or deny the request.

1. **Authentication** – AWS first authenticates the principal that makes the request, if necessary. This step is not necessary for a few services, such as Amazon S3, that allow some requests from anonymous users.
2. **Processing the request context (p. 667)** – AWS processes the information gathered in the request to determine which policies apply to the request.
3. **Evaluating policies within a single account (p. 667)** – AWS evaluates all of the policy types, which affect the order in which the policies are evaluated.

4. **Determining whether a request is allowed or denied within an account (p. 669)** – AWS then processes the policies against the request context to determine whether the request is allowed or denied.

Processing the request context

AWS processes the request to gather the following information into a *request context*:

- **Actions (or operations)** – The actions or operations that the principal wants to perform.
- **Resources** – The AWS resource object upon which the actions or operations are performed.
- **Principal** – The user, role, federated user, or application that sent the request. Information about the principal includes the policies that are associated with that principal.
- **Environment data** – Information about the IP address, user agent, SSL enabled status, or the time of day.
- **Resource data** – Data related to the resource that is being requested. This can include information such as a DynamoDB table name or a tag on an Amazon EC2 instance.

AWS then uses this information to find policies that apply to the request context.

Evaluating policies within a single account

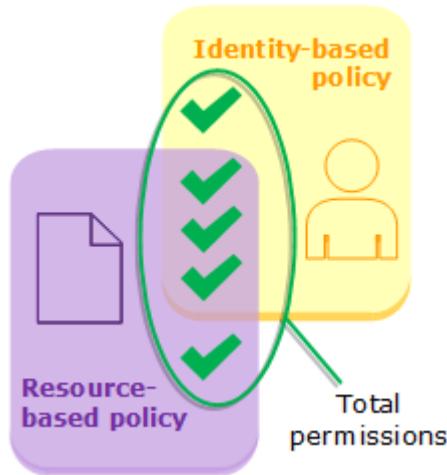
How AWS evaluates policies depends on the types of policies that apply to the request context. The following policy types, listed in order of frequency, are available for use within a single AWS account. For more information about these policy types, see [Policies and permissions in IAM \(p. 351\)](#). To learn how AWS evaluates policies for cross-account access, see [Cross-account policy evaluation logic \(p. 674\)](#).

1. **Identity-based policies** – Identity-based policies are attached to an IAM identity (user, group of users, or role) and grant permissions to IAM entities (users and roles). If only identity-based policies apply to a request, then AWS checks all of those policies for at least one Allow.
2. **Resource-based policies** – Resource-based policies grant permissions to the principal (account, user, role, or federated user) specified as the principal. The permissions define what the principal can do with the resource to which the policy is attached. If resource-based policies and identity-based policies both apply to a request, then AWS checks all the policies for at least one Allow.
3. **IAM permissions boundaries** – Permissions boundaries are an advanced feature that sets the maximum permissions that an identity-based policy can grant to an IAM entity (user or role). When you set a permissions boundary for an entity, the entity can perform only the actions that are allowed by both its identity-based policies and its permissions boundaries. An implicit deny in a permissions boundary does not limit the permissions granted by a resource-based policy.
4. **AWS Organizations service control policies (SCPs)** – Organizations SCPs specify the maximum permissions for an organization or organizational unit (OU). The SCP maximum applies to principals in member accounts, including each AWS account root user. If an SCP is present, identity-based and resource-based policies grant permissions to principals in member accounts only if those policies and the SCP allow the action. If both a permissions boundary and an SCP are present, then the boundary, the SCP, and the identity-based policy must all allow the action.
5. **Session policies** – Session policies are advanced policies that you pass as parameters when you programmatically create a temporary session for a role or federated user. To create a role session programmatically, use one of the `AssumeRole*` API operations. When you do this and pass session policies, the resulting session's permissions are the intersection of the IAM entity's identity-based policy and the session policies. To create a federated user session, you use an IAM user's access keys to programmatically call the `GetFederationToken` API operation. A resource-based policy has a different effect on the evaluation of session policy permissions. The difference depends on whether the user or role's ARN or the session's ARN is listed as the principal in the resource-based policy. For more information, see [Session policies \(p. 353\)](#).

Remember, an explicit deny in any of these policies overrides the allow.

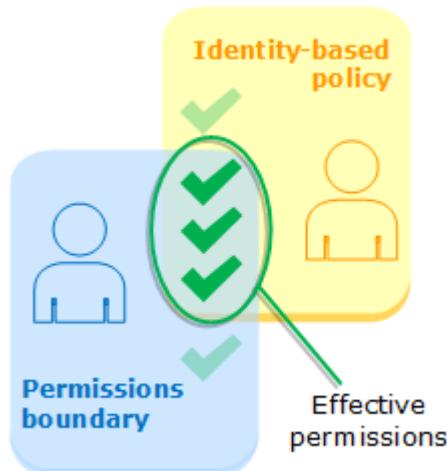
Evaluating identity-based policies with resource-based policies

Identity-based policies and resource-based policies grant permissions to the identities or resources to which they are attached. When an IAM entity (user or role) requests access to a resource within the same account, AWS evaluates all the permissions granted by the identity-based and resource-based policies. The resulting permissions are the total permissions of the two types. If an action is allowed by an identity-based policy, a resource-based policy, or both, then AWS allows the action. An explicit deny in either of these policies overrides the allow.



Evaluating identity-based policies with permissions boundaries

When AWS evaluates the identity-based policies and permissions boundary for a user, the resulting permissions are the intersection of the two categories. That means that when you add a permissions boundary to a user with existing identity-based policies, you might reduce the actions that the user can perform. Alternatively, when you remove a permissions boundary from a user, you might increase the actions they can perform. An explicit deny in either of these policies overrides the allow. To view information about how other policy types are evaluated with permissions boundaries, see [Evaluating effective permissions with boundaries \(p. 366\)](#).



Evaluating identity-based policies with Organizations SCPs

When a user belongs to an account that is a member of an organization, the resulting permissions are the intersection of the user's policies and the SCP. This means that an action must be allowed by both the identity-based policy and the SCP. An explicit deny in either of these policies overrides the allow.



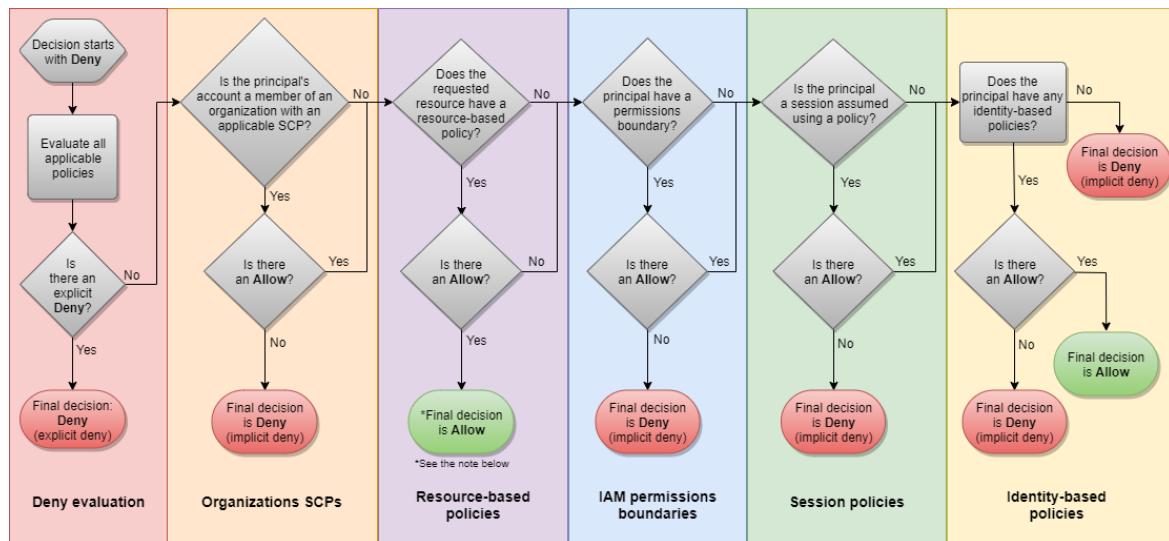
You can learn [whether your account is a member of an organization](#) in AWS Organizations. Organization members might be affected by an SCP. To view this data using the AWS CLI command or AWS API operation, you must have permissions for the `organizations:DescribeOrganization` action for your Organizations entity. You must have additional permissions to perform the operation in the Organizations console. To learn whether an SCP is denying access to a specific request, or to change your effective permissions, contact your AWS Organizations administrator.

Determining whether a request is allowed or denied within an account

Assume that a principal sends a request to AWS to access a resource in the same account as the principal's entity. The AWS enforcement code decides whether the request should be allowed or denied. AWS gathers all of the policies that apply to the request context. The following is a high-level summary of the AWS evaluation logic on those policies within a single account.

- By default, all requests are implicitly denied. (Alternatively, by default, the AWS account root user has full access.)
- An explicit allow in an identity-based or resource-based policy overrides this default.
- If a permissions boundary, Organizations SCP, or session policy is present, it might override the allow with an implicit deny.
- An explicit deny in any policy overrides any allows.

The following flow chart provides details about how the decision is made.



- Deny evaluation** – By default, all requests are denied. This is called an [implicit deny \(p. 673\)](#). The AWS enforcement code evaluates all policies within the account that apply to the request. These include AWS Organizations service control policies (SCPs), resource-based policies, IAM permissions boundaries, role session policies, and identity-based policies. In all those policies, the enforcement code looks for a Deny statement that applies to the request. This is called an [explicit deny \(p. 673\)](#). If the code finds even one explicit deny that applies, the code returns a final decision of **Deny**. If there is no explicit deny, the code continues.
- Organizations SCPs** – Then the code evaluates AWS Organizations service control policies (SCPs) that apply to the request. SCPs apply to principals of the account where the SCPs are attached. If the enforcement code does not find any applicable Allow statements in the SCPs, then the request is implicitly denied. The code returns a final decision of **Deny**. If there is no SCP, or if the SCP allows the requested action, the code continues.
- Resource-based policies** – If the requested resource has a resource-based policy that allows the principal to perform the requested action, then the code returns a final decision of **Allow**. If there is no resource-based policy, or if the policy does not include an Allow statement, then the code continues.

Note

This logic can behave differently if you specify the ARN of an IAM role or user as the principal of the resource-based policy. Someone can use session policies to create a temporary credential session for that role or federated user. In that case, the effective permissions for the session might not exceed those allowed by the identity-based policy of the user or role. For more information, see [Session Policies](#).

- IAM permissions boundaries** – The enforcement code then checks whether the IAM entity that is used by the principal has a permissions boundary. If the policy that is used to set the permissions boundary does not allow the requested action, then the request is implicitly denied. The code returns a final decision of **Deny**. If there is no permissions boundary, or if the permissions boundary allows the requested action, the code continues.
- Session policies** – The code then checks whether the principal is using a session that was assumed by passing a session policy. You can pass a session policy while using the AWS CLI or AWS API to get temporary credentials for a role or federated user. If the session policy is present and does not allow the requested action, then the request is implicitly denied. The code returns a final decision of **Deny**. If there is no session policy, or if the policy allows the requested action, the code continues.
- Identity-based policies** – The code then checks the identity-based policies for the principal. For an IAM user, these include user policies and policies from groups to which the user belongs. If any statement in any applicable identity-based policies allows the requested action, then the enforcement

code returns a final decision of **Allow**. If there are no statements that allow the requested action, then the request is implicitly denied, and the code returns a final decision of **Deny**.

7. **Errors** – If the AWS enforcement code encounters an error at any point during the evaluation, then it generates an exception and closes.

Example identity-based and resource-based policy evaluation

The most common types of policies are identity-based policies and resource-based policies.

Assume that Carlos has the user name `carlossalazar` and he tries to save a file to the `carlossalazar-logs` Amazon S3 bucket.

Also assume that the following policy is attached to the `carlossalazar` IAM user.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowS3ListRead",
            "Effect": "Allow",
            "Action": [
                "s3>ListAllMyBuckets",
                "s3:HeadBucket"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowS3Self",
            "Effect": "Allow",
            "Action": "s3:*",
            "Resource": [
                "arn:aws:s3::::carlossalazar/*",
                "arn:aws:s3::::carlossalazar"
            ]
        },
        {
            "Sid": "DenyS3Logs",
            "Effect": "Deny",
            "Action": "s3:*",
            "Resource": [
                "arn:aws:s3::::log*",
                "arn:aws:s3::::log*/"
            ]
        }
    ]
}
```

The `AllowS3ListRead` statement in this policy allows Carlos to view a list of all of the buckets in the account. The `AllowS3Self` statement allows Carlos full access to the bucket with the same name as his user name. The `DenyS3Logs` statement denies Carlos access to any S3 bucket with `log` in its name.

Additionally, the following resource-based policy (called a bucket policy) is attached to the `carlossalazar` bucket.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:*",
            "Principal": { "AWS": "arn:aws:iam::111122223333:user/carlossalazar" },
            "Resource": "arn:aws:s3:::carlossalazar"
        }
    ]
}
```

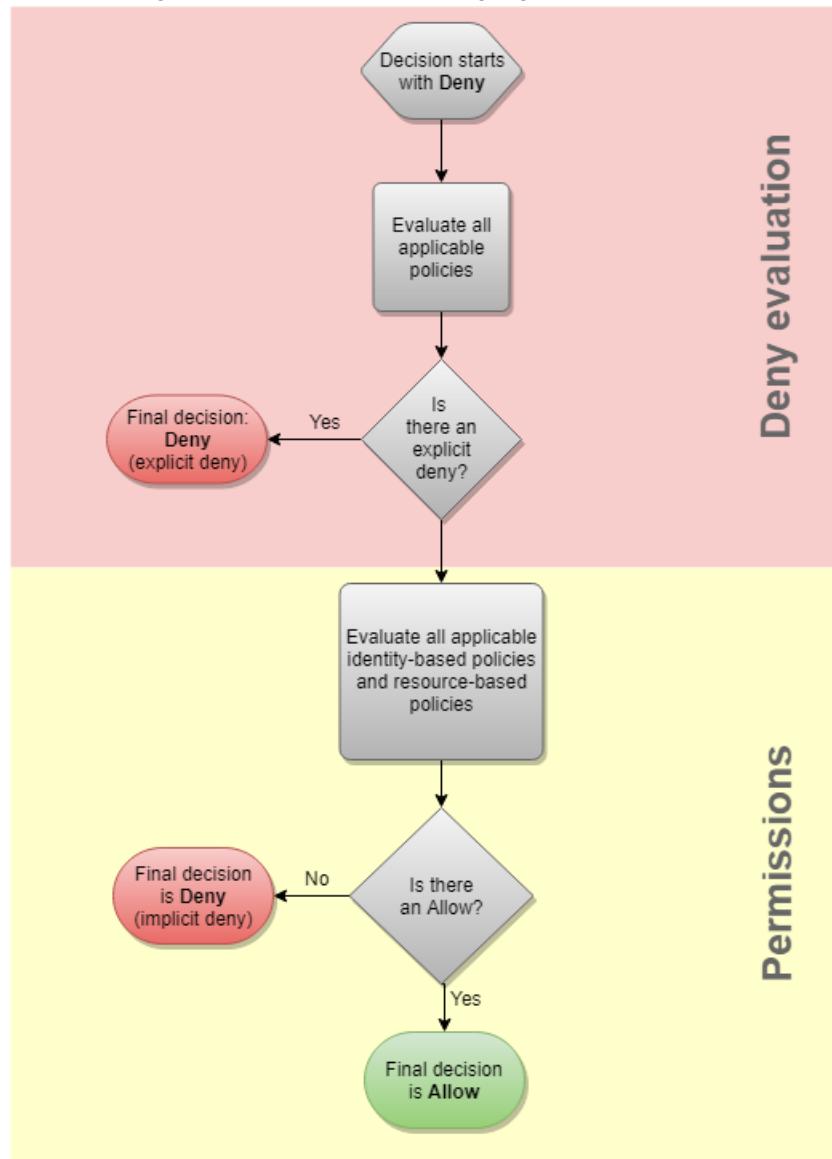
```

        "Resource": "*"
    ]
}

```

This policy specifies that only the `carlossalazar` user can access the `carlossalazar` bucket.

When Carlos makes his request to save a file to the `carlossalazar-logs` bucket, AWS determines what policies apply to the request. In this case, only the identity-based policy and the resource-based policy apply. These are both permissions policies. Because no permissions boundaries apply, the evaluation logic is reduced to the following logic.



AWS first checks for a Deny statement that applies to the context of the request. It finds one, because the identity-based policy explicitly denies Carlos access to any S3 buckets used for logging. Carlos is denied access.

Assume that he then realizes his mistake and tries to save the file to the `carlossalazar` bucket. AWS checks for a Deny statement and does not find one. It then checks the permissions policies. Both

the identity-based policy and the resource-based policy allow the request. Therefore, AWS allows the request. If either of them explicitly denied the statement, the request would have been denied. If one of the policy types allows the request and the other doesn't, the request is still allowed.

The difference between explicit and implicit denies

A request results in an explicit deny if an applicable policy includes a Deny statement. If policies that apply to a request include an Allow statement and a Deny statement, the Deny statement trumps the Allow statement. The request is explicitly denied.

An implicit denial occurs when there is no applicable Deny statement but also no applicable Allow statement. Because an IAM user, role, or federated user is denied access by default, they must be explicitly allowed to perform an action. Otherwise, they are implicitly denied access.

When you design your authorization strategy, you must create policies with Allow statements to allow your principals to successfully make requests. However, you can choose any combination of explicit and implicit denies. For example, you can create the following policy to allow an administrator full access to all resources in AWS, but explicitly deny access to billing. If someone adds another policy to this administrator granting them access to billing, it is still denied because of this explicit deny.

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

Alternatively, you can create the following policy to allow a user to manage users, but not groups or any other resources in IAM. Those actions are implicitly denied, as are actions in other services. However, if someone adds a policy to the user that allows them to perform these other actions, then they are allowed.

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "iam:AttachUserPolicy",
            "iam>CreateUser",
            "iam>DeleteUser",
            "iam>DeleteUserPolicy",
            "iam:DetachUserPolicy",
            "iam:GetUser",
            "iam:GetUserPolicy",
            "iam>ListAttachedUserPolicies",
            "iam>ListUserPolicies",
            "iam>ListUsers",
            "iam:PutUserPolicy",
            "iam:UpdateUser"
        ],
        "Resource": "*"
    }
}
```

```
}
```

Cross-account policy evaluation logic

You can allow a principal in one account to access resources in a second account. This is called cross-account access. When you allow cross-account access, the account where the principal exists is called the *trusted* account. The account where the resource exists is the *trusting* account.

To allow cross-account access, you attach a resource-based policy to the resource that you want to share. You must also attach an identity-based policy to the identity that acts the principal in the request. The resource-based policy in the trusting account must specify the principal of the trusted account that will have access to the resource. You can specify the entire account or its IAM users, federated users, IAM roles, or assumed-role sessions. You can also specify an AWS service as a principal. For more information, see [Specifying a principal \(p. 631\)](#).

The principal's identity-based policy must allow the requested access to the resource in the trusting service. You can do this by specifying the ARN of the resource or by allowing access to all resources (*).

In IAM, you can attach a resource-based policy to an IAM role to allow principals in other accounts to assume that role. The role's resource-based policy is called a role trust policy. After assuming that role, the allowed principals can use the resulting temporary credentials to access multiple resources in your account. This access is defined in the role's identity-based permissions policy. To learn how allowing cross-account access using roles is different from allowing cross-account access using other resource-based policies, see [How IAM roles differ from resource-based policies \(p. 286\)](#).

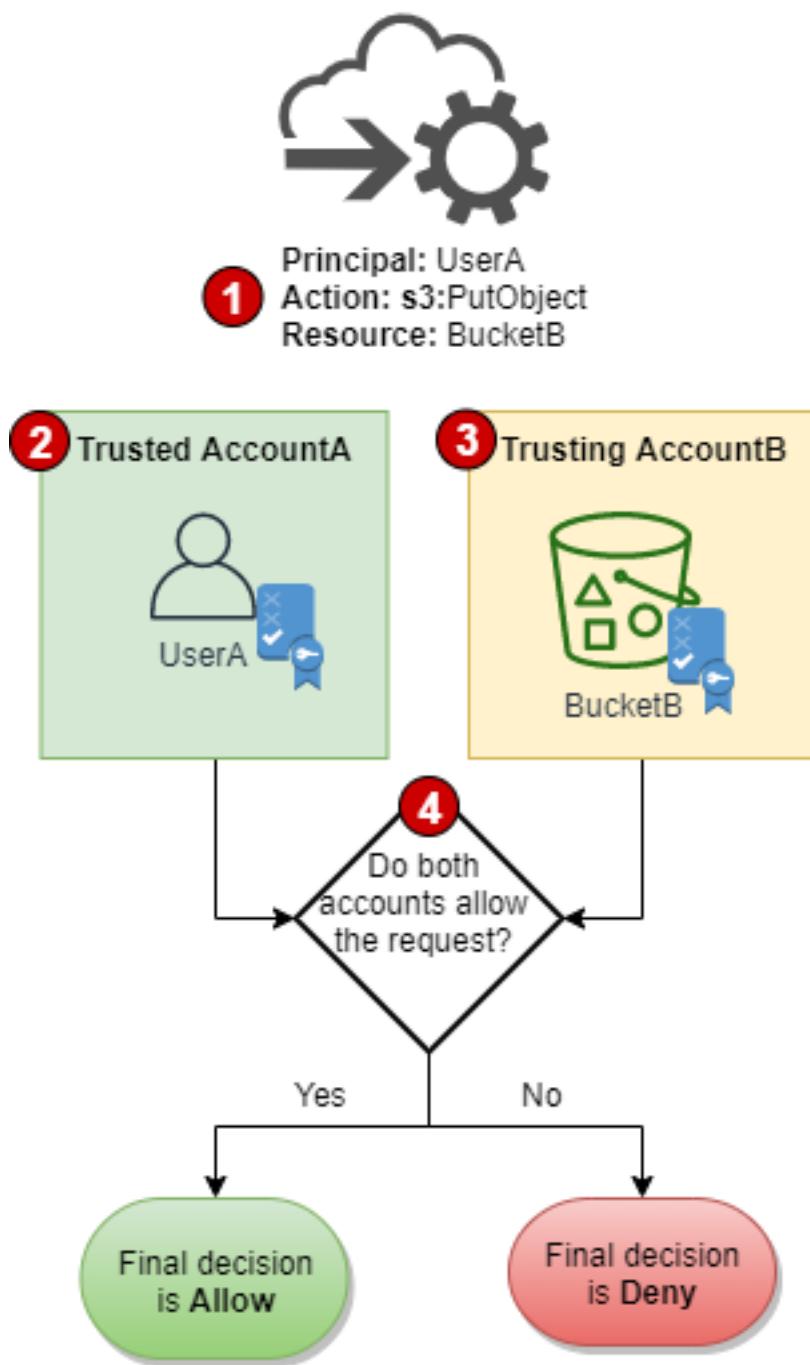
Important

Other services can affect the policy evaluation logic. For example, AWS Organizations supports [service control policies](#) that can be applied to principals one or more accounts. AWS Resource Access Manager supports [policy fragments](#) that control which actions that principals are allowed to perform on resources that are shared with them.

Determining whether a cross-account request is allowed

For cross-account requests, the requester in the trusted AccountA must have an identity-based policy. That policy must allow them to make a request to the resource in the trusting AccountB. Additionally, the resource-based policy in AccountB must allow the requester in AccountA to access the resource.

When you make a cross-account request, AWS performs two evaluations. AWS evaluates the request in the trusting account and the trusted account. For more information about how a request is evaluated within a single account, see [Determining whether a request is allowed or denied within an account \(p. 669\)](#). The request is allowed only if both evaluations return a decision of `Allow`.



1. When a principal in one account makes a request to access a resource in another account, this is a cross-account request.
2. The requesting principal exists in the trusted account (AccountA). When AWS evaluates this account, it checks the identity-based policy and any policies that can limit an identity-based policy. For more information, see [Evaluating policies within a single account \(p. 667\)](#).
3. The requested resource exists in the trusting account (AccountB). When AWS evaluates this account, it checks the resource-based policy that is attached to the requested resource and any policies that can limit a resource-based policy. For more information, see [Evaluating policies within a single account \(p. 667\)](#).

4. AWS allows the request only if both account policy evaluations allow the request.

Example cross-account policy evaluation

The following example demonstrates a scenario where a user in one account is granted permissions by a resource-based policy in a second account.

Assume that Carlos is a developer with an IAM user named `carlossalazar` in account 111111111111. He wants to save a file to the `Production-logs` Amazon S3 bucket in account 222222222222.

Also assume that the following policy is attached to the `carlossalazar` IAM user.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowS3ListRead",
            "Effect": "Allow",
            "Action": "s3>ListAllMyBuckets",
            "Resource": "*"
        },
        {
            "Sid": "AllowS3ProductionObjectActions",
            "Effect": "Allow",
            "Action": "s3:*Object*",
            "Resource": "arn:aws:s3:::Production/*"
        },
        {
            "Sid": "DenyS3Logs",
            "Effect": "Deny",
            "Action": "s3:*",
            "Resource": [
                "arn:aws:s3:::*log*",
                "arn:aws:s3:::*log*/"
            ]
        }
    ]
}
```

The `AllowS3ListRead` statement in this policy allows Carlos to view a list of all of the buckets in Amazon S3. The `AllowS3ProductionObjectActions` statement allows Carlos full access to objects in the `Production` bucket. The `DenyS3Logs` statement denies Carlos access to any S3 bucket with `log` in its name. It also denies access to all objects in those buckets.

Additionally, the following resource-based policy (called a bucket policy) is attached to the `Production` bucket in account 222222222222.

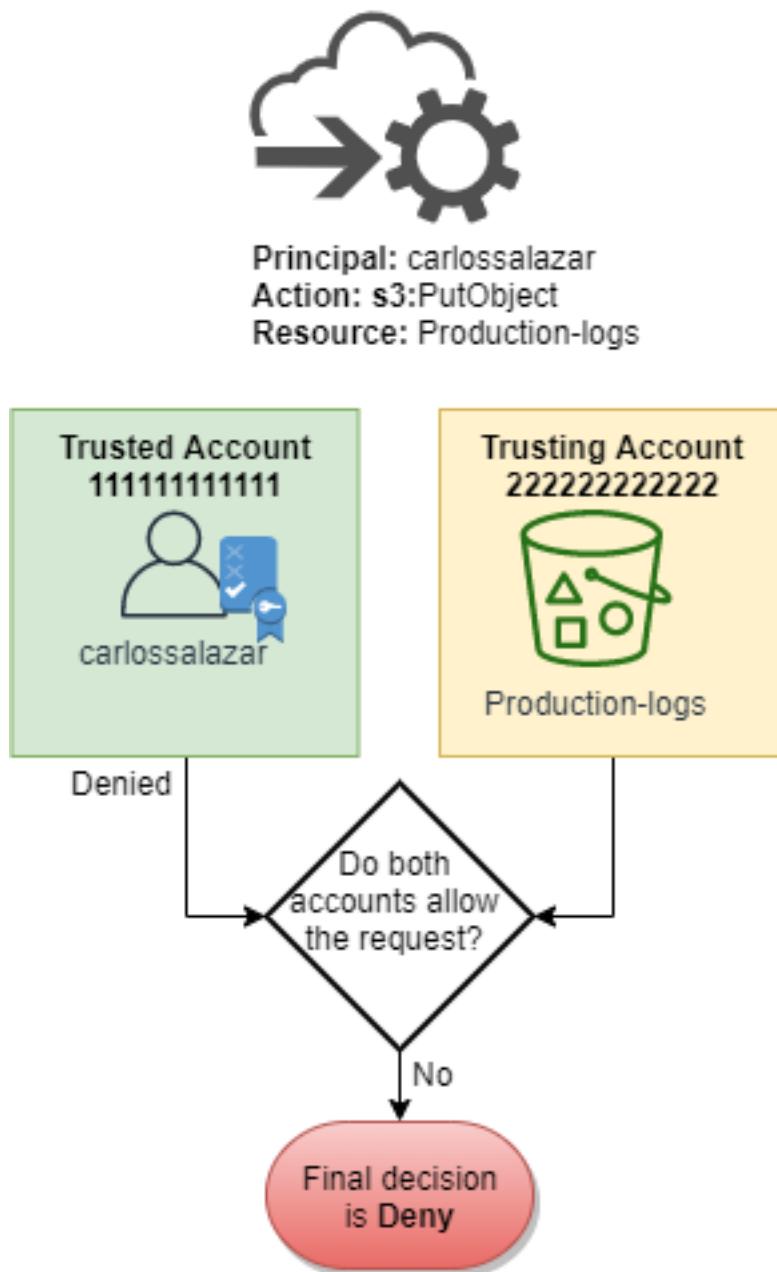
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject*",
                "s3:PutObject*",
                "s3:ReplicateObject",
                "s3:RestoreObject"
            ],
            "Principal": { "AWS": "arn:aws:iam::111111111111:user/carlossalazar" },
            "Resource": "arn:aws:s3:::Production/*"
        }
    ]
}
```

```
    ]  
}
```

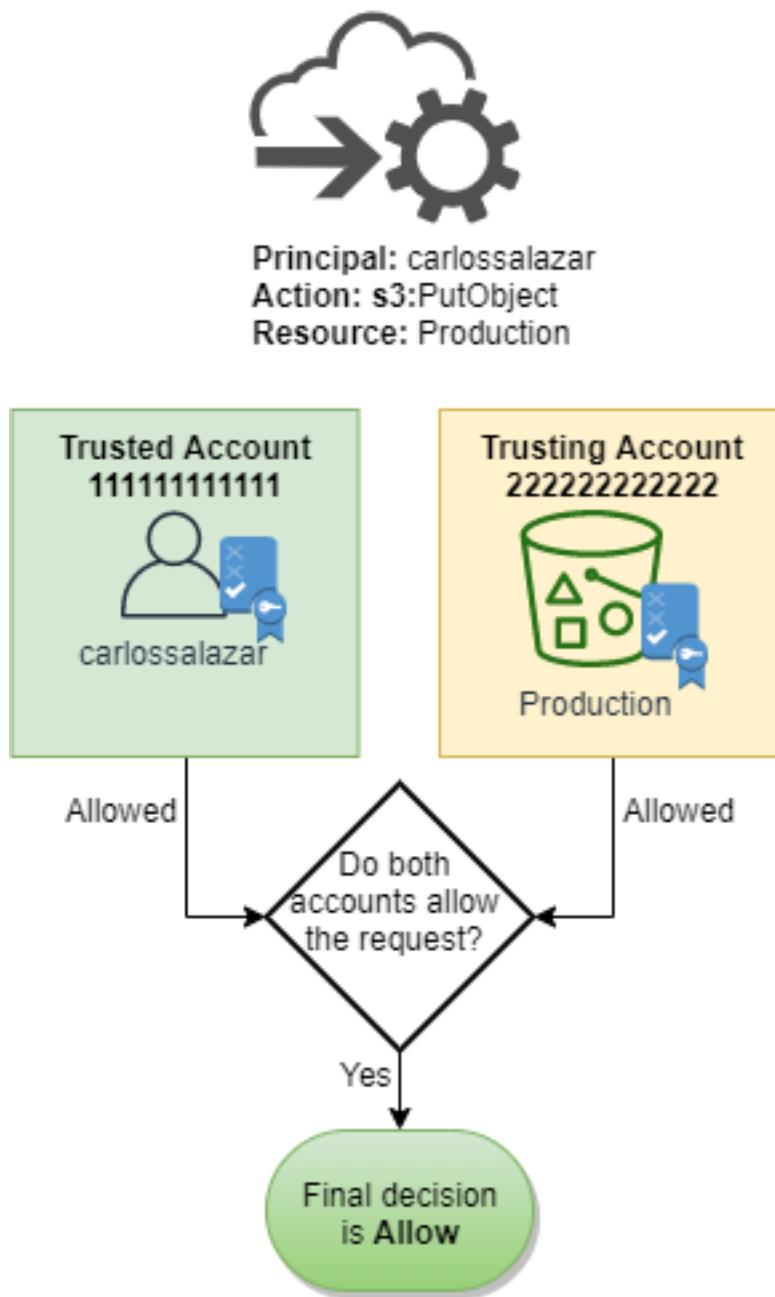
This policy allows the `carlossalazar` user to access objects in the `Production` bucket. He can create and edit, but not delete the objects in the bucket. He can't manage the bucket itself.

When Carlos makes his request to save a file to the `Production-logs` bucket, AWS determines what policies apply to the request. In this case, the identity-based policy attached to the `carlossalazar` user is the only policy that applies in account `111111111111`. In account `222222222222`, there is no resource-based policy attached to the `Production-logs` bucket. When AWS evaluates account `111111111111`, it returns a decision of `Deny`. This is because the `DenyS3Logs` statement in the identity-based policy explicitly denies access to any log buckets. For more information about how a request is evaluated within a single account, see [Determining whether a request is allowed or denied within an account \(p. 669\)](#).

Because the request is explicitly denied within one of the accounts, the final decision is to deny the request.



Assume that Carlos then realizes his mistake and tries to save the file to the Production bucket. AWS first checks account 111111111111 to determine if the request is allowed. Only the identity-based policy applies, and it allows the request. AWS then checks account 222222222222. Only the resource-based policy attached to the Production bucket applies, and it allows the request. Because both accounts allow the request, the final decision is to allow the request.



Grammar of the IAM JSON policy language

This page presents a formal grammar for the language used to create JSON policies in IAM. We present this grammar so that you can understand how to construct and validate policies.

For examples of policies, see the following topics:

- [Policies and permissions in IAM \(p. 351\)](#)
- [Example IAM identity-based policies \(p. 389\)](#)
- [Example Policies for Working in the Amazon EC2 Console](#) and [Example Policies for Working With the AWS CLI, the Amazon EC2 CLI, or an AWS SDK](#) in the [Amazon EC2 User Guide for Linux Instances](#).

- [Bucket Policy Examples](#) and [User Policy Examples](#) in the *Amazon Simple Storage Service Developer Guide*.

For examples of policies used in other AWS services, go to the documentation for those services.

Topics

- [The policy language and JSON \(p. 680\)](#)
- [Conventions used in this grammar \(p. 680\)](#)
- [Grammar \(p. 681\)](#)
- [Policy grammar notes \(p. 682\)](#)

The policy language and JSON

Policies are expressed in JSON. When a policy is submitted to IAM, it is first validated to make sure that the JSON syntax is correct. In this document, we do not provide a complete description of what constitutes valid JSON. However, here are some basic JSON rules:

- White space between individual entities is allowed.
- Values are enclosed in quotation marks. Quotation marks are optional for numeric and Boolean values.
- Many elements (for example, `action_string_list` and `resource_string_list`) can take a JSON array as a value. Arrays can take one or more values. If more than one value is included, the array is in square brackets ([and]) and comma-delimited, as in the following example:

```
"Action" : [ "ec2:Describe*", "ec2>List*" ]
```

- Basic JSON data types (Boolean, number, and string) are defined in [RFC 7159](#).

You can use a JSON validator to check the syntax of a policy. You can find a validator online, and many code editors and XML-editing tools include JSON validation features.

Conventions used in this grammar

The following conventions are used in this grammar:

- The following characters are JSON tokens and *are* included in policies:
`{ } [] " , :`
- The following characters are special characters in the grammar and are *not* included in policies:
`= < > () |`
- If an element allows multiple values, it is indicated using repeated values, a comma delimiter, and an ellipsis (...). Examples:

```
[<action_string>, <action_string>, ...]
```

```
<principal_map> = { <principal_map_entry>, <principal_map_entry>, ... }
```

If multiple values are allowed, it is also valid to include only one value. For only one value, the trailing comma must be omitted. If the element takes an array (marked with [and]) but only one value is included, the brackets are optional. Examples:

```
"Action": [<action_string>]
```

```
"Action": <action_string>
```

- A question mark (?) following an element indicates that the element is optional. Example:

```
<version_block?>
```

However, be sure to refer to the notes that follow the grammar listing for details about optional elements.

- A vertical line (|) between elements indicates alternatives. In the grammar, parentheses define the scope of the alternatives. Example:

```
("Principal" | "NotPrincipal")
```

- Elements that must be literal strings are enclosed in double quotation marks ("). Example:

```
<version_block> = "Version" : ("2008-10-17" | "2012-10-17")
```

For additional notes, see [Policy grammar notes \(p. 682\)](#) following the grammar listing.

Grammar

The following listing describes the policy language grammar. For conventions used in the listing, see the preceding section. For additional information, see the notes that follow.

Note

This grammar describes policies marked with a version of 2008-10-17 and 2012-10-17. A Version policy element is different from a policy version. The Version policy element is used within a policy and defines the version of the policy language. A policy version, on the other hand, is created when you make changes to a customer managed policy in IAM. The changed policy doesn't overwrite the existing policy. Instead, IAM creates a new version of the managed policy. To learn more about the Version policy element see [IAM JSON policy elements: Version \(p. 629\)](#). To learn more about policy versions, see the section called "Versioning IAM policies" (p. 462).

```
policy  = {
    <version_block?>
    <id_block?>
    <statement_block>
}

<version_block> = "Version" : ("2008-10-17" | "2012-10-17")

<id_block> = "Id" : <policy_id_string>

<statement_block> = "Statement" : [ <statement>, <statement>, ... ]

<statement> = {
    <sid_block?>,
    <principal_block?>,
    <effect_block>,
    <action_block>,
    <resource_block>,
    <condition_block?>
}

<sid_block> = "Sid" : <sid_string>

<effect_block> = "Effect" : ("Allow" | "Deny")

<principal_block> = ("Principal" | "NotPrincipal") : ("*" | <principal_map>)

<principal_map> = { <principal_map_entry>, <principal_map_entry>, ... }

<principal_map_entry> = ("AWS" | "Federated" | "Service" | "CanonicalUser") :
    [<principal_id_string>, <principal_id_string>, ...]
```

```
<action_block> = ("Action" | "NotAction") :  
    ("*" | [<action_string>, <action_string>, ...])  
  
<resource_block> = ("Resource" | "NotResource") :  
    ("*" | [<resource_string>, <resource_string>, ...])  
  
<condition_block> = "Condition" : { <condition_map> }  
<condition_map> = {  
    <condition_type_string> : { <condition_key_string> : <condition_value_list> },  
    <condition_type_string> : { <condition_key_string> : <condition_value_list> }, ...  
}  
<condition_value_list> = [<condition_value>, <condition_value>, ...]  
<condition_value> = ("string" | "number" | "Boolean")
```

Policy grammar notes

- A single policy can contain an array of statements.
- Policies have a maximum size between 2048 characters and 10,240 characters, depending on what entity the policy is attached to. For more information, see [IAM and STS quotas \(p. 606\)](#). Policy size calculations do not include white space characters.
- Individual elements must not contain multiple instances of the same key. For example, you cannot include the `Effect` block twice in the same statement.
- Blocks can appear in any order. For example, `version_block` can follow `id_block` in a policy. Similarly, `effect_block`, `principal_block`, `action_block` can appear in any order within a statement.
- The `id_block` is optional in resource-based policies. It must *not* be included in identity-based policies.
- The `principal_block` element is required in resource-based policies (for example, in Amazon S3 bucket policies) and in trust policies for IAM roles. It must *not* be included in identity-based policies.
- The `principal_map` element in Amazon S3 bucket policies can include the CanonicalUser ID. Most resource-based policies do not support this mapping. To learn more about using the canonical user ID in a bucket policy, see [Specifying a Principal in a Policy](#) in the *Amazon Simple Storage Service Developer Guide*.
- Each string value (`policy_id_string`, `sid_string`, `principal_id_string`, `action_string`, `resource_string`, `condition_type_string`, `condition_key_string`, and the string version of `condition_value`) can have its own minimum and maximum length restrictions, specific allowed values, or required internal format.

Notes about string values

This section provides additional information about string values that are used in different elements in a policy.

`action_string`

Consists of a service namespace, a colon, and the name of an action. Action names can include wildcards. Examples:

```
"Action": "ec2:StartInstances"  
  
"Action": [  
    "ec2:StartInstances",  
    "ec2:StopInstances"  
]  
  
"Action": "cloudformation:*
```

```
"Action": "*"  
  
"Action": [  
    "s3:Get*",  
    "s3>List*"  
]
```

policy_id_string

Provides a way to include information about the policy as a whole. Some services, such as Amazon SQS and Amazon SNS, use the `Id` element in reserved ways. Unless otherwise restricted by an individual service, `policy_id_string` can include spaces. Some services require this value to be unique within an AWS account.

Note

The `id_block` is allowed in resource-based policies, but not in identity-based policies.

There is no limit to the length, although this string contributes to the overall length of the policy, which is limited.

```
"Id": "Admin_Policy"  
  
"Id": "cd3ad3d9-2776-4ef1-a904-4c229d1642ee"
```

sid_string

Provides a way to include information about an individual statement. For IAM policies, basic alphanumeric characters (A-Z,a-z,0-9) are the only allowed characters in the `Sid` value. Other AWS services that support resource policies may have other requirements for the `Sid` value. For example, some services require this value to be unique within an AWS account, and some services allow additional characters such as spaces in the `Sid` value.

```
"Sid": "1"  
  
"Sid": "ThisStatementProvidesPermissionsForConsoleAccess"
```

principal_id_string

Provides a way to specify a principal using the [Amazon Resource Name \(ARN\)](#) (p. 601) of the AWS account, IAM user, IAM role, federated user, or assumed-role user. For an AWS account, you can also use the short form `AWS:accountnumber` instead of the full ARN. For all of the options including AWS services, assumed roles, and so on, see [Specifying a principal \(p. 631\)](#).

Note that you can use `*` only to specify "everyone/anonymous." You cannot use it to specify part of a name or ARN.

resource_string

In most cases, consists of an [Amazon Resource Name \(p. 601\)](#) (ARN).

```
"Resource": "arn:aws:iam::123456789012:user/Bob"  
  
"Resource": "arn:aws:s3:::examplebucket/*"
```

condition_type_string

Identifies the type of condition being tested, such as `StringEquals`, `StringLike`, `NumericLessThan`, `DateGreaterThanOrEqual`, `Bool`, `BinaryEquals`, `IpAddress`, `ArnEquals`, etc. For a complete list of condition types, see [IAM JSON policy elements: Condition operators \(p. 644\)](#).

```
"Condition": {  
    "NumericLessThanEquals": {  
        "s3:max-keys": "10"  
    }  
}  
  
"Condition": {  
    "Bool": {  
        "aws:SecureTransport": "true"  
    }  
}  
  
"Condition": {  
    "StringEquals": {  
        "s3:x-amz-server-side-encryption": "AES256"  
    }  
}
```

condition_key_string

Identifies the condition key whose value will be tested to determine whether the condition is met. AWS defines a set of condition keys that are available in all AWS services, including `aws:PrincipalType`, `aws:SecureTransport`, and `aws:userid`.

For a list of AWS condition keys, see [AWS global condition context keys \(p. 692\)](#). For condition keys that are specific to a service, see the documentation for that service such as the following:

- [Specifying Conditions in a Policy](#) in the *Amazon Simple Storage Service Developer Guide*
- [IAM Policies for Amazon EC2](#) in the *Amazon EC2 User Guide for Linux Instances*.

```
"Condition":{  
    "Bool": {  
        "aws:SecureTransport": "true"  
    }  
}  
  
"Condition": {  
    "StringNotEquals": {  
        "s3:x-amz-server-side-encryption": "AES256"  
    }  
}  
  
"Condition": {  
    "StringEquals": {  
        "ec2:ResourceTag/purpose": "test"  
    }  
}
```

AWS managed policies for job functions

AWS managed policies for job functions are designed to closely align to common job functions in the IT industry. You can use these policies to easily grant the permissions needed to carry out the tasks expected of someone in a specific job function. These policies consolidate permissions for many services into a single policy that's easier to work with than having permissions scattered across many policies.

You can attach these policies for job functions to any group, user, or role.

Use Roles to Combine Services

Some of the policies use IAM service roles to help you take advantage of features found in other AWS services. These policies grant access to `iam:passrole`, which allows a user with the policy to pass a role

to an AWS service. This role delegates IAM permissions to the AWS service to carry out actions on your behalf.

You must create the roles according to your needs. For example, the Network Administrator policy allows a user with the policy to pass a role named "flow-logs-vpc" to the Amazon CloudWatch service. CloudWatch uses that role to log and capture IP traffic for VPCs created by the user.

To follow security best practices, the policies for job functions include filters that limit the names of valid roles that can be passed. This helps avoid granting unnecessary permissions. If your users do require the optional service roles, you must create a role that follows the naming convention specified in the policy. You then grant permissions to the role. Once that is done, the user can configure the service to use the role, granting it whatever permissions the role provides.

Keep Up to Date

These policies are all maintained by AWS and are kept up to date to include support for new services and new capabilities as they are added by AWS. These policies cannot be modified by customers. You can make a copy of the policy and then modify the copy, but that copy is not automatically updated as AWS introduces new services and API operations.

Job functions

Names of policies

- [Administrator \(p. 685\)](#)
- [Billing \(p. 685\)](#)
- [Database administrator \(p. 686\)](#)
- [Data scientist \(p. 687\)](#)
- [Developer power user \(p. 687\)](#)
- [Network administrator \(p. 688\)](#)
- [Security auditor \(p. 688\)](#)
- [Support user \(p. 688\)](#)
- [System administrator \(p. 689\)](#)
- [View-only user \(p. 689\)](#)

In the following sections, each policy's name is a link to the policy details page in the AWS Management Console. There you can see the policy document and review the permissions it grants.

Administrator

AWS managed policy name: [AdministratorAccess](#)

Use case: This user has full access and can delegate permissions to every service and resource in AWS.

Policy description: This policy grants all actions for all AWS services and for all resources in the account.

Note

Before an IAM user or role can access the AWS Billing and Cost Management console with the permissions in this policy, you must first activate IAM user and role access. To do this, follow the instructions in [Step 1 of the tutorial about delegating access to the billing console \(p. 30\)](#).

Billing

AWS managed policy name: [Billing](#)

Use case: This user needs to view billing information, set up payments, and authorize payments. The user can monitor the costs accumulated for the entire AWS service.

Policy description: This policy grants full permissions for managing billing, costs, payment methods, budgets, and reports.

Note

Before an IAM user or role can access the AWS Billing and Cost Management console with the permissions in this policy, you must first activate IAM user and role access. To do this, follow the instructions in [Step 1 of the tutorial about delegating access to the billing console \(p. 30\)](#).

Database administrator

AWS managed policy name: [DatabaseAdministrator](#)

Use case: This user sets up, configures, and maintains databases in the AWS Cloud.

Policy description: This policy grants permissions to create, configure, and maintain databases. It includes access to AWS database services, such as Amazon DynamoDB, Amazon Relational Database Service (RDS), and Amazon Redshift. View the policy for the full list of database services that this policy supports.

This job function policy supports the ability to pass roles to AWS services. The policy allows the `iam:PassRole` action for only those roles named in the following table. For more information, see [Creating roles and attaching policies \(console\) \(p. 689\)](#) later in this topic.

Optional IAM service roles for the database administrator job function

Use case	Role name (* is a wildcard)	Service role type to select	Select this AWS managed policy
Allow the user to monitor RDS databases	rds-monitoring-role	Amazon RDS Role for Enhanced Monitoring	AmazonRDSEnhancedMonitoringRole
Allow AWS Lambda to monitor your database and access external databases	rdbms-lambda-access	Amazon EC2	AWSLambdaFullAccess
Allow Lambda to upload files to Amazon S3 and to Amazon Redshift clusters with DynamoDB	lambda_exec_role	AWS Lambda	Create a new managed policy as defined in the AWS Big Data Blog
Allow Lambda functions to act as triggers for your DynamoDB tables	lambda-dynamodb-*	AWS Lambda	AWSLambdaDynamoDBExecutionRole
Allow Lambda functions to access Amazon RDS in a VPC	lambda-vpc-execution-role	Create a role with a trust policy as defined in the AWS Lambda Developer Guide	AWSLambdaVPCAccessExecutionRole
Allow AWS Data Pipeline to access your AWS resources	DataPipelineDefaultRole	Create a role with a trust policy as defined in the AWS Data Pipeline Developer Guide	AWSDataPipelineRole
Allow your applications running on Amazon EC2	DataPipelineDefaultRole	Create a role with a trust policy as defined in the AWS Data Pipeline Developer Guide	AmazonEC2RoleforDataPipelineRole

Use case	Role name (* is a wildcard)	Service role type to select	Select this AWS managed policy
instances to access your AWS resources		AWS Data Pipeline Developer Guide	

Data scientist

AWS managed policy name: [DataScientist](#)

Use case: This user runs Hadoop jobs and queries. The user also accesses and analyzes information for data analytics and business intelligence.

Policy description: This policy grants permissions to create, manage, and run queries on an Amazon EMR cluster and perform data analytics with tools such as Amazon QuickSight. The policy includes access to additional data scientist services, such as AWS Data Pipeline, Amazon EC2, Amazon Kinesis, Amazon Machine Learning, and SageMaker. View the policy for the full list of data scientist services that this policy supports.

This job function policy supports the ability to pass roles to AWS services. One statement allows passing any role to SageMaker. Another statement allows the `iam:PassRole` action for only those roles named in the following table. For more information, see [Creating roles and attaching policies \(console\) \(p. 689\)](#) later in this topic.

Optional IAM service roles for the data scientist job function

Use case	Role name (* is a wildcard)	Service role type to select	AWS managed policy to select
Allow Amazon EC2 instances access to services and resources suitable for clusters	EMR-EC2_DefaultRole	Amazon EMR for EC2	AmazonElasticMapReduceforEC2Role
Allow Amazon EMR access to access the Amazon EC2 service and resources for clusters	EMR_DefaultRole	Amazon EMR	AmazonElasticMapReduceRole
Allow Kinesis Data Analytics to access streaming data sources	kinesis-*	Create a role with a trust policy as defined in the AWS Big Data Blog .	See the AWS Big Data Blog , which outlines four possible options depending on your use case
Allow AWS Data Pipeline to access your AWS resources	DataPipelineDefaultRole	Create a role with a trust policy as defined in the AWS Data Pipeline Developer Guide	AWSDataPipelineRole
Allow your applications running on Amazon EC2 instances to access your AWS resources	DataPipelineDefaultRoleRes	Create a role with a trust policy as defined in the AWS Data Pipeline Developer Guide	AmazonEC2RoleforDataPipelineRole

Developer power user

AWS managed policy name: [PowerUserAccess](#)

Use case: This user performs application development tasks and can create and configure resources and services that support AWS aware application development.

Policy description: The first statement of this policy uses the [NotAction \(p. 638\)](#) element to allow all actions for all AWS services and for all resources except AWS Identity and Access Management and AWS Organizations. The second statement grants IAM permissions to create a service-linked role. This is required by some services that must access resources in another service, such as an Amazon S3 bucket. It also grants Organizations permissions to view information about the user's organization, including the management account email and organization limitations. Although this policy limits IAM and Organizations access, it allows the user to perform all AWS SSO actions if AWS SSO is enabled.

Network administrator

AWS managed policy name: [NetworkAdministrator](#)

Use case: This user is tasked with setting up and maintaining AWS network resources.

Policy description: This policy grants permissions to create and maintain network resources in Auto Scaling, Amazon EC2, AWS Direct Connect, Route 53, Amazon CloudFront, Elastic Load Balancing, AWS Elastic Beanstalk, Amazon SNS, CloudWatch, CloudWatch Logs, Amazon S3, IAM, and Amazon Virtual Private Cloud.

This job function requires the ability to pass roles to AWS services. The policy grants `iam:GetRole` and `iam:PassRole` for only those roles named in the following table. For more information, see [Creating roles and attaching policies \(console\) \(p. 689\)](#) later in this topic.

Optional IAM service roles for the network administrator job function

Use case	Role name (* is a wildcard)	Service role type to select	AWS managed policy to select
Allows Amazon VPC to create and manage logs in CloudWatch Logs on the user's behalf to monitor IP traffic going in and out of your VPC	flow-logs-*	Create a role with a trust policy as defined in the Amazon VPC User Guide	This use case does not have an existing AWS managed policy, but the documentation lists the required permissions. See Amazon VPC User Guide .

Security auditor

AWS managed policy name: [SecurityAudit](#)

Use case: This user monitors accounts for compliance with security requirements. This user can access logs and events to investigate potential security breaches or potential malicious activity.

Policy description: This policy grants permissions to view configuration data for many AWS services and to review their logs.

Support user

AWS managed policy name: [SupportUser](#)

Use case: This user contacts AWS Support, creates support cases, and views the status of existing cases.

Policy description: This policy grants permissions to create and update AWS Support cases.

System administrator

AWS managed policy name: [SystemAdministrator](#)

Use case: This user sets up and maintains resources for development operations.

Policy description: This policy grants permissions to create and maintain resources across a large variety of AWS services, including AWS CloudTrail, Amazon CloudWatch, AWS CodeCommit, AWS CodeDeploy, AWS Config, AWS Directory Service, Amazon EC2, AWS Identity and Access Management, AWS Key Management Service, AWS Lambda, Amazon RDS, Route 53, Amazon S3, Amazon SES, Amazon SQS, AWS Trusted Advisor, and Amazon VPC.

This job function requires the ability to pass roles to AWS services. The policy grants `iam:GetRole` and `iam:PassRole` for only those roles named in the following table. For more information, see [Creating roles and attaching policies \(console\) \(p. 689\)](#) later in this topic.

Optional IAM service roles for the system administrator job function

Use case	Role name (* is a wildcard)	Service role type to select	AWS managed policy to select
Allow apps running in EC2 instances in an Amazon ECS cluster to access Amazon ECS	ecr-sysadmin-*	Amazon EC2 Role for EC2 Container Service	AmazonEC2ContainerServiceforEC2Role
Allow a user to monitor databases	rds-monitoring-role	Amazon RDS Role for Enhanced Monitoring	AmazonRDSEnhancedMonitoringRole
Allow apps running in EC2 instances to access AWS resources.	ec2-sysadmin-*	Amazon EC2	Sample policy for role that grants access to an S3 bucket as shown in the Amazon EC2 User Guide for Linux Instances ; customize as needed
Allow Lambda to read DynamoDB streams and write to CloudWatch Logs	lambda-sysadmin-*	AWS Lambda	AWSLambdaDynamoDBExecutionRole

View-only user

AWS managed policy name: [ViewOnlyAccess](#)

Use case: This user can view a list of AWS resources and basic metadata in the account across all services. The user cannot read resource content or metadata that goes beyond the quota and list information for resources.

Policy description: This policy grants `List*`, `Describe*`, `Get*`, `View*`, and `Lookup*` access to resources for most AWS services. To see what actions this policy includes for each service, see [ViewOnlyAccess](#).

Creating roles and attaching policies (console)

Several of the previously listed policies grant the ability to configure AWS services with roles that enable those services to perform operations on your behalf. The job function policies either specify exact role names that you must use or at least include a prefix that specifies the first part of the name that can be used. To create one of these roles, perform the steps in the following procedure.

To create a role for an AWS service (IAM console)

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 IAM console, choose **Roles**, and then choose **Create role**.
3. Choose the **AWS service** role type, and then choose the service that you want to allow to assume this role.
4. Choose the use case for your service. If the specified service has only one use case, it is selected for you. Use cases are defined by the service to include the trust policy that the service requires. Then choose **Next: Permissions**.
5. If possible, select the policy to use for the permissions policy or choose **Create policy** to open a new browser tab and create a new policy from scratch. For more information, see step 4 in the procedure [Creating IAM policies](#) in the *IAM User Guide*. After you create the policy, close that tab and return to your original tab. Select the check box next to the permissions policies that you want the service to have.

Depending on the use case that you selected, the service might allow you to do any of the following:

- Nothing, because the service defines the permissions for the role
 - Allow you to choose from a limited set of permissions
 - Allow you to choose from any permissions
 - Allow you to select no policies at this time, create the policies later, and then attach them to the role
6. (Optional) Set a [permissions boundary](#). This is an advanced feature that is available for service roles, but not service-linked roles.

Expand the **Set permissions boundary** section and choose **Use a permissions boundary to control the maximum role permissions**. IAM includes a list of the AWS managed and customer managed policies in your account. Select the policy to use for the permissions boundary or choose **Create policy** to open a new browser tab and create a new policy from scratch. For more information, see step 4 in the procedure [Creating IAM policies](#) in the *IAM User Guide*. After you create the policy, close that tab and return to your original tab to select the policy to use for the permissions boundary.

7. Choose **Next: Tags**.
8. (Optional) Add metadata to the user by attaching tags as key-value pairs. For more information about using tags in IAM, see [Tagging IAM entities](#) in the *IAM User Guide*.
9. Choose **Next: Review**.
10. For **Role name**, the degree of role name customization is defined by the service. If the service defines the role's name, this option is not editable. In other cases, the service might define a prefix for the role and allow you to enter an optional suffix. Some services allow you to specify the entire name of your role.

If possible, enter a role name or role name suffix to help you identify the purpose of this role. Role names must be unique within your AWS account. They are not distinguished by case. For example, you cannot create roles named both **PRODROLE** and **prodrole**. Because various entities might reference the role, you cannot edit the name of the role after it has been created.

11. (Optional) For **Role description**, enter a description for the new role.
12. Review the role and then choose **Create role**.

Example 1: Configuring a user as a database administrator (console)

This example shows the steps required to configure Alice, an IAM user, as a [Database Administrator](#) (p. 686). You use the information in first row of the table in that section and allow the

user to enable Amazon RDS monitoring. You attach the [DatabaseAdministrator](#) policy to Alice's IAM user so that she can manage the Amazon database services. That policy also enables Alice to pass a role called `rds-monitoring-role` to the Amazon RDS service that allows the service to monitor the RDS databases on her behalf.

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. Choose **Policies** and then type **database** in the search box.
3. Select the check box for the **DatabaseAdministrator** policy, choose **Policy actions**, and then choose **Attach**.
4. In the list of users, select **Alice** and then choose **Attach policy**. Alice now can administer AWS databases. However, to allow Alice to monitor those databases, you must configure the service role.
5. In the navigation pane of the IAM console, choose **Roles**, and then choose **Create role**.
6. Choose the **AWS Service** role type, and then choose **Amazon RDS**.
7. Choose the **Amazon RDS Role for Enhanced Monitoring** use case.
8. Amazon RDS defines the permissions for your role. Choose **Next: Review** to continue.
9. The role name must be one of those specified by the DatabaseAdministrator policy that Alice now has. One of those is `rds-monitoring-role`. Type that for the **Role name**.
10. (Optional) For **Role description**, type a description for the new role.
11. After you review the details, choose **Create role**.
12. Alice can now enable **RDS Enhanced Monitoring** in the **Monitoring** section of the Amazon RDS console. For example, she might do this when she creates a DB instance, creates a read replica, or modifies a DB instance. She must type the role name she created (`rds-monitoring-role`) in the **Monitoring Role** box when she sets **Enable Enhanced Monitoring** to **Yes**.

Example 2: Configuring a user as a network administrator (console)

This example shows the steps required to configure Juan, an IAM user, as a [Network Administrator](#) (p. 688). It uses the information in the table in that section to allow Juan to monitor IP traffic going to and from a VPC. It also allows Juan to capture that information in the logs in CloudWatch Logs. You attach the [NetworkAdministrator](#) policy to Juan's IAM user so that he can configure AWS network resources. That policy also enables Juan to pass a role whose name begins with `f1ow-logs*` to Amazon EC2 when you create a flow log. In this scenario, unlike Example 1, there isn't a predefined service role type, so you must perform a few steps differently.

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, choose **Policies** and then type **network** in the search box.
3. Select the check box next to **NetworkAdministrator** policy, choose **Policy actions**, and then choose **Attach**.
4. In the list of users, select the check box next to **Juan** and then choose **Attach policy**. Juan now can administer AWS network resources. However, to enable monitoring of IP traffic in your VPC, you must configure the service role.
5. Because the service role you need to create doesn't have a predefined managed policy, you must first create it. In the navigation pane, choose **Policies**, then choose **Create policy**.
6. Choose the **JSON** tab and copy the text from the following JSON policy document. Paste this text into the **JSON** text box.

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

```

    "Statement": [
        {
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:DescribeLogGroups",
                "logs:DescribeLogStreams"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}

```

- When you are finished, choose **Review policy**. The [Policy Validator \(p. 444\)](#) reports any syntax errors.

Note

You can switch between the **Visual editor** and **JSON** tabs any time. However, if you make changes or choose **Review policy** in the **Visual editor** tab, IAM might restructure your policy to optimize it for the visual editor. For more information, see [Policy restructuring \(p. 571\)](#).

- On the **Review** page, type **vpc-flow-logs-policy-for-service-role** for the policy name. Review the policy **Summary** to see the permissions granted by your policy, and then choose **Create policy** to save your work.

The new policy appears in the list of managed policies and is ready to attach.

- In the navigation pane of the IAM console, choose **Roles**, and then choose **Create role**.
- Choose the **AWS Service** role type, and then choose **Amazon EC2**.
- Choose the **Amazon EC2** use case.
- On the **Attach permissions policies** page, choose the policy you created earlier, **vpc-flow-logs-policy-for-service-role**, and then choose **Next: Review**.
- The role name must be permitted by the NetworkAdministrator policy that Juan now has. Any name that begins with **flow-logs-** is allowed. For this example, type **flow-logs-for-juan** for the **Role name**.
- (Optional) For **Role description**, type a description for the new role.
- After you review the details, choose **Create role**.
- Now you can configure the trust policy required for this scenario. On the **Roles** page, choose the **flow-logs-for-juan** role (the name, not the check box). On the details page for your new role, choose the **Trust relationships** tab, and then choose **Edit trust relationship**.
- Change the "Service" line to read as follows, replacing the entry for `ec2.amazonaws.com`:

```

    "Service": "vpc-flow-logs.amazonaws.com"

```

- Juan can now create flow logs for a VPC or subnet in the Amazon EC2 console. When you create the flow log, specify the **flow-logs-for-juan** role. That role has the permissions to create the log and write data to it.

AWS global condition context keys

When a [principal \(p. 5\)](#) makes a [request \(p. 5\)](#) to AWS, AWS gathers the request information into a [request context \(p. 5\)](#). You can use the **Condition** element of a JSON policy to compare keys in the request context with key values that you specify in your policy. To learn more about the circumstances under which a global key is included in the request context, see the **Availability** information for each global condition key. For information about how to use the **Condition** element in a JSON policy, see [IAM JSON policy elements: Condition \(p. 641\)](#).

Note

If you use condition keys that are available only in some circumstances, you can use the [IfExists \(p. 650\)](#) versions of the condition operators. If the condition keys are missing from a request context, the policy can fail the evaluation. For example, use the following condition block with `...IfExists` operators to match when a request comes from a specific IP range or from a specific VPC. If either or both keys are not included in the request context, the condition still returns `true`. The values are only checked if the specified key is included in the request context.

```
"Condition": {
    "IpAddressIfExists": {"aws:SourceIp" : ["xxx"] },
    "StringEqualsIfExists" : {"aws:SourceVpc" : ["yyy"]}
}
```

Global condition keys are condition keys with an `aws:` prefix. AWS services can support global condition keys or provide service-specific keys that include their service prefix. For example, IAM condition keys include the `iam:` prefix. For more information, see [Actions, Resources, and Condition Keys for AWS Services](#) and choose the service whose keys you want to view.

aws:CalledVia

Works with [string operators \(p. 644\)](#).

Use this key to compare the services in the policy with the services that made requests on behalf of the IAM principal (user or role). When a principal makes a request to an AWS service, that service might use the principal's credentials to make subsequent requests to other services. The `aws:CalledVia` key contains an ordered list of each service in the chain that made requests on the principal's behalf.

For example, you can use AWS CloudFormation to read and write from an Amazon DynamoDB table. DynamoDB then uses encryption supplied by AWS Key Management Service (AWS KMS).

- **Availability** – This key is present in the request when a service that supports `aws:CalledVia` uses the credentials of an IAM principal to make a request to another service. This key is not present if the service uses a [service role](#) or [service-linked role](#) to make a call on the principal's behalf. This key is also not present when the principal makes the call directly.

To use the `aws:CalledVia` condition key in a policy, you must provide the service principals to allow or deny AWS service requests. AWS supports using the following services with `aws:CalledVia`.

CalledVia services

AWS service	Service principal
Amazon Athena	athena.amazonaws.com
AWS CloudFormation	cloudformation.amazonaws.com
Amazon DynamoDB	dynamodb.amazonaws.com
AWS Key Management Service (AWS KMS)	kms.amazonaws.com

To allow or deny access when *any* service makes a request using the principal's credentials, use the [aws:ViaAWSService \(p. 707\)](#) condition key. That condition key supports AWS services.

The `aws:CalledVia` key is a [multivalued key \(p. 652\)](#). However, you can't enforce order using this key in a condition. Using the example above, **User 1** makes a request to AWS CloudFormation, which

calls DynamoDB, which calls AWS KMS. These are three separate requests. The final call to AWS KMS is performed by User 1 *via* AWS CloudFormation and then DynamoDB.

In this case, the `aws:CalledVia` key in the request context includes `cloudformation.amazonaws.com` and `dynamodb.amazonaws.com`, in that order. If you care only that the call was made via DynamoDB somewhere in the chain of requests, you can use this condition key in your policy.

For example, the following policy allows managing the AWS KMS key named `my-example-key`, but only if DynamoDB is one of the requesting services. The [ForAnyValue:StringEquals \(p. 653\)](#) condition operator ensures that DynamoDB is one of the calling services. If the principal makes the call to AWS KMS directly, the condition returns `false` and the request is not allowed by this policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "KmsActionsIfCalledViaDynamodb",
            "Effect": "Allow",
            "Action": [
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:ReEncrypt*",
                "kms:GenerateDataKey",
                "kms:DescribeKey"
            ],
            "Resource": "arn:aws:kms:region:111122223333:key/my-example-key",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "aws:CalledVia": ["dynamodb.amazonaws.com"]
                }
            }
        }
    ]
}
```

If you want to enforce which service makes the first or last call in the chain, you can use the [aws:CalledViaFirst \(p. 695\)](#) and [aws:CalledViaLast \(p. 695\)](#) keys. For example, the following policy allows managing the key named `my-example-key` in AWS KMS. These AWS KMS operations are allowed only if multiple requests were included in the chain. The first request must be made via AWS CloudFormation and the last via DynamoDB. If other services make requests in the middle of the chain, the operation is still allowed.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "KmsActionsIfCalledViaChain",
            "Effect": "Allow",
            "Action": [
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:ReEncrypt*",
                "kms:GenerateDataKey",
                "kms:DescribeKey"
            ],
            "Resource": "arn:aws:kms:region:111122223333:key/my-example-key",
            "Condition": {
                "StringEquals": {
                    "aws:CalledViaFirst": "cloudformation.amazonaws.com",
                    "aws:CalledViaLast": "dynamodb.amazonaws.com"
                }
            }
        }
    ]
}
```

```
        }
    ]
}
```

The [aws:CalledViaFirst \(p. 695\)](#) and [aws:CalledViaLast \(p. 695\)](#) keys are present in the request when a service uses an IAM principal's credentials to call another service. They indicate the first and last services that made calls in the chain of requests. For example, assume that AWS CloudFormation calls another service named `x_Service`, which calls DynamoDB, which then calls AWS KMS. The final call to AWS KMS is performed by `User_1` via AWS CloudFormation, then `x_Service`, and then DynamoDB. It was first called via AWS CloudFormation and last called via DynamoDB.

aws:CalledViaFirst

Works with [string operators \(p. 644\)](#).

Use this key to compare the services in the policy with the *first service* that made a request on behalf of the IAM principal (user or role). For more information, see [aws:CalledVia \(p. 693\)](#).

- **Availability** – This key is present in the request when a service uses the credentials of an IAM principal to make at least one other request to a different service. This key is not present if the service uses a [service role](#) or [service-linked role](#) to make a call on the principal's behalf. This key is also not present when the principal makes the call directly.

aws:CalledViaLast

Works with [string operators \(p. 644\)](#).

Use this key to compare the services in the policy with the *last service* that made a request on behalf of the IAM principal (user or role). For more information, see [aws:CalledVia \(p. 693\)](#).

- **Availability** – This key is present in the request when a service uses the credentials of an IAM principal to make at least one other request to a different service. This key is not present if the service uses a [service role](#) or [service-linked role](#) to make a call on the principal's behalf. This key is also not present when the principal makes the call directly.

aws:CurrentTime

Works with [date operators \(p. 646\)](#).

Use this key to compare the date and time of the request with the date and time that you specify in the policy. To view an example policy that uses this condition key, see [AWS: Allows access based on date and time \(p. 392\)](#).

- **Availability** – This key is always included in the request context.

aws:EpochTime

Works with [date operators \(p. 646\)](#) or [numeric operators \(p. 646\)](#).

Use this key to compare the date and time of the request in epoch or Unix time with the value that you specify in the policy. This key also accepts the number of seconds since January 1, 1970.

- **Availability** – This key is always included in the request context.

aws:MultiFactorAuthAge

Works with [numeric operators \(p. 646\)](#).

Use this key to compare the number of seconds since the requesting principal was authorized using MFA with the number that you specify in the policy. For more information about MFA, see [Using multi-factor authentication \(MFA\) in AWS \(p. 111\)](#).

- **Availability** – This key is included in the request context only if the principal making the call was authenticated using MFA. If MFA was not used, this key is not present.

aws:MultiFactorAuthPresent

Works with [Boolean operators \(p. 647\)](#).

Use this key to check whether multi-factor authentication (MFA) was used to validate the temporary security credentials that made the request.

- **Availability** – This key is included in the request context only when the principal uses temporary credentials to make the request. The key is not present in AWS CLI, AWS API, or AWS SDK requests that are made using long-term credentials.

Temporary credentials are used to authenticate IAM roles, federated users, IAM users with temporary tokens from `sts:GetSessionToken`, and users of the AWS Management Console. IAM user access keys are long-term credentials, but in some cases, AWS creates temporary credentials on behalf of IAM users to perform operations. In these cases, the `aws:MultiFactorAuthPresent` key is present in the request and set to a value of `false`. There are two common cases where this can happen:

- IAM users in the AWS Management Console unknowingly use temporary credentials. Users sign into the console using their user name and password, which are long-term credentials. However, in the background, the console generates temporary credentials on behalf of the user.
- If an IAM user makes a call to an AWS service, the service re-uses the user's credentials to make another request to a different service. For example, when calling Athena to access an Amazon S3 bucket, or when using AWS CloudFormation to create an Amazon EC2 instance. For the subsequent request, AWS uses temporary credentials.

To learn which services support using temporary credentials, see [AWS services that work with IAM \(p. 611\)](#).

The `aws:MultiFactorAuthPresent` key is not present when an API or CLI command is called with long-term credentials, such as user access key pairs. Therefore we recommend that when you check for this key that you use the [...IfExists \(p. 650\)](#) versions of the condition operators.

It is important to understand that the following Condition element is *not* a reliable way to check whether a request is authenticated using MFA.

```
##### WARNING: NOT RECOMMENDED #####
"Effect" : "Deny",
"Condition" : { "Bool" : { "aws:MultiFactorAuthPresent" : "false" } }
```

This combination of the Deny effect, Bool element, and `false` value denies requests that can be authenticated using MFA, but were not. This applies only to temporary credentials that support using MFA. This statement does not deny access to requests that are made using long-term credentials, or to requests that are authenticated using MFA. Use this example with caution because its logic is complicated and it does not test whether MFA-authentication was actually used.

Also do not use the combination of the Deny effect, Null element, and true because it behaves the same way and the logic is even more complicated.

Recommended Combination

We recommend that you use the [BoolIfExists \(p. 650\)](#) operator to check whether a request is authenticated using MFA.

```
"Effect" : "Deny",
"Condition" : { "BoolIfExists" : { "aws:MultiFactorAuthPresent" : "false" } }
```

This combination of Deny, BoolIfExists, and false denies requests that are not authenticated using MFA. Specifically, it denies requests from temporary credentials that do not include MFA. It also denies requests that are made using long-term credentials, such as AWS CLI or AWS API operations made using access keys. The *IfExists operator checks for the presence of the aws:MultiFactorAuthPresent key and whether or not it could be present, as indicated by its existence. Use this when you want to deny any request that is not authenticated using MFA. This is more secure, but can break any code or scripts that use access keys to access the AWS CLI or AWS API.

Alternative Combinations

You can also use the [BoolIfExists \(p. 650\)](#) operator to allow MFA-authenticated requests and AWS CLI or AWS API requests that are made using long-term credentials.

```
"Effect" : "Allow",
"Condition" : { "BoolIfExists" : { "aws:MultiFactorAuthPresent" : "true" } }
```

This condition matches either if the key exists and is present **or** if the key does not exist. This combination of Allow, BoolIfExists, and true allows requests that are authenticated using MFA, or requests that cannot be authenticated using MFA. This means that AWS CLI, AWS API, and AWS SDK operations are allowed when the requester uses their long-term access keys. This combination does not allow requests from temporary credentials that could, but do not include MFA.

When you create a policy using the IAM console visual editor and choose **MFA required**, this combination is applied. This setting requires MFA for console access, but allows programmatic access with no MFA.

Alternatively, you can use the Bool operator to allow programmatic and console requests only when authenticated using MFA.

```
"Effect" : "Allow",
"Condition" : { "Bool" : { "aws:MultiFactorAuthPresent" : "true" } }
```

This combination of the Allow, Bool, and true allows only MFA-authenticated requests. This applies only to temporary credentials that support using MFA. This statement does not allow access to requests that were made using long-term access keys, or to requests made using temporary credentials without MFA.

Do not use a policy construct similar to the following to check whether the MFA key is present:

```
##### WARNING: USE WITH CAUTION #####
"Effect" : "Allow",
"Condition" : { "Null" : { "aws:MultiFactorAuthPresent" : "false" } }
```

This combination of the Allow effect, Null element, and false value allows only requests that can be authenticated using MFA, regardless of whether the request is actually authenticated. This allows all

requests that are made using temporary credentials, and denies access for long-term credentials. Use this example with caution because it does not test whether MFA-authentication was actually used.

aws:PrincipalAccount

Works with [string operators \(p. 644\)](#).

Use this key to compare the account to which the requesting principal belongs with the account identifier that you specify in the policy.

- **Availability** – This key is always included in the request context.

aws:PrincipalArn

Works with [ARN operators \(p. 649\)](#) and [string operators \(p. 644\)](#).

Use this key to compare the [Amazon Resource Name \(p. 601\)](#) (ARN) of the principal that made the request with the ARN that you specify in the policy. For IAM roles, the request context returns the ARN of the role, not the ARN of the user that assumed the role. To learn which types of principals you can specify in this condition key, see [Specifying a principal \(p. 631\)](#).

- **Availability** – This key is always included in the request context.

aws:PrincipalOrgID

Works with [string operators \(p. 644\)](#).

Use this key to compare the identifier of the organization in AWS Organizations to which the requesting principal belongs with the identifier specified in the policy.

- **Availability** – This key is included in the request context only if the principal is a member of an organization.

This global key provides an alternative to listing all the account IDs for all AWS accounts in an organization. You can use this condition key to simplify specifying the `Principal` element in a [resource-based policy \(p. 374\)](#). You can specify the `organization ID` in the condition element. When you add and remove accounts, policies that include the `aws:PrincipalOrgID` key automatically include the correct accounts and don't require manual updating.

For example, the following Amazon S3 bucket policy allows members of any account in the o-xxxxxxxxxxxx organization to add an object into the `policy-ninja-dev` bucket.

```
{  
    "Version": "2012-10-17",  
    "Statement": {  
        "Sid": "AllowPutObject",  
        "Effect": "Allow",  
        "Principal": "*",  
        "Action": "s3:PutObject",  
        "Resource": "arn:aws:s3:::policy-ninja-dev/*",  
        "Condition": {"StringEquals":  
            {"aws:PrincipalOrgID": ["o-xxxxxxxxxxxx"]}}  
    }  
}
```

Note

This global condition also applies to the management account of an AWS organization.

For more information about AWS Organizations, see [What Is AWS Organizations?](#) in the *AWS Organizations User Guide*.

aws:PrincipalOrgPaths

Works with [string operators](#) (p. 644).

Use this key to compare the AWS Organizations path for the principal who is making the request to the path in the policy. That principal can be an IAM user, IAM role, federated user, or AWS account root user. In a policy, this condition key ensures that the requester is an account member within the specified organization root or organizational units (OUs) in AWS Organizations. An AWS Organizations path is a text representation of the structure of an Organizations entity. For more information about using and understanding paths, see [Understand the AWS Organizations entity path](#) (p. 481).

- **Availability** – This key is included in the request context only if the principal is a member of an organization.

Note

Organization IDs are globally unique but OU IDs and root IDs are unique only within an organization. This means that no two organizations share the same organization ID. However, another organization might have an OU or root with the same ID as yours. We recommend that you always include the organization ID when you specify an OU or root.

For example, the following condition returns true for principals in accounts that are attached directly to the ou-jk10-awsdddd OU, but not in its child OUs.

```
"Condition" : { "ForAnyValue:StringEquals" : {
    "aws:PrincipalOrgPaths": ["o-a1b2c3d4e5/r-f6g7h8i9j0example/ou-ghio-awsccccc/ou-jk10-
awsdddd/"]
}}
```

The following condition returns true for principals in an account that is attached directly to the OU or any of its child OUs. When you include a wildcard, you must use the `StringLike` condition operator.

```
"Condition" : { "ForAnyValue:StringLike" : {
    "aws:PrincipalOrgPaths": ["o-a1b2c3d4e5/r-f6g7h8i9j0example/ou-ghio-awsccccc/ou-jk10-
awsdddd/*"]
}}
```

The following condition returns true for principals in an account that is attached directly to the OU or any of its child OUs. The previous condition is for the OU or any children. The following condition is for only the children.

```
"Condition" : { "ForAnyValue:StringLike" : {
    "aws:PrincipalOrgPaths": ["o-a1b2c3d4e5/r-f6g7h8i9j0example/ou-ghio-awsccccc/ou-jk10-
awsdddd/*"]
}}
```

The following condition allows access for every principal in the o-a1b2c3d4e5 organization, regardless of their parent OU.

```
"Condition" : { "ForAnyValue:StringLike" : {
```

```
    "aws:PrincipalOrgPaths": [ "o-a1b2c3d4e5/*" ]  
}}
```

`aws:PrincipalOrgPaths` is a multivalued condition key. Multivalued keys include one or more values in a list format. The result is a logical OR. When you use multiple values with the `ForAnyValue` condition operator, the principal's path must match one of the paths listed in the policy. For policies that include multiple values for a single key, you must enclose the conditions within brackets like an array ("Key":["Value1", "Value2"]). You should also include these brackets when there is a single value. For more information about multivalued condition keys, see [Creating a condition with multiple keys or values \(p. 652\)](#).

```
"Condition": {  
    "ForAnyValue:StringLike": {  
        "aws:PrincipalOrgPaths": [  
            "o-a1b2c3d4e5/r-f6g7h8i9j0example/ou-def0-awsbbbbbb/*",  
            "o-a1b2c3d4e5/r-f6g7h8i9j0example/ou-jkl0-awsdddddd/*"  
        ]  
    }  
}
```

aws:PrincipalTag

Works with [string operators \(p. 644\)](#).

Use this key to compare the tag attached to the principal making the request with the tag that you specify in the policy. If the principal has more than one tag attached, the request context includes one `aws:PrincipalTag` key for each attached tag key.

- **Availability** – This key is included in the request context if the principal is using an IAM user with attached tags. It is included for a principal using an IAM role with attached tags or [session tags \(p. 293\)](#).

You can add custom attributes to a user or role in the form of a key-value pair. For more information about IAM tags, see [Tagging IAM users and roles \(p. 289\)](#). You can use `aws:PrincipalTag` to [control access \(p. 385\)](#) for AWS principals.

This example shows how you might create a policy that allows users with the `tagManager=true` tag to manage IAM users, groups, or roles. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#).

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "iam:*",  
            "Resource": "*",  
            "Condition": {"StringEquals": {"aws:PrincipalTag/tagManager": "true"}}  
        }  
    ]  
}
```

aws:PrincipalType

Works with [string operators \(p. 644\)](#).

Use this key to compare the type of principal making the request with the principal type that you specify in the policy. For more information, see [Specifying a principal \(p. 631\)](#).

- **Availability** – This key is always included in the request context.

aws:referer

Works with [string operators \(p. 644\)](#).

Use this key to compare who referred the request in the client browser with the referer that you specify in the policy. The `aws:referer` request context value is provided by the caller in an HTTP header. The `Referer` header is included in a web browser request when you select a link on a web page. The `Referer` header contains the URL of the web page where the link was selected.

- **Availability** – This key is included in the request context only if the request to the AWS resource was invoked by linking from a web page URL in the browser. This key is not included for programmatic requests because it doesn't use a browser link to access the AWS resource.

For example, you can access an Amazon S3 object directly using a URL or using direct API invocation. For more information, see [Amazon S3 API operations directly using a web browser](#). When you access an Amazon S3 object from a URL that exists in a webpage, the URL of the source web page is used in `aws:referer`. When you access an Amazon S3 object by typing the URL into your browser, `aws:referer` is not present. When you invoke the API directly, `aws:referer` is also not present. You can use the `aws:referer` condition key in a policy to allow requests made from a specific referer, such as a link on a web page in your company's domain.

Warning

This key should be used carefully. It is dangerous to include a publicly known referer header value. Unauthorized parties can use modified or custom browsers to provide any `aws:referer` value that they choose. As a result, `aws:referer` should not be used to prevent unauthorized parties from making direct AWS requests. It is offered only to allow customers to protect their digital content, such as content stored in Amazon S3, from being referenced on unauthorized third-party sites.

aws:RequestedRegion

Works with [string operators \(p. 644\)](#).

Use this key to compare the AWS Region that was called in the request with the Region that you specify in the policy. You can use this global condition key to control which Regions can be requested. To view the AWS Regions for each service, see [Service endpoints and quotas](#) in the *Amazon Web Services General Reference*.

- **Availability** – This key is always included in the request context.

Some global services, such as IAM, have a single endpoint. Because this endpoint is physically located in the US East (N. Virginia) Region, IAM calls are always made to the `us-east-1` Region. For example, if you create a policy that denies access to all services if the requested Region is not `us-west-2`, then IAM calls always fail. To view an example of how to work around this, see [NotAction with Deny \(p. 638\)](#).

Note

The `aws:RequestedRegion` condition key allows you to control which endpoint of a service is invoked but does not control the impact of the operation. Some services have cross-Region impacts. For example, Amazon S3 has API operations that control cross-Region replication. You can invoke `s3:PutBucketReplication` in one Region (which is affected by the `aws:RequestedRegion` condition key), but other Regions are affected based on the replications configuration settings.

You can use this context key to limit access to AWS services within a given set of Regions. For example, the following policy allows a user to view all of the Amazon EC2 instances in the AWS Management Console. However it only allows them to make changes to instances in Ireland (eu-west-1), London (eu-west-2), or Paris (eu-west-3).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "InstanceConsoleReadOnly",
            "Effect": "Allow",
            "Action": [
                "ec2:Describe*",
                "ec2:Export*",
                "ec2:Get*",
                "ec2:Search*"
            ],
            "Resource": "*"
        },
        {
            "Sid": "InstanceWriteRegionRestricted",
            "Effect": "Allow",
            "Action": [
                "ec2:Associate*",
                "ec2:Import*",
                "ec2:Modify*",
                "ec2:Monitor*",
                "ec2:Reset*",
                "ec2:Run*",
                "ec2:Start*",
                "ec2:Stop*",
                "ec2:Terminate*"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestedRegion": [
                        "eu-west-1",
                        "eu-west-2",
                        "eu-west-3"
                    ]
                }
            }
        }
    ]
}
```

aws:RequestTag/*tag-key*

Works with [string operators \(p. 644\)](#).

Use this key to compare the tag key-value pair that was passed in the request with the tag pair that you specify in the policy. For example, you could check whether the request includes the tag key "Dept" and that it has the value "Accounting". For more information, see [Controlling access during AWS requests \(p. 388\)](#).

- **Availability** – This key is included in the request context when tags are passed in the request. When multiple tags are passed in the request, there is one context key for each tag key-value pair.

This context key is formatted "aws:RequestTag/*tag-key*":"*tag-value*" where *tag-key* and *tag-value* are a tag key and value pair.

Because you can include multiple tag key-value pairs in a request, the request content could be a [multivalued \(p. 652\)](#) request. In this case, you should consider using the `ForAllValues` or `ForAnyValue` set operators. For more information, see [Using multiple keys and values \(p. 653\)](#).

aws:ResourceTag/*tag-key*

Works with [string operators \(p. 644\)](#).

Use this key to compare the tag key-value pair that you specify in the policy with the key-value pair that is attached to the resource. For example, you could require that access to a resource is allowed only if the resource has the attached tag key "Dept" with the value "Marketing". For more information, see [Controlling access to AWS resources \(p. 388\)](#).

- **Availability** – This key is included in the request context when the requested resource already has attached tags. This key is returned only for resources that [support authorization based on tags \(p. 611\)](#). There is one context key for each tag key-value pair.

This context key is formatted "`aws:ResourceTag/tag-key" ":" tag-value" where tag-key and tag-value are a tag key and value pair.`

For examples of using the `aws:ResourceTag` key to control access to IAM resources, see [Controlling access to AWS resources \(p. 388\)](#).

For examples of using the `aws:ResourceTag` key to control access to other AWS resources, see [Controlling access to AWS resources using resource tags \(p. 387\)](#).

For a tutorial on using the `aws:ResourceTag` condition key for attribute based access control (ABAC), see [IAM Tutorial: Define permissions to access AWS resources based on tags \(p. 45\)](#).

aws:SecureTransport

Works with [Boolean operators \(p. 647\)](#).

Use this key to check whether the request was sent using SSL. The request context returns `true` or `false`. In a policy, you can allow specific actions only if the request is sent using SSL.

- **Availability** – This key is always included in the request context.

aws:SourceAccount

Works with [string operators \(p. 644\)](#).

Use this key to compare the account ID of the resource making a service-to-service request with the account ID that you specify in the policy.

- **Availability** – This key is included in the request context only if accessing a resource triggers an AWS service to call another service on behalf of the resource owner. The calling service must pass the resource ARN of the source to the called service. This ARN includes the source account ID.

You can use this condition key to check that Amazon S3 is not being used as a [confused deputy \(p. 227\)](#). For example, when an Amazon S3 bucket update triggers an Amazon SNS topic post, the Amazon S3 service invokes the `sns:Publish` API operation. The bucket is considered the source of the SNS request and the value of the key is the account ID from the bucket's ARN.

aws:SourceArn

Works with [ARN operators \(p. 649\)](#) and [string operators \(p. 644\)](#).

Use this key to compare the [Amazon Resource Name \(ARN\) \(p. 601\)](#) of the resource making a service-to-service request with the ARN that you specify in the policy.

This key does not work with the ARN of the principal making the request. Instead, use [aws:PrincipalArn \(p. 698\)](#). The source's ARN includes the account ID, so it is not necessary to use aws:SourceAccount with aws:SourceArn.

- **Availability** – This key is included in the request context only if accessing a resource triggers an AWS service to call another service on behalf of the resource owner. The calling service must pass the ARN of the original resource to the called service.

You can use this condition key to check that Amazon S3 is not being used as a [confused deputy \(p. 227\)](#). For example, when an Amazon S3 bucket update triggers an Amazon SNS topic post, the Amazon S3 service invokes the sns:Publish API operation. The bucket is considered the source of the SNS request and the value of the key is the bucket's ARN.

aws:SourceIp

Works with IP address operators (p. 648).

Use this key to compare the requester's IP address with the IP address that you specify in the policy.

- **Availability** – This key is included in the request context, except when the requester uses a VPC endpoint to make the request.

The aws:SourceIp condition key can be used in a policy to allow principals to make requests only from within a specified IP range. However, this policy denies access if an AWS service makes calls on the principal's behalf. In this case, you can use aws:SourceIp with the [aws:ViaAWSService \(p. 707\)](#) key to ensure that the source IP restriction applies only to requests made directly by a principal.

For example, you can attach the following policy to an IAM user. This policy allows the user to put an object into the my-service-bucket Amazon S3 bucket directly if they make the call from the specified IP address. However, if the user makes another request that causes a service to call Amazon S3, the IP address restriction does not apply. The PrincipalPutObjectIfIpAddress statement restricts the IP address only if the request is not made by a service. The ServicePutObject statement allows the operation without IP address restriction if the request is made by a service.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "PrincipalPutObjectIfIpAddress",  
            "Effect": "Allow",  
            "Action": "s3:PutObject",  
            "Resource": "arn:aws:s3:::my-service-bucket/*",  
            "Condition": {  
                "Bool": {"aws:ViaAWSService": "false"},  
                "IpAddress": {"aws:SourceIp": "123.45.167.89"}  
            }  
        },  
        {  
            "Sid": "ServicePutObject",  
            "Effect": "Allow",  
            "Action": "s3:PutObject",  
            "Resource": "arn:aws:s3:::my-service-bucket/*",  
            "Condition": {  
                "Bool": {"aws:ViaAWSService": "true"}  
            }  
        }  
    ]  
}
```

```
    ]  
}
```

If the request comes from a host that uses an Amazon VPC endpoint, then the `aws:SourceIp` key is not available. You should instead use a VPC-specific key such as [aws:VpcSourceIp \(p. 704\)](#). For more information about using VPC endpoints, see [VPC Endpoints - Controlling the Use of Endpoints](#) in the *Amazon VPC User Guide*.

aws:SourceVpc

Works with [string operators \(p. 644\)](#).

Use this key to check whether the request comes from the VPC that you specify in the policy. In a policy, you can use this key to allow access to only a specific VPC. For more information, see [Restricting Access to a Specific VPC](#) in the *Amazon Simple Storage Service Developer Guide*.

- **Availability** – This key is included in the request context only if the requester uses a VPC endpoint to make the request.

aws:SourceVpce

Works with [string operators \(p. 644\)](#).

Use this key to compare the VPC endpoint identifier of the request with the endpoint ID that you specify in the policy. In a policy, you can use this key to restrict access to a specific VPC endpoint. For more information, see [Restricting Access to a Specific VPC Endpoint](#) in the *Amazon Simple Storage Service Developer Guide*.

- **Availability** – This key is included in the request context only if the requester uses a VPC endpoint to make the request.

aws:TagKeys

Works with [string operators \(p. 644\)](#).

Use this key to compare the tag keys in a request with the keys that you specify in the policy. As a best practice when you use policies to control access using tags, use the `aws:TagKeys` condition key to define what tag keys are allowed. For example policies and more information, see [the section called "Controlling access based on tag keys" \(p. 389\)](#).

- **Availability** – This key is included in the request context only if the operation supports attaching tags to resources.

This context key is formatted "`aws:TagKeys`": "`tag-key`" where `tag-key` is a list of tag keys without values (for example, `["Dept", "Cost-Center"]`).

Because you can include multiple tag key-value pairs in a request, the request content could be a [multivalued \(p. 652\)](#) request. In this case, you should consider using the `ForAllValues` or `ForAnyValue` set operators. For more information, see [Using multiple keys and values \(p. 653\)](#).

Some services support tagging with resource operations, such as creating, modifying, or deleting a resource. To allow tagging and operations as a single call, you must create a policy that includes both the tagging action and the resource-modifying action. You can then use the `aws:TagKeys` condition key to enforce using specific tag keys in the request. For example, to limit tags when someone creates an Amazon EC2 snapshot, you must include the `ec2:CreateSnapshot` creation action

and the `ec2:CreateTags` tagging action in the policy. To view a policy for this scenario that uses `aws:TagKeys`, see [Creating a Snapshot with Tags](#) in the *Amazon EC2 User Guide for Linux Instances*.

aws:TokenIssueTime

Works with [date operators \(p. 646\)](#).

Use this key to compare the date and time that temporary security credentials were issued with the date and time that you specify in the policy.

- **Availability** – This key is included in the request context only when the principal uses temporary credentials to make the request. They key is not present in AWS CLI, AWS API, or AWS SDK requests that are made using access keys.

To learn which services support using temporary credentials, see [AWS services that work with IAM \(p. 611\)](#).

aws:UserAgent

Works with [string operators \(p. 644\)](#).

Use this key to compare the requester's client application with the application that you specify in the policy.

- **Availability** – This key is always included in the request context.

Warning

This key should be used carefully. Since the `aws:UserAgent` value is provided by the caller in an HTTP header, unauthorized parties can use modified or custom browsers to provide any `aws:UserAgent` value that they choose. As a result, `aws:UserAgent` should not be used to prevent unauthorized parties from making direct AWS requests. You can use it to allow only specific client applications, and only after testing your policy.

aws:userid

Works with [string operators \(p. 644\)](#).

Use this key to compare the requester's principal identifier with the ID that you specify in the policy. For IAM users, the request context value is the user ID. For IAM roles, this value format can vary. For details about how the information appears for different principals, see [Specifying a principal \(p. 631\)](#).

- **Availability** – This key is included in the request context for all signed requests. Anonymous requests do not include this key.

aws:username

Works with [string operators \(p. 644\)](#).

Use this key to compare the requester's user name with the user name that you specify in the policy. For details about how the information appears for different principals, see [Specifying a principal \(p. 631\)](#).

- **Availability** – This key is always included in the request context for IAM users. Anonymous requests and requests that are made using the AWS account root user or IAM roles do not include this key. Requests made using AWS SSO credentials do not include this key in the context. To learn how to

control access to AWS SSO users, see `identitystore:UserId` in [Using predefined attributes from the AWS SSO identity store for access control in AWS](#).

aws:ViaAWSService

Works with Boolean operators ([p. 647](#)).

Use this key to check whether an AWS service makes a request to another service on your behalf.

The request context key returns `true` when a service uses the credentials of an IAM principal to make a request on behalf of the principal. The context key returns `false` if the service uses a [service role](#) or [service-linked role](#) to make a call on the principal's behalf. The request context key also returns `false` when the principal makes the call directly.

- **Availability** – This key is always included in the request context for most services.

The following services do not currently support `aws:ViaAWSService`:

- Amazon EC2
- AWS Glue
- AWS Lake Formation
- AWS OpsWorks

You can use this condition key to allow or deny access based on whether a request was made by a service. To view an example policy, see [AWS: Denies access to AWS based on the source IP \(p. 404\)](#).

aws:VpcSourceIp

Works with IP address operators ([p. 648](#)).

Use this key to compare the IP address from which a request was made with the IP address that you specify in the policy. In a policy, the key matches only if the request originates from the specified IP address and it goes through a VPC endpoint.

- **Availability** – This key is included in the request context only if the request is made using a VPC endpoint.

For more information, see [Controlling Access to Services with VPC Endpoints](#) in the *Amazon VPC User Guide*.

Other cross-service condition keys

Global condition keys are condition keys with an `aws:` prefix. Individual services can create their own condition keys. These service-specific condition keys include a prefix that matches the name of the service, such as `iam:` or `sts:`.

Services can create service-specific keys that are available in the request context for other services. These keys are available across multiple services, but are not global condition keys. For example, AWS STS supports [SAML-based federation condition keys \(p. 713\)](#). These keys are available when a user who was federated using SAML performs AWS operations in other services. Other examples include `identitystore:UserId` and `ec2:SourceInstanceArn`.

To view the service-specific condition keys for a service, see [Actions, Resources, and Condition Keys for AWS Services](#) and choose the service whose keys you want to view.

IAM and AWS STS condition context keys

You can use the `Condition` element in a JSON policy to test the value of keys that are included in the request context of all AWS requests. These keys provide information about the request itself or the resources that the request references. You can check that keys have specified values before allowing the action requested by the user. This gives you granular control over when your JSON policy statements match or don't match an incoming request. For information about how to use the `Condition` element in a JSON policy, see [IAM JSON policy elements: Condition \(p. 641\)](#).

This topic describes the keys defined and provided by the IAM service (with an `iam:` prefix) and the AWS Security Token Service (AWS STS) service (with an `sts:` prefix). Several other AWS services also provide service-specific keys that are relevant to the actions and resources defined by that service. For more information, see [Actions, Resources, and Condition Keys for AWS Services](#). The documentation for a service that supports condition keys often has additional information. 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](#).

Topics

- [Available keys for IAM \(p. 708\)](#)
- [Available keys for AWS web identity federation \(p. 710\)](#)
- [Available keys for SAML-based AWS STS federation \(p. 713\)](#)
- [Available keys for AWS STS \(p. 717\)](#)

Available keys for IAM

You can use the following condition keys in policies that control access to IAM resources:

iam:AssociatedResourceArn

Works with [ARN operators \(p. 649\)](#).

Specifies the ARN of the resource to which this role will be associated at the destination service. The resource usually belongs to the service to which the principal is passing the role. Sometimes, the resource might belong to a third service. For example, you might pass a role to Amazon EC2 Auto Scaling that they use on an Amazon EC2 instance. In this case, the condition would match the ARN of the Amazon EC2 instance.

This condition key applies to only the [PassRole \(p. 251\)](#) action in a policy. It can't be used to limit any other action.

Use this condition key in a policy to allow an entity to pass a role, but only if that role is associated with the specified resource. You can use wildcards (*) to allow operations performed on a specific type of resource without restricting the Region or resource ID. For example, you can allow an IAM user or role to pass any role to the Amazon EC2 service to be used with instances in the Region "us-east-1" or "us-west-1". The IAM user or role would not be allowed to pass roles to other services, and it doesn't allow Amazon EC2 to use the role with instances in other Regions.

```
{  
    "Effect": "Allow",  
    "Action": "iam:PassRole",  
    "Resource": "*",  
    "Condition": {  
        "StringEquals": {"iam:PassedToService": "ec2.amazonaws.com"},  
        "StringLike": {  
            "iam:AssociatedResourceARN": [  
                "arn:aws:ec2:us-east-1:111122223333:instance/*",  
                "arn:aws:ec2:us-west-1:111122223333:instance/*"  
            ]  
        }  
    }  
}
```

```
    } ]  
}
```

Note

AWS services that support [iam:PassedToService \(p. 709\)](#) also support this condition key.

iam:AWSServiceName

Works with [string operators \(p. 644\)](#).

Specifies the AWS service to which this role is attached.

iam:OrganizationsPolicyId

Works with [string operators \(p. 644\)](#).

Checks that the policy with the specified AWS Organizations ID matches the policy used in the request. To view an example IAM policy that uses this condition key, see [IAM: View service last accessed information for an Organizations policy \(p. 428\)](#).

iam:PassedToService

Works with [string operators \(p. 644\)](#).

Specifies the service principal of the service to which a role can be passed. This condition key applies to only the [PassRole \(p. 251\)](#) action in a policy. It can't be used to limit any other action.

When you use this condition key in a policy, specify the service using a service principal. A service principal is the name of a service that can be specified in the `Principal` element of a policy. This is the usual format: `SERVICE_NAME_URL.amazonaws.com`.

You can use `iam:PassedToService` to restrict your users so that they can pass roles only to specific services. For example, a user might create a [service role \(p. 168\)](#) that trusts CloudWatch to write log data to an Amazon S3 bucket on their behalf. Then the user must attach a permissions policy and a trust policy to the new service role. In this case, the trust policy must specify `cloudwatch.amazonaws.com` in the `Principal` element. To view a policy that allows the user to pass the role to CloudWatch, see [IAM: Pass an IAM role to a specific AWS service \(p. 422\)](#).

By using this condition key, you can ensure that users create service roles only for the services that you specify. For example, if a user with the preceding policy attempts to create a service role for Amazon EC2, the operation will fail. The failure occurs because the user does not have permission to pass the role to Amazon EC2.

Note

Some services, such as AWS CodeBuild and AWS CodeCommit do not support this condition key.

iam:PermissionsBoundary

Works with [string operators \(p. 644\)](#).

Checks that the specified policy is attached as permissions boundary on the IAM principal resource. For more information, see [Permissions boundaries for IAM entities \(p. 365\)](#)

iam:PolicyARN

Works with [ARN operators \(p. 649\)](#).

Checks the Amazon Resource Name (ARN) of a managed policy in requests that involve a managed policy. For more information, see [Controlling access to policies \(p. 380\)](#).

iam:ResourceTag/*key-name*

Works with [string operators \(p. 644\)](#).

Checks that the tag attached to the identity resource (user or role) matches the specified key name and value.

Note

IAM does not support using the [aws:ResourceTag \(p. 703\)](#) global condition key. AWS STS supports both the IAM key and the global key.

You can add custom attributes to a user or role in the form of a key-value pair. For more information about IAM tags, see [the section called “Tagging users and roles” \(p. 289\)](#). You can use `iam:ResourceTag` to [control access \(p. 385\)](#) to IAM users and roles. However, because IAM does not support tags for groups, you cannot use tags to control access to groups.

This example shows how you might create a policy that allows deleting users with the `status=terminated` tag. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in [create a policy \(p. 439\)](#) or [edit a policy \(p. 465\)](#).

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "iam:DeleteUser",  
            "Resource": "*"  
            "Condition": {"StringLike": {"iam:ResourceTag/status": "terminated"}}  
        }  
    ]  
}
```

Available keys for AWS web identity federation

You can use web identity federation to give temporary security credentials to users who have been authenticated through an identity provider (IdP). Examples of such providers include Login with Amazon, Amazon Cognito, Google, or Facebook. In that case, additional condition keys are available when the temporary security credentials are used to make a request. You can use these keys to write policies that limit the access of federated users to resources that are associated with a specific provider, app, or user. These keys are typically used in the trust policy for a role.

aws:FederatedProvider

Works with [string operators \(p. 644\)](#).

The `FederatedProvider` key identifies which of the IdPs was used to authenticate the user. For example, if the user was authenticated through Amazon Cognito, the key would contain `cognito-identity.amazonaws.com`. Similarly, if the user was authenticated through Login with Amazon, the key would contain the value `www.amazon.com`. You might use the key in a resource policy like the following, which uses the `aws:FederatedProvider` key as a policy variable in the ARN of a resource. The policy allows any user who has been authenticated using an IdP to get objects out of a folder in an Amazon S3 bucket. However, the bucket must be specific to the provider that the user authenticates with.

```
{  
    "Version": "2012-10-17",  
    "Statement": {  
        "Effect": "Allow",  
        "Principal": "*",  
        "Action": "s3:GetObject",  
        "Resource": "arn:aws:s3:::BUCKET-NAME/${aws:FederatedProvider}/*"  
    }  
}
```

amr

Works with [string operators \(p. 644\)](#).

Example: `cognito-identity.amazonaws.com:com:amr`

If you are using Amazon Cognito for web identity federation, the `cognito-identity.amazonaws.com:amr` key (Authentication Methods Reference) includes login information about the user. The key is multivalued, meaning that you test it in a policy using [condition set operators \(p. 652\)](#). The key can contain the following values:

- If the user is unauthenticated, the key contains only `unauthenticated`.
- If the user is authenticated, the key contains the value `authenticated` and the name of the login provider used in the call (`graph.facebook.com`, `accounts.google.com`, or `www.amazon.com`).

As an example, the following condition in the trust policy for an Amazon Cognito role tests whether the user is unauthenticated:

```
"Condition": {  
    "StringEquals":  
        { "cognito-identity.amazonaws.com:aud": "us-east-2:identity-pool-id" },  
    "ForAnyValue:StringLike":  
        { "cognito-identity.amazonaws.com:amr": "unauthenticated" }  
}
```

aud

Works with [string operators \(p. 644\)](#).

Use the `aud` condition key to verify that the Google client ID or Amazon Cognito identity pool ID matches the one that you specify in the policy. You can use the `aud` key with the `sub` key for the same identity provider.

Examples:

- `accounts.google.com:aud`
- `cognito-identity.amazonaws.com:aud`

The `accounts.google.com:aud` condition key matches the following Google ID Token fields.

- `aud` for OAuth 2.0 Google client IDs of your application, when the `azp` field is not set. When the `azp` field is set, the `aud` field matches the [accounts.google.com:oaud \(p. 712\)](#) condition key.
- `azp` when the `azp` field is set. This can happen for hybrid apps where a web application and Android app have a different OAuth 2.0 Google client ID but share the same Google APIs project.

For more information about Google `aud` and `azp` fields, see the [Google Identity Platform OpenID Connect Guide](#).

When you write a policy using the `accounts.google.com:aud` condition key, you must know whether the app is a hybrid app that sets the `azp` field.

azp Field Not Set

The following example policy works for non-hybrid apps that do not set the `azp` field. In this case the Google ID Token `aud` field value matches both the `accounts.google.com:aud` and the `accounts.google.com:oaud` condition key values.

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

```

    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {"Federated": "accounts.google.com"},
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "accounts.google.com:aud": "aud-value",
                    "accounts.google.com:oaud": "aud-value",
                    "accounts.google.com:sub": "sub-value"
                }
            }
        }
    ]
}

```

azp Field Set

The following example policy works for hybrid apps that do set the azp field. In this case the Google ID Token aud field value matches only the accounts.google.com:oaud condition key value. The azp field value matches the accounts.google.com:aud condition key value.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {"Federated": "accounts.google.com"},
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "accounts.google.com:aud": "azp-value",
                    "accounts.google.com:oaud": "aud-value",
                    "accounts.google.com:sub": "sub-value"
                }
            }
        }
    ]
}

```

id

Works with [string operators \(p. 644\)](#).

Examples:

- graph.facebook.com:app_id
- graph.facebook.com:id
- www.amazon.com:app_id
- www.amazon.com:user_id

Use these keys to verify that the application (or site) ID or user ID matches the one that you specify in the policy. This works for Facebook or Login with Amazon. You can use the app_id key with the id key for the same identity provider.

oaud

Works with [string operators \(p. 644\)](#).

Example: accounts.google.com:oaud

If you use Google for web identity federation, this key specifies the Google audience (aud) that this ID token is intended for. It must be one of the OAuth 2.0 client IDs of your application.

sub

Works with [string operators \(p. 644\)](#).

Examples:

- accounts.google.com:sub
- cognito-identity.amazonaws.com:sub

Use these keys to verify that the user ID matches the one that you specify in the policy. You can use the sub key with the aud key for the same identity provider.

More Information About Web Identity Federation

For more information about web identity federation, see the following:

- [Amazon Cognito Overview](#) in the *AWS Mobile SDK for Android Developer Guide* guide
- [Amazon Cognito Overview](#) in the *AWS Mobile SDK for iOS Developer Guide* guide
- [About web identity federation \(p. 177\)](#)

Available keys for SAML-based AWS STS federation

If you are working with [SAML-based federation](#) using AWS Security Token Service (AWS STS), you can include additional condition keys in the policy.

SAML role trust policies

In the trust policy of a role, you can include the following keys, which help you establish whether the caller is allowed to assume the role. Except for saml:doc, all the values are derived from the SAML assertion. All items in the list are available in the IAM console visual editor when you create or edit a policy with conditions. Items marked with [] can have a value that is a list of the specified type.

saml:aud

Works with [string operators \(p. 644\)](#).

An endpoint URL to which SAML assertions are presented. The value for this key comes from the SAML Recipient field in the assertion, *not* the Audience field.

saml:commonName[]

Works with [string operators \(p. 644\)](#).

This is a commonName attribute.

saml:cn[]

Works with [string operators \(p. 644\)](#).

This is an eduOrg attribute.

saml:doc

Works with [string operators \(p. 644\)](#).

This represents the principal that was used to assume the role. The format is *account-ID/provider-friendly-name*, such as 123456789012/SAMLProviderName. The account-ID value refers to the account that owns the [SAML provider \(p. 193\)](#).

saml:edupersonaffiliation[]

Works with [string operators \(p. 644\)](#).

This is an eduPerson attribute.

saml:edupersonassurance[]

Works with [string operators \(p. 644\)](#).

This is an eduPerson attribute.

saml:edupersonentitlement[]

Works with [string operators \(p. 644\)](#).

This is an eduPerson attribute.

saml:edupersonnickname[]

Works with [string operators \(p. 644\)](#).

This is an eduPerson attribute.

saml:edupersonorgdn

Works with [string operators \(p. 644\)](#).

This is an eduPerson attribute.

saml:edupersonorgunitdn[]

Works with [string operators \(p. 644\)](#).

This is an eduPerson attribute.

saml:edupersonprimaryaffiliation

Works with [string operators \(p. 644\)](#).

This is an eduPerson attribute.

saml:edupersonprimaryorgunitdn

Works with [string operators \(p. 644\)](#).

This is an eduPerson attribute.

saml:edupersonprincipalname

Works with [string operators \(p. 644\)](#).

This is an eduPerson attribute.

saml:edupersonscopedaffiliation[]

Works with [string operators \(p. 644\)](#).

This is an eduPerson attribute.

saml:edupersontargetedid[]

Works with [string operators \(p. 644\)](#).

This is an eduPerson attribute.

saml:eduorghomepageuri[]

Works with [string operators \(p. 644\)](#).

This is an eduOrg attribute.

saml:eduorgidentityauthnpolicyuri[]

Works with [string operators \(p. 644\)](#).

This is an eduOrg attribute.

saml:eduorglegalname[]

Works with [string operators \(p. 644\)](#).

This is an eduOrg attribute.

saml:eduorgsuperioruri[]

Works with [string operators \(p. 644\)](#).

This is an eduOrg attribute.

saml:eduorgwhitepagesuri[]

Works with [string operators \(p. 644\)](#).

This is an eduOrg attribute.

saml:givenName[]

Works with [string operators \(p. 644\)](#).

This is a givenName attribute.

saml:iss

Works with [string operators \(p. 644\)](#).

The issuer, which is represented by a URN.

saml:mail[]

Works with [string operators \(p. 644\)](#).

This is a mail attribute.

saml:name[]

Works with [string operators \(p. 644\)](#).

This is a name attribute.

saml:namequalifier

Works with [string operators \(p. 644\)](#).

A hash value based on the friendly name of the SAML provider. The value is the concatenation of the following values, in order and separated by a '/' character:

1. The Issuer response value (`saml:iss`)
2. The AWS account ID
3. The friendly name (the last part of the ARN) of the SAML provider in IAM

The concatenation of the account ID and friendly name of the SAML provider is available to IAM policies as the key `saml:doc`. For more information, see [Uniquely identifying users in SAML-based federation \(p. 185\)](#).

saml:organizationStatus[]

Works with [string operators \(p. 644\)](#).

This is an organizationStatus attribute.

saml:primaryGroupSID[]

Works with [string operators \(p. 644\)](#).

This is a `primaryGroupSID` attribute.

saml:sub

Works with [string operators \(p. 644\)](#).

This is the subject of the claim, which includes a value that uniquely identifies an individual user within an organization (for example, `_cbb88bf52c2510eabe00c1642d4643f41430fe25e3`).

saml:sub_type

Works with [string operators \(p. 644\)](#).

This key can have the value `persistent`, `transient`, or consist of the full Format URI from the `Subject` and `NameID` elements used in your SAML assertion. A value of `persistent` indicates that the value in `saml:sub` is the same for a user between sessions. If the value is `transient`, the user has a different `saml:sub` value for each session. For information about the `NameID` element's `Format` attribute, see [Configuring SAML assertions for the authentication response \(p. 199\)](#).

saml:surname[]

Works with [string operators \(p. 644\)](#).

This is a `surnameuid` attribute.

saml:uid[]

Works with [string operators \(p. 644\)](#).

This is a `uid` attribute.

saml:x500UniqueIdentifier[]

Works with [string operators \(p. 644\)](#).

This is an `x500UniqueIdentifier` attribute.

For general information about `eduPerson` and `eduOrg` attributes, see the [REFEDS Wiki website](#). For a list of `eduPerson` attributes, see [eduPerson Object Class Specification \(201602\)](#).

Condition keys whose type is a list can include multiple values. To create conditions in the policy for list values, you can use [set operators \(p. 652\)](#) (`ForAllValues`, `ForAnyValue`). For example, to allow any user whose affiliation is "faculty" or "staff" (but not "student"), you might use a condition like the following:

```
"Condition": {  
    "ForAllValues:StringLike": {  
        "saml:edupersonaffiliation": [ "faculty", "staff" ]  
    }  
}
```

SAML role permissions policies

In the permissions policy of a role for SAML federation that defines what users are allowed to access in AWS, you can include the following keys:

saml:namequalifier

Works with [string operators \(p. 644\)](#).

This contains a hash value that represents the combination of the `saml:doc` and `saml:iss` values. It is used as a namespace qualifier; the combination of `saml:namequalifier` and `saml:sub` uniquely identifies a user.

saml:sub

Works with [string operators \(p. 644\)](#).

This is the subject of the claim, which includes a value that uniquely identifies an individual user within an organization (for example, `_cbb88bf52c2510eabe00c1642d4643f41430fe25e3`).

saml:sub_type

Works with [string operators \(p. 644\)](#).

This key can have the value `persistent`, `transient`, or consist of the full Format URI from the `Subject` and `NameID` elements used in your SAML assertion. A value of `persistent` indicates that the value in `saml:sub` is the same for a user between sessions. If the value is `transient`, the user has a different `saml:sub` value for each session. For information about the `NameID` element's `Format` attribute, see [Configuring SAML assertions for the authentication response \(p. 199\)](#).

For more information about using these keys, see [About SAML 2.0-based federation \(p. 182\)](#).

Available keys for AWS STS

You can use the following condition keys in IAM role trust policies for roles that are assumed using AWS Security Token Service (AWS STS) operations.

sts:AWSServiceName

Works with [string operators \(p. 644\)](#).

Use this key to specify a service where a bearer token can be used. When you use this condition key in a policy, specify the service using a service principal. A service principal is the name of a service that can be specified in the `Principal` element of a policy. For example, `codeartifact.amazonaws.com` is the AWS CodeArtifact service principal.

Some AWS services require that you have permission to get an AWS STS service bearer token before you can access their resources programmatically. For example, AWS CodeArtifact requires principals to use bearer tokens to perform some operations. The `aws codeartifact get-authorization-token` command returns a bearer token. You can then use the bearer token to perform AWS CodeArtifact operations. For more information about bearer tokens, see [Using bearer tokens \(p. 332\)](#).

Availability – This key is present in requests that get a bearer token. You cannot make a direct call to AWS STS to get a bearer token. When you perform some operations in other services, the service requests the bearer token on your behalf.

You can use this condition key to allow principals to get a bearer token for use with a specific service.

sts:DurationSeconds

Works with [numeric operators \(p. 646\)](#).

Use this key to specify the duration (in seconds) that a principal can use when getting an AWS STS bearer token.

Some AWS services require that you have permission to get an AWS STS service bearer token before you can access their resources programmatically. For example, AWS CodeArtifact requires principals to use bearer tokens to perform some operations. The `aws codeartifact get-authorization-token` command returns a bearer token. You can then use the bearer token to perform AWS CodeArtifact operations. For more information about bearer tokens, see [Using bearer tokens \(p. 332\)](#).

Availability – This key is present in requests that get a bearer token. You cannot make a direct call to AWS STS to get a bearer token. When you perform some operations in other services, the

service requests the bearer token on your behalf. The key is not present for AWS STS assume-role operations.

sts:ExternalId

Works with [string operators \(p. 644\)](#).

Use this key to require that a principal provide a specific identifier when assuming an IAM role.

Availability – This key is present in the request when the principal provides an external ID while assuming a role using the AWS CLI or AWS API.

A unique identifier that might be required when you assume a role in another account. If the administrator of the account to which the role belongs provided you with an external ID, then provide that value in the `ExternalId` parameter. This value can be any string, such as a passphrase or account number. The primary function of the external ID is to address and prevent the confused deputy problem. For more information about the external ID and the confused deputy problem, see [How to use an external ID when granting access to your AWS resources to a third party \(p. 225\)](#).

The `ExternalId` value must have a minimum of 2 characters and a maximum of 1,224 characters. The value must be alphanumeric without white space. It can also include the following symbols: plus (+), equal (=), comma (,), period (.), at (@), colon (:), forward slash (/), and hyphen (-).

sts:RoleSessionName

Works with [string operators \(p. 644\)](#).

Use this key to compare the session name that a principal specifies when assuming a role with the value that is specified in the policy.

Availability – This key is present in the request when the principal assumes the role using the AWS Management Console, the assume-role CLI command, or the `AssumeRole` API operation.

You can use this key in a role trust policy to require that your users provide a specific session name when they assume a role. For example, you can require that IAM users specify their own user name as their session name. After the IAM user assumes the role, activity appears in [AWS CloudTrail logs \(p. 340\)](#) with the session name that matches their user name. This makes it easier for administrators to determine which user performed a specific action in AWS.

The following role trust policy requires that IAM users in account 111122223333 provide their IAM user name as the session name when they assume the role. This requirement is enforced using the `aws:username` [condition variable \(p. 658\)](#) in the condition key. This policy allows IAM users to assume the role to which the policy is attached. This policy does not allow anyone using temporary credentials to assume the role because the `username` variable is present for only IAM users.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "RoleTrustPolicyRequireUsernameForSessionName",
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Principal": {"AWS": "arn:aws:iam::111122223333:root"},
            "Condition": {
                "StringLike": {"sts:RoleSessionName": "${aws:username}"}
            }
        }
    ]
}
```

When an administrator views the AWS CloudTrail log for an action, they can compare the session name to the user names in their account. In the following example, the user named matjac

performed the operation using the role named `MateoRole`. The administrator can then contact Mateo Jackson, who has the user named `matjac`.

```
"assumedRoleUser": {  
    "assumedRoleId": "AROACQRSTUVWRDAOEXAMPLE:matjac",  
    "arn": "arn:aws:sts::111122223333:assumed-role/MateoRole/matjac"  
}
```

If you allow [cross-account access using roles \(p. 171\)](#), then users in one account can assume a role in another account. The ARN of the assumed role user listed in CloudTrail includes the account *where the role exists*. It does not include the account of the user that assumed the role. Users are unique only within an account. Therefore, we recommend that you use this method for checking CloudTrail logs only for roles that are assumed by users in accounts that you administer. Your users might use the same user name in multiple accounts.

sts:TransitiveTagKeys

Works with [string operators \(p. 644\)](#).

Use this key to compare the transitive session tag keys in the request with those specified in the policy.

Availability – This key is present in the request when you make a request using temporary security credentials. These include credentials created using any assume-role operation, or the `GetFederationToken` operation.

When you make a request using temporary security credentials, the [request context \(p. 642\)](#) includes the `aws:PrincipalTag` (p. 700) context key. This key includes a list of [session tags \(p. 293\)](#), [transitive session tags \(p. 299\)](#), and role tags. Transitive session tags are tags that persist into all subsequent sessions when you use the session credentials to assume another role. Assuming one role from another is called [role chaining \(p. 169\)](#).

You can use this condition key in a policy to require setting specific session tags as transitive when assuming a role or federating a user.

Actions, resources, and condition keys for AWS services

Each AWS service can define actions, resources, and condition context keys for use in IAM policies. For a list of AWS services and their actions, resources, and condition context keys, see [Actions, resources, and condition keys](#) in the *Service Authorization Reference*.

Resources to learn more about IAM

IAM is a rich product, and you'll find many resources to help you learn more about how IAM can help you secure your AWS account and resources.

Topics

- [Users and groups \(p. 720\)](#)
- [Credentials \(passwords, access keys, and MFA devices\) \(p. 720\)](#)
- [Permissions and policies \(p. 720\)](#)
- [Federation and delegation \(p. 721\)](#)
- [IAM and other AWS products \(p. 721\)](#)
- [General security practices \(p. 722\)](#)
- [General resources \(p. 722\)](#)

Users and groups

Consult these resources for creating, managing, and using users and groups.

- [Creating your first IAM admin user and group \(p. 20\)](#) – A step-by-step procedure that shows how to create an IAM users and assign permissions.
- [IAM Identities \(users, groups, and roles\) \(p. 72\)](#) – An in-depth discussion of how to administer IAM users and groups.
- [Guidelines for When to Use Accounts, Users, and Groups](#) – An AWS Security Blog post that discusses how to organize user access with separate AWS accounts or with IAM users and groups in a single account.

Credentials (passwords, access keys, and MFA devices)

Review the following guides to manage passwords for your AWS account and for IAM users. You'll also find information about *access keys*—the secret key that you use to make programmatic calls to AWS.

- [AWS Security Credentials](#) – Describes the types of credentials you use to access Amazon Web Services, explains how to create and manage them, and includes recommendations for managing access keys securely.
- [Managing user passwords in AWS \(p. 92\)](#) and [Managing access keys for IAM users \(p. 102\)](#) – Describes options for managing credentials for IAM users in your account.
- [Using multi-factor authentication \(MFA\) in AWS \(p. 111\)](#) – Describes how to configure your account and IAM users to require both a password and a one-time use code that is generated on a device before sign-in is allowed. (This is sometimes called two-factor authentication.)

Permissions and policies

Learn the inner workings of IAM policies and find tips on the best ways to confer permissions:

- [Policies and permissions in IAM \(p. 351\)](#) – Introduces the policy language that is used to define permissions. Describes how permissions can be attached to users or groups or, for some AWS products, to resources themselves.
- [IAM JSON policy elements reference \(p. 628\)](#) – Provides descriptions and examples of each policy language element.
- [Example IAM identity-based policies \(p. 389\)](#) – Shows examples of policies for common tasks in various AWS products.
- [AWS Policy Generator](#) – Create custom policies by choosing products and actions from a list.
- [IAM Policy Simulator](#) – Test whether a policy would allow or deny a specific request to AWS.

Federation and delegation

You can grant access to resources in your AWS account for users who are authenticated (signed in) elsewhere. These can be IAM users in another AWS account (known as *delegation*), users who are authenticated with your organization's sign-in process, or users from an Internet identity provider like Login with Amazon, Facebook, Google, or any other OpenID Connect (OIDC) compatible identity provider. In these cases, the users get temporary security credentials to access AWS resources.

- [IAM Tutorial: Delegate access across AWS accounts using IAM roles \(p. 34\)](#) – Guides you through granting cross-account access to an IAM user in another AWS account.
- [Common scenarios for temporary credentials \(p. 302\)](#) – Describes ways in which users can be federated into AWS after being authenticated outside of AWS.
- [Web Identity Federation Playground](#) – Lets you experiment with Login with Amazon, Google, or Facebook to authenticate and then make a call to Amazon S3.

IAM and other AWS products

Most AWS products are integrated with IAM so that you can use IAM features to help protect access to the resources in those products. The following resources discuss IAM and security for some of the most popular AWS products. For a complete list of products that work with IAM, including links to more information on each, see [AWS services that work with IAM \(p. 611\)](#).

Using IAM with Amazon EC2

- [Controlling Access to Amazon EC2 Resources](#) – Describes how to use IAM features to permit users to administer Amazon EC2 instances, volumes, and more.
- [Using instance profiles \(p. 270\)](#) – Describes how to use IAM roles to securely provide credentials for applications that run on Amazon EC2 instances and that need access to other AWS products.

Using IAM with Amazon S3

- [Managing Access Permissions to Your Amazon S3 Resources](#) – Discusses the Amazon S3 security model for buckets and objects, which includes IAM policies.
- [Writing IAM Policies: Grant Access to User-Specific Folders in an Amazon S3 Bucket](#) – Discusses how to let users protect their own folders in Amazon S3. (For more posts about Amazon S3 and IAM, choose the **S3** tag below the title of the blog post.)

Using IAM with Amazon RDS

- [Using AWS Identity and Access Management \(IAM\) to Manage Access to Amazon RDS Resources](#) – Describes how to use IAM to control access to database instances, database snapshots, and more.
- [A Primer on RDS Resource-Level Permissions](#) – Describes how to use IAM to control access to specific Amazon RDS instances.

Using IAM with Amazon DynamoDB

- [Using IAM to Control Access to DynamoDB Resources](#) – Describes how to use IAM to permit users to administer DynamoDB tables and indexes.
- The following video (8:55) explains how to provide access control for individual DynamoDB database items or attributes (or both).

[Getting Started with Fine-Grained Access Control for DynamoDB](#)

General security practices

Find expert tips and guidance on the best ways to secure your AWS account and resources:

- [AWS Security Best Practices \(PDF\)](#) – Provides an in-depth look at how to manage security across AWS accounts and products, including suggestions for security architecture, use of IAM, encryption and data security, and more.
- [Security best practices in IAM \(p. 527\)](#) – Offers recommendations for ways to use IAM to help secure your AWS account and resources.
- [AWS CloudTrail User Guide](#) – Use AWS CloudTrail to track a history of API calls made to AWS and store that information in log files. This helps you determine which users and accounts accessed resources in your account, when the calls were made, what actions were requested, and more.

General resources

Explore the following resources to learn more about IAM and AWS.

- [Product Information for IAM](#) – General information about the AWS Identity and Access Management product.
- [Discussion Forums for AWS Identity and Access Management](#) – A community forum for customers to discuss technical questions related to IAM.
- [Classes & Workshops](#) – Links to role-based and specialty courses as well as self-paced labs to help sharpen your AWS skills and gain practical experience.
- [AWS Developer Tools](#) – Links to developer tools, SDKs, IDE toolkits, and command line tools for developing and managing AWS applications.
- [AWS Whitepapers](#) – Links to a comprehensive list of technical AWS whitepapers, covering topics such as architecture, security, and economics and authored by AWS Solutions Architects or other technical experts.
- [AWS Support Center](#) – The hub for creating and managing your AWS Support cases. Also includes links to other helpful resources, such as forums, technical FAQs, service health status, and AWS Trusted Advisor.

- **AWS Support** – The primary webpage for information about AWS Support, a one-on-one, fast-response support channel to help you build and run applications in the cloud.
- **Contact Us** – A central contact point for inquiries concerning AWS billing, account, events, abuse, and other issues.
- **AWS Site Terms** – Detailed information about our copyright and trademark; your account, license, and site access; and other topics.

Calling the IAM API using HTTP query requests

Contents

- [Endpoints \(p. 724\)](#)
- [HTTPS required \(p. 725\)](#)
- [Signing IAM API requests \(p. 725\)](#)

You can access the IAM and AWS STS services programmatically using the Query API. Query API requests are HTTPS requests that must contain an `Action` parameter to indicate the action to be performed. IAM and AWS STS support GET and POST requests for all actions. That is, the API does not require you to use GET for some actions and POST for others. However, GET requests are subject to the limitation size of a URL; although this limit is browser dependent, a typical limit is 2048 bytes. Therefore, for Query API requests that require larger sizes, you must use a POST request.

The response is an XML document. For details about the response, see the individual action pages in the [IAM API Reference](#) or the [AWS Security Token Service API Reference](#).

Tip

Instead of making direct calls to the IAM or AWS STS API operations, you can use one of the AWS SDKs. The AWS SDKs consist of libraries and sample code for various programming languages and platforms (Java, Ruby, .NET, iOS, Android, etc.). The SDKs provide a convenient way to create programmatic access to IAM and AWS. For example, the SDKs take care of tasks such as cryptographically signing requests (see below), managing errors, and retrying requests automatically. For information about the AWS SDKs, including how to download and install them, see the [Tools for Amazon Web Services](#) page.

For details about the API actions and errors, see the [IAM API Reference](#) or the [AWS Security Token Service API Reference](#).

Endpoints

IAM and AWS STS each have a single global endpoint:

- (IAM) <https://iam.amazonaws.com>
- (AWS STS) <https://sts.amazonaws.com>

Note

AWS STS also supports sending requests to regional endpoints in addition to the global endpoint. Before you can use AWS STS in a Region, you must first activate STS in that Region for your AWS account. For more information about activating additional Regions for AWS STS, see [Managing AWS STS in an AWS Region \(p. 327\)](#).

For more information about AWS endpoints and Regions for all services, see [Service endpoints and quotas](#) in the [AWS General Reference](#).

HTTPS required

Because the Query API returns sensitive information such as security credentials, you must use HTTPS with all API requests.

Signing IAM API requests

Requests must be signed using an access key ID and a secret access key. We strongly recommend that you do not use your AWS account root user credentials for everyday work with IAM. You can use the credentials for an IAM user or you can use AWS STS to generate temporary security credentials.

To sign your API requests, we recommend using AWS Signature Version 4. For information about using Signature Version 4, go to [Signature Version 4 Signing Process](#) in the *AWS General Reference*.

If you need to use Signature Version 2, information about using Signature Version 2 is available in the [AWS General Reference](#).

For more information, see the following:

- [AWS Security Credentials](#). Provides general information about the types of credentials used for accessing AWS.
- [Security best practices in IAM \(p. 527\)](#). Presents a list of suggestions for using IAM service to help secure your AWS resources.
- [Temporary security credentials in IAM \(p. 301\)](#). Describes how to create and use temporary security credentials.

Document history for IAM

The following table describes major documentation updates for IAM.

update-history-change	update-history-description	update-history-date
Default password policy for IAM users	If you do not set a custom password policy for your AWS account, IAM user passwords must now meet the default AWS password policy.	November 18, 2020
The actions, resources, and condition keys pages for AWS services have moved	Each AWS service can define actions, resources, and condition context keys for use in IAM policies. You can now find the list of AWS services and their actions, resources, and condition context keys in the <i>Service Authorization Reference</i> .	November 16, 2020
IAM users longer role session duration	IAM users can now have a longer role session duration when switching roles in the AWS Management Console, reducing interruptions due to session expiration. Users are granted the maximum session duration set for the role, or the remaining time in the IAM user's session, whichever is less.	July 24, 2020
Use Service Quotas to request quick increases for IAM entities	You can request quota increases for adjustable IAM quotas using the Service Quotas console. Now, some increases are automatically approved in Service Quotas and available in your account within a few minutes. Larger requests are submitted to AWS Support.	June 25, 2020
Last accessed information in IAM now includes Amazon S3 management actions	In addition to service last accessed information, you can now view information in the IAM console about the last time an IAM principal used an Amazon S3 action. You can also use the AWS CLI or AWS API to retrieve the data report. The report includes information about the allowed services and actions that principals last attempted to access and when. You can use this information to identify	June 3, 2020

	unnecessary permissions so that you can refine your IAM policies to better adhere to the principle of least privilege.	
Security chapter addition	The security chapter helps you understand how to configure IAM and AWS STS to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your IAM resources.	April 29, 2020
sts:RoleSessionName	You can now write a policy that grants permissions based on the session name that a principal specifies when assuming a role.	April 21, 2020
AWS sign-in page update	When you sign in on the main AWS sign-in page, you can no choose to sign in as the AWS account root user or an IAM user. When you do, the label on the page indicates whether you should provide your root user email address or your IAM user account information. This documentation includes updated screen captures to help you understand the AWS sign-in pages.	March 4, 2020
aws:ViaAWSService and aws:CalledVia condition keys	You can now write a policy to limit whether services can make requests on behalf of an IAM principal (user or role). When a principal makes a request to an AWS service, that service might use the principal's credentials to make subsequent requests to other services. Use the aws:ViaAWSService condition key to match if any service makes a request using a principal's credentials. Use the aws:CalledVia condition keys to match if specific services make a request using a principal's credentials.	February 20, 2020
Policy simulator adds support for permissions boundaries	You can now test the effect of permissions boundaries on IAM entities with the IAM policy simulator.	January 23, 2020

Cross-account policy evaluation	You can now learn how AWS evaluates policies for cross-account access. This occurs when a resource in a trusting account includes a resource-based policy that allows a principal in another account to access the resource. The request must be allowed in both accounts.	January 2, 2020
Session tags	You can now include tags when you assume a role or federate a user in AWS STS. When you perform the <code>AssumeRole</code> or <code>GetFederationToken</code> operation, you can pass the session tags as attributes. When you perform the <code>AssumeRoleWithSAML</code> or <code>AssumeRoleWithWebIdentity</code> operations, you can pass attributes from your corporate identities to AWS.	November 22, 2019
Control access for groups of AWS accounts in AWS Organizations	You can now reference organizational units (OUs) from AWS Organizations in IAM policies. If you use Organizations to organize your accounts into OUs, you can require that principals belong to a specific OU before granting access to your resources. Principals include AWS account root user, IAM users and IAM roles. To do this, specify the OU path in the <code>aws:PrincipalOrgPaths</code> condition key in your policies.	November 20, 2019
Role last used	You can now view the date, time, and Region where a role was last used. This information also helps you identify unused roles in your account. You can use the AWS Management Console, AWS CLI and AWS API to view information about when a role was last used.	November 19, 2019

Update to the global condition context keys page	You can now learn when each of the global condition keys is included in the context of a request. You can also navigate to each key more easily using the page table of contents (TOC). The information on the page helps you to write more accurate policies. For example, if your employees use federation with IAM roles, you should use the <code>aws:userId</code> key and not the <code>aws:userName</code> key. The <code>aws:userName</code> key applies only to IAM users and not roles.	October 6, 2019
ABAC in AWS	Learn how attribute-based access control (ABAC) works in AWS using tags, and how it compares to the traditional AWS authorization model. Use the ABAC tutorial to learn how to create and test a policy that allows IAM roles with principal tags to access resources with matching tags. This strategy allows individuals to view or edit only the AWS resources required for their jobs.	October 3, 2019
AWS STS GetAccessKeyInfo operation	You can review the AWS access keys in your code to determine whether the keys are from an account that you own. You can pass an access key ID using the <code>aws sts get-access-key-info</code> AWS CLI command or the <code>GetAccessKeyInfo</code> AWS API operation.	July 24, 2019

Viewing Organizations service last accessed information in IAM	You can now view service last accessed information for an AWS Organizations entity or policy in the AWS Organizations section of the IAM console. You can also use the AWS CLI or AWS API to retrieve the data report. This data includes information about the allowed services that principals in an Organizations account last attempted to access and when. You can use this information to identify unnecessary permissions so that you can refine your Organizations policies to better adhere to the principle of least privilege.	June 20, 2019
Using a managed policy as a session policy	You can now pass up to 10 managed policy ARNs when you assume a role. This allows you to limit the permissions of the role's temporary credentials.	May 7, 2019
AWS STS Region compatibility of session tokens for the global endpoint	You can now choose whether to use version 1 or version 2 global endpoint tokens. Version 1 tokens are valid only in AWS Regions that are available by default. These tokens will not work in manually enabled Regions, such as Asia Pacific (Hong Kong). Version 2 tokens are valid in all Regions. However, version 2 tokens are longer and might affect systems where you temporarily store tokens.	April 26, 2019
Allow enabling and disabling AWS regions	You can now create a policy that allows an administrator to enable and disable the Asia Pacific (Hong Kong) Region (ap-east-1).	April 24, 2019
IAM user my security credentials page	IAM users can now manage all of their own credentials on the My Security Credentials page. This AWS Management Console page displays account information such as the account ID and canonical user ID. Users can also view and edit their own passwords, access keys, X.509 certificates, SSH keys, and Git credentials.	January 24, 2019

Access advisor API	You can now use the AWS CLI and AWS API to view service last accessed information.	December 7, 2018
Tagging IAM users and roles	You can now use IAM tags to add custom attributes to an identity (IAM user or role) using a tag key-value pair. You can also use tags to control an identity's access to resources or to control what tags can be attached to an identity.	November 14, 2018
U2F security keys	You can now use U2F security keys as a multi-factor authentication (MFA) option when signing in to the AWS Management Console.	September 25, 2018
Support for Amazon VPC endpoints	You can now establish a private connection between your VPC and AWS STS in the US West (Oregon) Region.	July 31, 2018
Permissions boundaries	New feature makes it easier to grant trusted employees the ability to manage IAM permissions without also granting full IAM administrative access.	July 12, 2018
aws:PrincipalOrgID	New condition key provides an easier way to control access to AWS resources by specifying the AWS organization of IAM principals.	May 17, 2018
aws:RequestedRegion	New condition key provides an easier way to use IAM policies to control access to AWS Regions.	April 25, 2018
Increased session duration for IAM roles	An IAM role can now have a session duration of 12 hours.	March 28, 2018
Updated role-creation workflow	New workflow improves the process of creating trust relationships and attaching permissions to roles.	September 8, 2017
AWS account sign-in process	Updated AWS sign-in experience allows both root users and IAM users to use the Sign In to the Console link on the AWS Management Console's home page.	August 25, 2017
Example IAM policies	Documentation update features more than 30 example policies.	August 2, 2017

IAM best practices	Information added to the Users section of the IAM console makes it easier to follow IAM best practices.	July 5, 2017
Auto Scaling resources	Resource-level permissions can control access to and permissions for Auto Scaling resources.	May 16, 2017
Amazon RDS for MySQL and Amazon Aurora databases	Database administrators can associate database users with IAM users and roles and thus manage user access to all AWS resources from a single location.	April 24, 2017
Service-linked roles	Service-linked roles provide an easier and more secure way to delegate permissions to AWS services.	April 19, 2017
Policy summaries	New policy summaries make it easier to understand permissions in IAM policies.	March 23, 2017