
AWS Identity and Access Management

User Guide



AWS Identity and Access Management: User Guide

Copyright © 2023 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
When do I use IAM	3
When you are performing different job functions	3
When you are authorized to access AWS resources	3
When you sign-in as an IAM user	4
When you assume an IAM role	4
When you create policies and permissions	5
How IAM works	6
Terms	7
Principal	7
Request	8
Authentication	8
Authorization	8
Actions or operations	9
Resources	9
Users in AWS	9
First-time access only: Your root user credentials	10
IAM users and users in IAM Identity Center	10
Federating existing users	10
Access control methods	11
Permissions and policies in IAM	13
Policies and accounts	13
Policies and users	13
Policies and groups	14
Federated users and roles	14
Identity-based and resource-based policies	15
What is ABAC?	15
Comparing ABAC to the traditional RBAC model	16
Security features outside IAM	17
Quick links to common tasks	18
IAM console search	19
Using IAM console search	20
Icons in the IAM console search results	20
Sample search phrases	21
Working with AWS SDKs	21
Getting set up	23
Sign up for an AWS account	23
Create an administrative user	24
Prepare for least-privilege permissions	24
Your AWS account ID and its alias	25
View your AWS account ID	25
About account aliases	26
Creating, deleting, and listing an AWS account alias	27
Getting started	30
Prerequisites	30
Create your first IAM user	30
Create your first role	31
Create your first IAM policy	33
Getting access keys for programmatic access	33
Tutorials	35
Grant access to the billing console	35

Prerequisites	36
Step 1: Activate access to billing data on your AWS test account	36
Step 2: Sign-in as the root user on your AWS test account and create test users and groups	36
Step 3: Create a role to grant access to the AWS Billing console	38
Step 4: Test access to the console	38
Summary	39
Related resources	39
Delegate access across AWS accounts using roles	40
Prerequisites	41
Create a role in the Production Account	41
Grant access to the role	44
Test access by switching roles	45
Related resources	48
Summary	48
Create a customer managed policy	49
Prerequisites	49
Step 1: Create the policy	49
Step 2: Attach the policy	50
Step 3: Test user access	50
Related resources	51
Summary	51
Use attribute-based access control (ABAC)	51
Tutorial overview	51
Prerequisites	52
Step 1: Create test users	53
Step 2: Create the ABAC policy	54
Step 3: Create roles	57
Step 4: Test creating secrets	57
Step 5: Test viewing secrets	59
Step 6: Test scalability	60
Step 7: Test updating and deleting secrets	61
Summary	62
Related resources	63
Use SAML session tags for ABAC	63
Permit users to manage their credentials and MFA settings	66
Prerequisites	67
Step 1: Create a policy to enforce MFA sign-in	67
Step 2: Attach policies to your test user group	68
Step 3: Test your user's access	68
Related resources	69
Identities	70
AWS account root user	70
IAM users	70
IAM user groups	71
IAM roles	71
Temporary credentials in IAM	72
When to use IAM Identity Center users?	72
When to create an IAM user (instead of a role)	72
When to create an IAM role (instead of a user)	73
Users	74
How AWS identifies an IAM user	74
IAM users and credentials	74
IAM users and permissions	75
IAM users and accounts	76
IAM 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	88
Managing passwords	93
Access keys	103
Retrieving lost passwords or access keys	112
Multi-factor authentication (MFA)	114
Finding unused credentials	161
Getting credential reports	164
Using IAM with CodeCommit	168
Using IAM with Amazon Keyspaces	170
Managing server certificates	171
User groups	175
Creating user groups	177
Managing user groups	178
Roles	183
Terms and concepts	184
Common scenarios	187
Identity providers and federation	198
Service-linked roles	242
Creating roles	250
Using roles	274
Managing roles	384
Tagging IAM resources	399
Choose an AWS tag naming convention	400
Rules for tagging in IAM and AWS STS	400
Tagging IAM users	402
Tagging IAM roles	405
Tagging customer managed policies	407
Tagging IAM identity providers	409
Tagging instance profiles	412
Tagging server certificates	414
Tagging virtual MFA devices	416
Session tags	417
Temporary security credentials	426
AWS STS and AWS regions	426
Common scenarios for temporary credentials	426
Requesting temporary security credentials	428
Using temporary credentials with AWS resources	438
Controlling permissions for temporary security credentials	441
Managing AWS STS in an AWS Region	461
Using AWS STS interface VPC endpoints	465
Using bearer tokens	467
Sample applications that use temporary credentials	467
Additional resources for temporary credentials	468
AWS account root user	468
Create or delete an AWS account	469
Enable MFA on the AWS account root user	469
Creating access keys for the root user	470
Deleting access keys for the root user	470
Changing the password for the root user	471
Securing the credentials for the root user	471
Transferring the root user owner	471
Log events with CloudTrail	471
IAM and AWS STS information in CloudTrail	472
Logging IAM and AWS STS API requests	472
Logging API requests to other AWS services	472

Logging user sign-in events	473
Logging sign-in events for temporary credentials	473
Example IAM API events in CloudTrail log	474
Example AWS STS API events in CloudTrail log	475
Example sign-in events in CloudTrail log	481
Access management	484
Access management resources	485
Policies and permissions	485
Policy types	485
Policies and the root user	490
Overview of JSON policies	490
Grant least privilege	493
Managed policies and inline policies	494
Permissions boundaries	501
Identity vs resource	511
Controlling access using policies	513
Control access to IAM users and roles using tags	521
Control access to AWS resources using tags	523
Cross account resource access	526
Example policies	529
Managing IAM policies	581
Creating IAM policies	581
Validating policies	588
Generating policies	589
Testing IAM policies	589
Add or remove identity permissions	598
Versioning IAM policies	606
Editing IAM policies	609
Deleting IAM policies	613
Refining permissions using access information	616
Understanding policies	634
Policy summary (list of services)	635
Service summary (list of actions)	645
Action summary (list of resources)	649
Example policy summaries	652
Permissions required	662
Permissions for administering IAM identities	662
Permissions for working in the AWS Management Console	663
Granting permissions across AWS accounts	664
Permissions for one service to access another	664
Required actions	664
Example policies for IAM	665
Code examples	668
IAM examples	670
Actions	677
Scenarios	874
AWS STS examples	997
Actions	998
Scenarios	1008
Security	1019
AWS security credentials	1019
Security considerations	1020
Federated identity	1021
Multi-factor authentication (MFA)	1021
Programmatic access	1021
Alternatives for long-term access keys	1022
Accessing AWS using your AWS credentials	1023

AWS security audit guidelines	1023
When to perform a security audit	1024
Guidelines for auditing	1024
Review your AWS account credentials	1024
Review your IAM users	1025
Review your IAM groups	1025
Review your IAM roles	1025
Review your IAM providers for SAML and OpenID Connect (OIDC)	1025
Review Your mobile apps	1026
Tips for reviewing IAM policies	1026
Data protection	1027
Data encryption in IAM and AWS STS	1028
Key management in IAM and AWS STS	1028
Internetwork traffic privacy in IAM and AWS STS	1028
Logging and monitoring	1028
Compliance validation	1029
Resilience	1030
Best practices for IAM resilience	1031
Infrastructure security	1031
Configuration and vulnerability analysis	1032
Security best practices and use cases	1032
Security best practices	1032
Business use cases	1037
AWS managed policies	1040
IAMReadOnlyAccess	1040
IAMUserChangePassword	1040
IAMAccessAnalyzerFullAccess	1041
IAMAccessAnalyzerReadOnlyAccess	1042
AccessAnalyzerServiceRolePolicy	1042
.....	1044
Policy updates	1044
IAM Access Analyzer	1046
Identifying resources shared with an external entity	1046
Validating policies	1047
Generating policies	1047
Findings for public and cross-account access	1047
How IAM Access Analyzer findings work	1048
Getting started with IAM Access Analyzer findings	1049
Working with findings	1051
Reviewing findings	1051
Filtering findings	1053
Archiving findings	1055
Resolving findings	1055
Supported resource types	1056
Settings	1060
Archive rules	1061
Monitoring with EventBridge	1062
Security Hub integration	1067
Logging with CloudTrail	1070
IAM Access Analyzer filter keys	1072
Using service-linked roles	1075
Preview access	1077
Previewing access in Amazon S3 console	1077
Previewing access with IAM Access Analyzer APIs	1078
IAM Access Analyzer policy validation	1080
Validating policies in IAM (console)	1080
Validating policies using Access Analyzer (AWS CLI or AWS API)	1081

Policy check reference	1082
IAM Access Analyzer policy generation	1154
How policy generation works	1154
Service and action-level information	1154
Things to know	1155
Permissions required	1155
Generate a policy based on CloudTrail activity (console)	1157
Generate a policy using AWS CloudTrail data in another account	1160
Generate a policy based on CloudTrail activity (AWS CLI)	1162
Generate a policy based on CloudTrail activity (AWS API)	1162
IAM Access Analyzer policy generation and action last accessed support	1163
IAM Access Analyzer quotas	1169
Troubleshooting IAM	1171
General issues	1171
I can't sign in to my AWS account	1171
I lost my access keys	1171
Policy variables aren't working	1172
Changes that I make are not always immediately visible	1172
I am not authorized to perform: iam>DeleteVirtualMFADevice	1172
How do I securely create IAM users?	1173
Additional resources	1173
Access denied error messages	1174
I get "access denied" when I make a request to an AWS service	1174
I get "access denied" when I make a request with temporary security credentials	1175
Access denied examples	1176
IAM policies	1180
Troubleshoot using the visual editor	1181
Troubleshoot using policy summaries	1184
Troubleshoot policy management	1189
Troubleshoot JSON policy documents	1190
FIDO security keys	1194
I can't enable my FIDO security key	1194
I can't sign in using my FIDO security key	1195
I lost or broke my FIDO security key	1195
Other issues	1195
IAM roles	1195
I can't assume a role	1196
A new role appeared in my AWS account	1197
I can't edit or delete a role in my AWS account	1197
I'm not authorized to perform: iam:PassRole	1198
Why can't I assume a role with a 12-hour session? (AWS CLI, AWS API)	1198
I receive an error when I try to switch roles in the IAM console	1198
My role has a policy that allows me to perform an action, but I get "access denied"	1199
The service did not create the role's default policy version	1199
There is no use case for a service role in the console	1200
IAM and Amazon EC2	1201
When attempting to launch an instance, I don't see the role I expected to see in the Amazon EC2 console IAM Role list	1201
The credentials on my instance are for the wrong role	1202
When I attempt to call the AddRoleToInstanceProfile, I get an AccessDenied error	1202
Amazon EC2: When I attempt to launch an instance with a role, I get an AccessDenied error	1202
I can't access the temporary security credentials on my EC2 instance	1202
What do the errors from the info document in the IAM subtree mean?	1203
IAM and Amazon S3	1204
How do I grant anonymous access to an Amazon S3 bucket?	1204
I'm signed in as an AWS account root user; why can't I access an Amazon S3 bucket under my account?	1204

SAML 2.0 federation	1204
Invalid SAML response	1205
RoleSessionName is required	1205
Not authorized for AssumeRoleWithSAML	1206
Invalid RoleSessionName characters	1206
Invalid Source Identity characters	1207
Invalid response signature	1207
Failed to assume role	1207
Could not parse metadata	1207
Specified provider doesn't exist	1208
DurationSeconds exceeds MaxSessionDuration	1208
Response does not contain the required audience	1208
Viewing a SAML response in your browser	1208
Reference	1211
Amazon Resource Names (ARNs)	1211
ARN format	1211
Look up the ARN format for a resource	1212
Paths in ARNs	1212
IAM identifiers	1213
Friendly names and paths	1213
IAM ARNs	1213
Unique identifiers	1218
IAM and AWS STS quotas	1220
IAM name requirements	1220
IAM object quotas	1220
IAM Access Analyzer quotas	1221
IAM Roles Anywhere quotas	1221
IAM and STS character limits	1221
Services that work with IAM	1224
Services that work with IAM	1225
More information	1242
Signing AWS API requests	1244
When to sign requests	1246
Why requests are signed	1246
Signature Version 4 request elements	1246
Authentication methods	1247
Create a signed request	1250
Request signature examples	1258
Troubleshoot	1258
Policy reference	1260
JSON element reference	1260
Policy evaluation logic	1306
Policy grammar	1323
AWS managed policies for job functions	1329
Global condition keys	1338
IAM condition keys	1367
Actions, resources, and condition keys	1383
Resources	1384
Identities	1384
Credentials (passwords, access keys, and MFA devices)	1384
Permissions and policies	1384
Federation and delegation	1385
IAM and other AWS products	1385
Using IAM with Amazon EC2	1385
Using IAM with Amazon S3	1385
Using IAM with Amazon RDS	1386
Using IAM with Amazon DynamoDB	1386

General security practices	1386
General resources	1386
Making HTTP query requests	1388
Endpoints	1388
HTTPS required	1389
Signing IAM API requests	1389
Document history	1390

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. With IAM, you can centrally manage permissions that control which AWS resources users can access. You use IAM to control who is authenticated (signed in) and authorized (has permissions) to use resources.

When you create an AWS account, you begin with one 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 don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see [Tasks that require root user credentials](#) in the [AWS Account Management Reference Guide](#).

Contents

- [Video introduction to IAM \(p. 1\)](#)
- [IAM features \(p. 1\)](#)
- [Accessing IAM \(p. 2\)](#)
- [When do I use IAM? \(p. 3\)](#)
- [How IAM works \(p. 6\)](#)
- [Overview of AWS identity management: Users \(p. 9\)](#)
- [Overview of access management: Permissions and policies \(p. 13\)](#)
- [What is ABAC for AWS? \(p. 15\)](#)
- [Security features outside IAM \(p. 17\)](#)
- [Quick links to common tasks \(p. 18\)](#)
- [IAM console search \(p. 19\)](#)
- [Using IAM with an AWS SDK \(p. 21\)](#)

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. If you already use a FIDO security key with other services, and it has an AWS supported configuration, you can use WebAuthn for MFA security. For more information, see [Supported configurations for using FIDO security keys \(p. 125\)](#).

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 \(p. 1224\)](#).

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 \(p. 1172\)](#).

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 [How to sign in to AWS](#) in the *AWS Sign-In User Guide*.

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 Query API

You can access IAM and AWS programmatically by using the IAM Query API, which lets you issue HTTPS requests directly to the service. When you use the Query API, you must include code to digitally sign requests using your credentials. For more information, see [Calling the IAM API using HTTP query requests \(p. 1388\)](#) and the [IAM API Reference](#).

When do I use IAM?

When you are performing different job functions

AWS Identity and Access Management is a core infrastructure service that provides the foundation for access control based on identities within AWS. You use IAM every time you access your AWS account.

How you use IAM differs, depending on the work that you do in AWS.

- Service user – If you use an AWS service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more advanced features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator.
- Service administrator – If you're in charge of an AWS resource at your company, you probably have full access to IAM. It's your job to determine which IAM features and resources your service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM.
- IAM administrator – If you're an IAM administrator, you manage IAM identities and write policies to manage access to IAM.

When you are authorized to access AWS resources

Authentication is how you sign in to AWS using your identity credentials. You must be *authenticated* (signed in to AWS) as the AWS account root user, as an IAM user, or by assuming an IAM role.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (successor to AWS Single Sign-On) (IAM Identity Center) users, your company's single sign-on authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see [How to sign in to your AWS account](#) in the *AWS Sign-In User Guide*.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see [Signing AWS API requests](#) in the *IAM User Guide*.

Regardless of the authentication method that you use, you might 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, see [Multi-factor authentication](#) in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide* and [Using multi-factor authentication \(MFA\) in AWS](#) in the *IAM User Guide*.

When you sign-in as an IAM user

An [IAM user](#) is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see [Rotate access keys regularly for use cases that require long-term credentials](#) in the *IAM User Guide*.

An [IAM group](#) is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see [When to create an IAM user \(instead of a role\)](#) in the *IAM User Guide*.

When you assume an IAM role

An [IAM role](#) is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by [switching roles](#). You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see [Using IAM roles](#) in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see [Creating a role for a third-party Identity Provider](#) in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see [Permission sets](#) in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*.

- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see [How IAM roles differ from resource-based policies](#) in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
- **Principal permissions** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see [Actions, resources, and condition keys for AWS Identity and Access Management](#) in the *Service Authorization Reference*.
- **Service role** – A service role is an [IAM role](#) that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see [Creating a role to delegate permissions to an AWS service](#) in the *IAM User Guide*.
- **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see [Using an IAM role to grant permissions to applications running on Amazon EC2 instances](#) in the *IAM User Guide*.

To learn whether to use IAM roles or IAM users, see [When to create an IAM role \(instead of a user\)](#) in the *IAM User Guide*.

When you create policies and permissions

You grant permissions to a user by creating a policy, which is a document that lists the actions that a user can perform and the resources those actions can affect. Any actions or resources that are not explicitly allowed are denied by default. Policies can be created and attached to principals (users, groups of users, roles assumed by users, and resources).

These policies are used with an IAM role:

- **Trust policy** – Defines which [principals \(p. 7\)](#) can assume the role, and under which conditions. A trust policy is a specific type of resource-based policy for IAM roles. A role can have only one trust policy.
- **Identity-based policies (inline and managed)** – These policies define the permissions that the user of the role is able to perform (or is denied from performing), and on which resources.

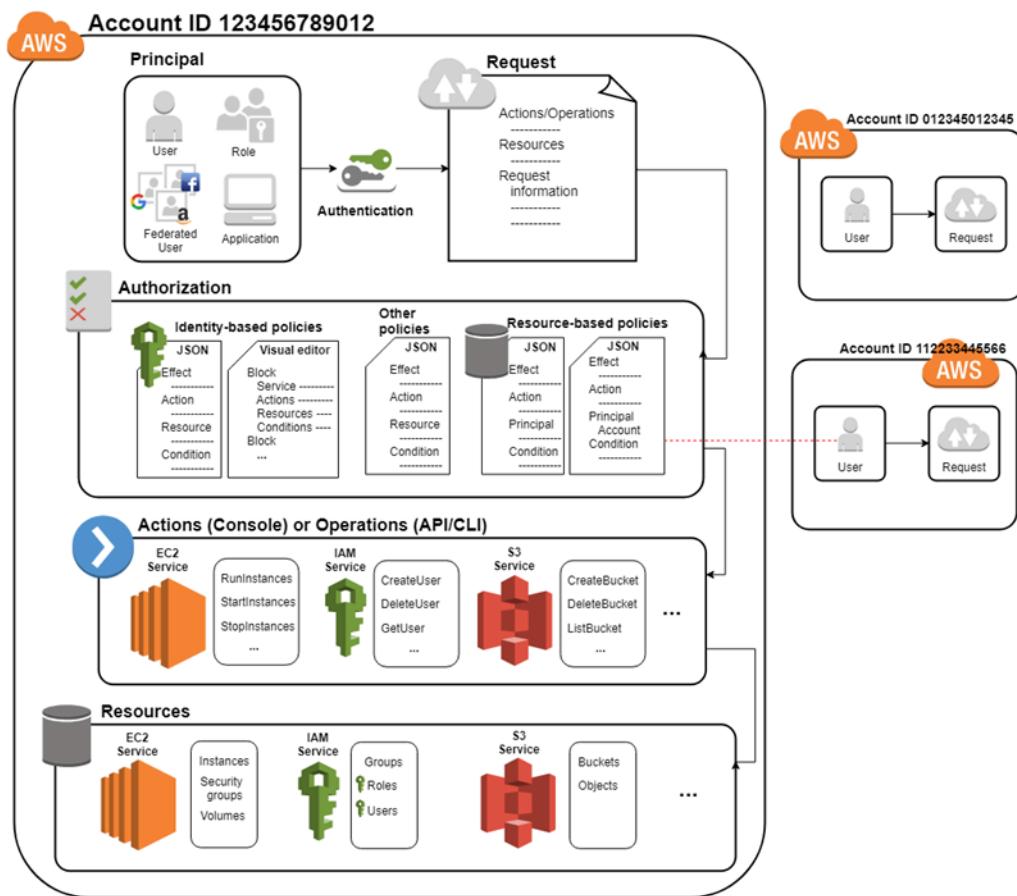
Use the [Example IAM identity-based policies \(p. 529\)](#) to help you define permissions for your IAM identities. After you find the policy that you need, choose *View policy* to view the JSON for the policy. You can use the JSON policy document as a template for your own policies.

Note

If you are using IAM Identity Center to manage your users, you assign permission sets in IAM Identity Center instead of attaching a permissions policy to a principal. When you assign a permission set to a group or user in AWS IAM Identity Center (successor to AWS Single Sign-On), IAM Identity Center creates corresponding IAM roles in each account, and attaches the policies specified in the permission set to those roles. IAM Identity Center manages the role, and allows the authorized users you've defined to assume the role. If you modify the permission set, IAM Identity Center ensures that the corresponding IAM policies and roles are updated accordingly. For more information about IAM Identity Center, see [What is IAM Identity Center?](#) in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*.

How IAM works

IAM provides the infrastructure necessary to control authentication and authorization for your AWS account. The IAM infrastructure is illustrated by the following diagram:



First, a human user or an application uses their sign-in credentials to authenticate with AWS. Authentication is provided by matching the sign-in credentials to a principal (an IAM user, federated user, IAM role, or application) trusted by the AWS account.

Next, a request is made to grant the principal access to resources. Access is granted in response to an authorization request. For example, when you first sign in to the console and are on the console Home page, you are not accessing a specific service. When you select a service, the request for authorization is sent to that service and it looks to see if your identity is on the list of authorized users, what policies

are being enforced to control the level of access granted, and any other policies that might be in effect. Authorization requests can be made by principals within your AWS account or from another AWS account that you trust.

Once authorized, the principal can take action or perform operations on resources in your AWS account. For example, the principal could launch a new Amazon Elastic Compute Cloud instance, modify IAM group membership, or delete Amazon Simple Storage Service buckets.

Basic concepts

- [Terms \(p. 7\)](#)
- [Principal \(p. 7\)](#)
- [Request \(p. 8\)](#)
- [Authentication \(p. 8\)](#)
- [Authorization \(p. 8\)](#)
- [Actions or operations \(p. 9\)](#)
- [Resources \(p. 9\)](#)

Terms

In the previous illustration we used specific terminology to describe how to obtain access to resources. These IAM terms are commonly used when working with AWS:

IAM 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.

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.

IAM Entities

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

Human users

Also known as *human identities*; the people, administrators, developers, operators, and consumers of your applications.

Workload

A collection of resources and code that delivers business value, such as an application or backend process. Can include applications, operational tools, and components.

Principal

A *principal* is a human user or workload that can make a request for an action or operation on an AWS resource. After authentication, the principal can be granted either permanent or temporary credentials to make requests to AWS, depending on the principal type. IAM users and root user are granted

permanent credentials, while roles are granted temporary credentials. As a [best practice \(p. 1032\)](#), we recommend that you require human users and workloads to access AWS resources using temporary credentials.

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 a federated user, you are authenticated by your identity provider and granted access to AWS resources by assuming IAM roles. As an IAM user, provide your account ID or alias, and then your user name and password. To authenticate workloads from the API or AWS CLI, you might use temporary credentials through being assigned a role or you might use long-term credentials by providing your access key and secret key. You might also be required to provide additional security information. As a best practice, AWS recommends that you use multi-factor authentication (MFA) and temporary credentials 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 \(p. 183\)](#).

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 \(p. 490\)](#) and specify the permissions for principal entities. There are [several types of policies \(p. 485\)](#) 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 \(p. 664\)](#). 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 \(p. 1306\)](#). 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

You can give access to your AWS account to specific users and provide them specific permissions to access resources in your AWS account. You can use both IAM and AWS IAM Identity Center (successor to AWS Single Sign-On) to create new users or federate existing users into AWS. The main difference between the two is that IAM users are granted long-term credentials to your AWS resources while users in IAM Identity Center have temporary credentials that are established each time the user signs-in to AWS. As a [best practice \(p. 1032\)](#), require human users to use federation with an identity provider to

access AWS using temporary credentials instead of as an IAM user. A primary use for IAM users is to give workloads that cannot use IAM roles the ability to make programmatic requests to AWS services using the API or CLI.

Topics

- [First-time access only: Your root user credentials \(p. 10\)](#)
- [IAM users and users in IAM Identity Center \(p. 10\)](#)
- [Federating existing users \(p. 10\)](#)
- [Access control methods \(p. 11\)](#)

First-time access only: Your root user credentials

When you create an AWS account, you begin with one 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 don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see [Tasks that require root user credentials](#) in the *AWS Account Management Reference Guide*. Only service control policies (SCPs) in organizations can restrict the permissions that are granted to the root user.

IAM users and users in IAM Identity Center

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.

IAM users are granted long-term credentials to your AWS resources. As a best practice, do not create IAM users with long-term credentials for your human users. Instead, require your human users to use temporary credentials when accessing AWS.

Note

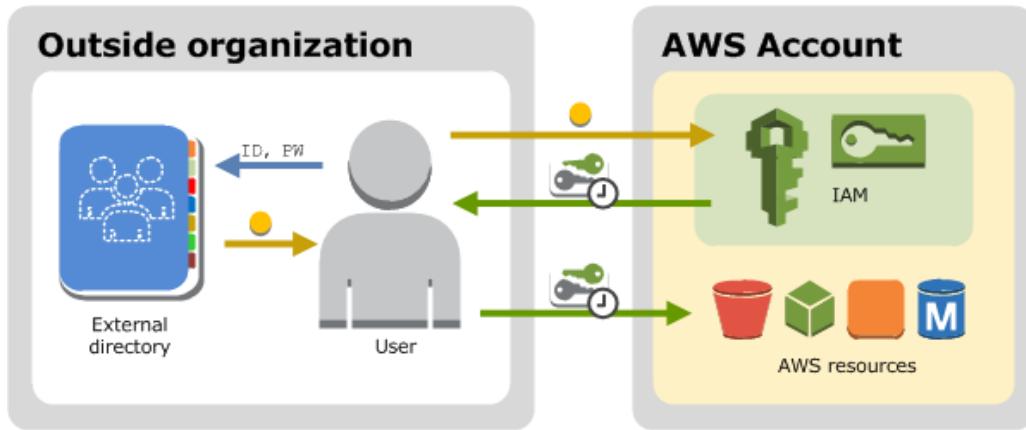
For scenarios in which you need IAM users with programmatic access and long-term credentials, we recommend that you rotate access keys. For more information, see [Rotating access keys \(p. 108\)](#).

In contrast, users in AWS IAM Identity Center (successor to AWS Single Sign-On) are granted short-term credentials to your AWS resources. For centralized access management, we recommend that you use [AWS IAM Identity Center \(successor to AWS Single Sign-On\) \(IAM Identity Center\)](#) to manage access to your accounts and permissions within those accounts. IAM Identity Center is automatically configured with an Identity Center directory as your default identity source where you can create users and groups, and assign their level of access to your AWS resources. For more information, see [What is AWS IAM Identity Center \(successor to AWS Single Sign-On\)](#) in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*.

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 or users in IAM Identity Center for them. Instead, you can *federate* those user identities into AWS using either IAM or AWS IAM Identity Center (successor to AWS Single Sign-On).

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



Federation is particularly useful in these cases:

- **Your users already exist 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 \(p. 426\)](#).

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 \(p. 233\)](#).

If your corporate directory is Microsoft Active Directory, you can use AWS IAM Identity Center (successor to AWS Single Sign-On) to connect a self-managed directory in Active Directory or a directory in [AWS Directory Service](#) to establish trust between your corporate directory and your AWS account.

If you are using an external identity provider (IdP) such as Okta or Azure Active Directory to manage users, you can use AWS IAM Identity Center (successor to AWS Single Sign-On) to establish trust between your IdP and your AWS account. For more information, see [Connect to an external identity provider](#) in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*.

- **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 \(p. 199\)](#).

Tip

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

Access control methods

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

Type of user access	Why would I use it?	Where can I get more information?
Single sign-on access for human users, such as your workforce users, to AWS resources using IAM Identity Center	<p>IAM Identity Center provides a central place that brings together administration of users and their access to AWS accounts and cloud applications.</p> <p>You can set up an identity store within IAM Identity Center or you can configure federation with an existing identity provider (IdP). Granting your human users limited credentials to AWS resources as needed is recommended as a security best practice.</p> <p>Users have an easier sign-in experience and you maintain control over their access to resources from a single system. IAM Identity Center supports multi-factor authentication (MFA) for additional account security.</p>	<p>For more information about setting up IAM Identity Center, see Getting Started in the <i>AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide</i></p> <p>For more information about using MFA in IAM Identity Center, see Multi-factor authentication in the <i>AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide</i></p>
Federated access for human users, such as your workforce users, to AWS services using IAM identity providers	IAM supports IdPs that are compatible with OpenID Connect (OIDC) or SAML 2.0 (Security Assertion Markup Language 2.0). After you create an IAM identity provider, you must create one or more IAM roles that can be dynamically assigned to a federated user.	For more information about IAM identity providers and federation, see Identity providers and federation (p. 198) .
Cross-account access between AWS accounts	<p>You want to share access to certain AWS resources with users in other AWS accounts.</p> <p>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.</p>	<p>For more information about IAM roles, see IAM roles (p. 183).</p> <p>For more information about service-linked roles, see Using service-linked roles (p. 242).</p> <p>For information about which services support using service-linked roles, see AWS services that work with IAM (p. 1224). Look for the services that have Yes in the Service-Linked Role column. To view the service-linked role documentation for that service select the link associated with the Yes in that column.</p>
Long-term credentials for designated IAM users in your AWS account	You might have specific use cases that require long-term credentials with IAM users in AWS. You can use IAM to create these IAM users in your AWS account, and use IAM to manage	For more information about setting up an IAM user, see Creating an IAM user in your AWS account (p. 76) .

Type of user access	Why would I use it?	Where can I get more information?
	<p>their permissions. Some of the use cases include the following:</p> <ul style="list-style-type: none"> • Workloads that cannot use IAM roles • Third-party AWS clients • AWS IAM Identity Center (successor to AWS Single Sign-On) is not available for your account and you have no other identity provider <p>As a best practice (p. 1032) in scenarios in which you need IAM users with programmatic access and long-term credentials, we recommend that you rotate access keys. For more information, see Rotating access keys (p. 108).</p>	

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 \(p. 485\)](#).

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.

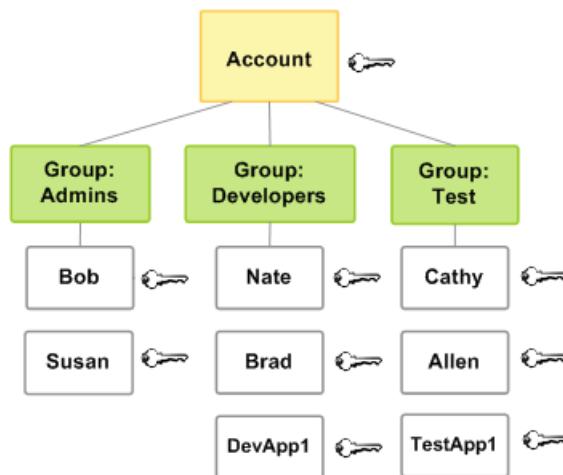
```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "dynamodb:*",
      "Resource": "arn:aws:dynamodb:us-east-2:123456789012:table/Books"
    }
  ]
}
```

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.

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 \(p. 1032\)](#).



Users or groups can have multiple policies attached to them that grant different permissions. In that case, the permissions for the users 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 \(federation\) \(p. 260\)](#).

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, edit, and validate an IAM policy in the visual editor or by creating the JSON policy document directly. For more information, see [Creating IAM policies \(p. 581\)](#) and [Editing IAM policies \(p. 609\)](#).
- **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 [Cross account resource access in IAM \(p. 526\)](#).

To see which services support resource-based policies, see [AWS services that work with IAM \(p. 1224\)](#). To learn more about resource-based policies, see [Identity-based policies and resource-based policies \(p. 511\)](#).

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*. You can attach tags to IAM resources, including IAM entities (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 *Star*, 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 \(p. 51\)](#). To learn about services that support ABAC, see [AWS services that work with IAM \(p. 1224\)](#).

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 \(p. 1329\)](#) 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 \(p. 1032\)](#), you grant the minimum permissions necessary for the job function. This is known as [granting least privilege \(p. 1035\)](#). 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, Star, 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 Star project might require additional resources, such as a new Amazon EC2 container. In that case, you must update the policy attached to the Star role to specify the new container resource. Otherwise, Star project members aren't allowed to access the new container.

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 Star 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 \(p. 1035\)](#). 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 \(p. 51\)](#).

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*

WorkSpaces

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

For more information, see [Getting Started with 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 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 for different user types

Sign in to the [IAM console](#) by choosing **IAM user** and entering your AWS account ID or account alias. On the next page, enter your IAM user name and your password.

To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

For help signing in using an IAM Identity Center user, see [Signing in to the AWS access portal](#) in the [AWS Sign-In User Guide](#).

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.

See [What is AWS Sign-In](#) in the [AWS Sign-In User Guide](#) for help determining your user type and sign-in page.

Manage passwords for 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 [Change the password for the AWS account root user](#) in the [AWS Account Management Reference Guide](#)

For an IAM user, see [Managing passwords for IAM users \(p. 97\)](#).

Manage permissions for 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.

To provide access, add permissions to your users, groups, or roles:

- Users and groups in AWS IAM Identity Center (successor to AWS Single Sign-On):

Create a permission set. Follow the instructions in [Create a permission set](#) in the [AWS IAM Identity Center \(successor to AWS Single Sign-On\) User Guide](#).

- Users managed in IAM through an identity provider:

Create a role for identity federation. Follow the instructions in [Creating a role for a third-party identity provider \(federation\)](#) in the [IAM User Guide](#).

- IAM users:

- Create a role that your user can assume. Follow the instructions in [Creating a role for an IAM user](#) in the [IAM User Guide](#).
- (Not recommended) Attach a policy directly to a user or add a user to a user group. Follow the instructions in [Adding permissions to a user \(console\)](#) in the [IAM User Guide](#).

For more information, see [Managing IAM policies \(p. 581\)](#).

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

See [Getting credential reports for your AWS account \(p. 164\)](#).

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 \(console\) \(p. 118\)](#)
- [Enable a virtual MFA device for an IAM user \(console\) \(p. 117\)](#)

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

- [Enable a FIDO security key for the AWS account root user \(console\) \(p. 123\)](#)
- [Enable a FIDO security key for another IAM user \(console\) \(p. 123\)](#)

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

- [Enable a hardware TOTP token for the AWS account root user \(console\) \(p. 132\)](#).
- [Enable a hardware TOTP token for another IAM user \(console\) \(p. 131\)](#)

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.

Before you start creating access keys, we highly recommend that you read the security best practices for access keys. For more information, see [Best practices for managing AWS access keys](#) in the [AWS General Reference](#).

For your AWS account, see [Programmatic access](#) in the [AWS General Reference](#).

For an IAM user, see [Managing access keys for IAM users \(p. 103\)](#).

Tag IAM resources

You can tag the following IAM resources:

- IAM users
- IAM roles
- Customer managed policies
- Identity providers
- Server certificates
- Virtual MFA devices

To learn about tags in IAM, see [Tagging IAM resources \(p. 399\)](#).

To learn about using tags to control access to AWS resources, see [Controlling access to AWS resources using tags \(p. 523\)](#).

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](#).

IAM console search

Use the IAM console search page as a faster option for finding IAM resources. You can use the console search page to 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)
- 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 the 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. Follow the sign-in procedure appropriate to your user type as described in the topic [How to sign in to AWS](#) in the *AWS Sign-In User Guide*.
2. On the **Console Home** page, select the IAM service.
3. In the navigation pane, choose **Search**.
4. In the **Search** box, type your search keywords.
5. Choose a link in the search results list to navigate to the corresponding part of the console.

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"

Icon	Description
	Results from the keyword delete

Sample search phrases

You can use the following phrases in the IAM search. Replace terms in italics with the names of the actual IAM users, groups, roles, access keys, policies, or identity providers 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 from *policy_name*

Using IAM with an AWS SDK

AWS software development kits (SDKs) are available for many popular programming languages. Each SDK provides an API, code examples, and documentation that make it easier for developers to build applications in their preferred language.

SDK documentation	Code examples
AWS SDK for C++	AWS SDK for C++ code examples
AWS SDK for Go	AWS SDK for Go code examples
AWS SDK for Java	AWS SDK for Java code examples

SDK documentation	Code examples
AWS SDK for JavaScript	AWS SDK for JavaScript code examples
AWS SDK for Kotlin	AWS SDK for Kotlin code examples
AWS SDK for .NET	AWS SDK for .NET code examples
AWS SDK for PHP	AWS SDK for PHP code examples
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) code examples
AWS SDK for Ruby	AWS SDK for Ruby code examples
AWS SDK for Rust	AWS SDK for Rust code examples
AWS SDK for Swift	AWS SDK for Swift code examples

For examples specific to IAM, see [Code examples for IAM using AWS SDKs \(p. 668\)](#).

Example availability

Can't find what you need? Request a code example by using the **Provide feedback** link at the bottom of this page.

Getting set up with IAM

Important

IAM [best practices \(p. 1032\)](#) recommend that you require human users to use federation with an identity provider to access AWS using temporary credentials instead of using IAM users with long-term credentials.

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 sign-in credentials private. You don't specifically sign up to use IAM. There is no charge to use IAM.

Use IAM to give identities, such as users and roles, access to resources in your account. For example, you can use IAM with existing users in your corporate directory that you manage external to AWS or you can create users in AWS using AWS IAM Identity Center (successor to AWS Single Sign-On). Federated identities assume defined IAM roles to access only the resources they need. For more information about IAM Identity Center, see [What is IAM Identity Center?](#) in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*.

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 \(p. 1224\)](#).

Topics

- [Sign up for an AWS account \(p. 23\)](#)
- [Create an administrative user \(p. 24\)](#)
- [Prepare for least-privilege permissions \(p. 24\)](#)
- [Your AWS account ID and its alias \(p. 25\)](#)

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

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.

When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, [assign administrative access to an administrative user](#), and use only the root user to perform [tasks that require root user access](#).

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to <https://aws.amazon.com/> and choosing **My Account**.

Create an administrative user

After you sign up for an AWS account, create an administrative user so that you don't use the root user for everyday tasks.

Secure your 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.

For help signing in by using root user, see [Signing in as the root user](#) in the *AWS Sign-In User Guide*.

2. Turn on multi-factor authentication (MFA) for your root user.

For instructions, see [Enable a virtual MFA device for your AWS account root user \(console\)](#) in the *IAM User Guide*.

Create an administrative user

- For your daily administrative tasks, grant administrative access to an administrative user in AWS IAM Identity Center (successor to AWS Single Sign-On).

For instructions, see [Getting started](#) in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*.

Sign in as the administrative user

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

For help signing in using an IAM Identity Center user, see [Signing in to the AWS access portal](#) in the *AWS Sign-In User Guide*.

Prepare for least-privilege permissions

Using *least-privilege permissions* is an IAM best practice recommendation. The concept of least-privilege permissions is to grant users the permissions required to perform a task and no additional permissions. As you get set up, consider how you are going to support least-privilege permissions. Both the root user and the administrator user have powerful privileges that aren't required for everyday tasks. While you are learning about AWS and testing out different services we recommend that you create at least one additional user in IAM Identity Center with lesser privileges that you can use in different scenarios. You can use IAM policies to define the actions that can be taken on specific resources under specific conditions and then connect to those resources with your lesser privileged account.

If you are using IAM Identity Center, consider using IAM Identity Center permissions sets to get started. To learn more, see [Create a permission set](#) in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*.

If you aren't using IAM Identity Center, use IAM roles to define the privileges for different IAM entities. To learn more, see [Creating IAM roles \(p. 250\)](#).

Both IAM roles and IAM Identity Center permissions sets can use AWS managed policies based on job functions. For details on the permissions granted by these policies, see [AWS managed policies for job functions \(p. 1329\)](#).

Important

Keep in mind that AWS managed policies might not grant least-privilege permissions for your specific use cases because they're available for use by all AWS customers. After getting set up, we recommend that you use IAM Access Analyzer to generate least-privilege policies based on your access activity that's logged in AWS CloudTrail. For more information about policy generation, see [IAM Access Analyzer policy generation](#).

Your AWS account ID and its alias

IAM users in the account sign in using a web URL that includes either the account alias or an account ID. If you don't have the URL, the AWS sign-in page requires that you provide either the AWS account alias or account ID.

If you don't know your account ID or alias:

- Check your browser history. If you have signed in previously, it could be stored in your recent web sites.
- If you have configured the AWS CLI or an AWS SDK with your account credentials, you can obtain your account ID from your configuration files.
- Ask your local administrator or account owner, AWS cannot provide account IDs to users.

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 because that captures information specific to your current browser session that interfere with future visits to the sign-in page.

Topics

- [View your AWS account ID \(p. 25\)](#)
- [About account aliases \(p. 26\)](#)
- [Creating, deleting, and listing an AWS account alias \(p. 27\)](#)

View your AWS account ID

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

View Your Account ID using the console

There are different ways to view your account ID in the console depending on your user type. If you have assumed a role, **My security credentials** is not available.

User type	Procedure
Root user	In the navigation bar at the upper right, choose your user name and then choose My security credentials . The account number appears under Account identifiers .
IAM user	In the navigation bar at the upper right, choose your user name and then choose My security credentials . The account number appears under Account details .

User type	Procedure
Assumed role	In the navigation bar at the upper right, choose Support , and then Support Center . Your currently signed-in 12-digit account number (ID) appears in the Support Center navigation pane.

View 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](#)

View 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/
```

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 contain only digits, lowercase letters, and hyphens. For more information on limitations on AWS account entities, see [IAM and AWS STS quotas \(p. 1220\)](#).
- The account alias must be unique across all Amazon Web Services products within a given network *partition*.

A *partition* is a group of AWS Regions. Each AWS account is scoped to one partition.

The following are the supported partitions:

- aws - AWS Regions
- aws-cn - China Regions
- aws-us-gov - AWS GovCloud (US) Regions

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.

Note

Account aliases are not secrets, and they will appear in your public-facing sign-in page URL. Do not include any sensitive information in your account alias.

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

Create or edit an account alias (console)

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

Minimum permissions

To perform the following steps, you must have at least the following IAM permissions:

- iam>ListAccountAliases
- iam>CreateAccountAlias

To create or edit 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. In the **AWS Account** section, next to **Account Alias**, choose **Create**. If an alias already exists, then choose **Edit**.
4. In the dialog box, enter the name you want to use for your alias, then choose **Save changes**.

Note

You can have only one alias associated with your AWS account at a time. If you create a new alias, the previous alias is removed, and the sign-in URL that was associated with the previous alias stops working.

Delete an account alias (console)

You can delete an account alias from the AWS Management Console.

Minimum permissions

To perform the following steps, you must have at least the following IAM permissions:

- iam>ListAccountAliases
- iam>CreateAccountAlias
- iam>DeleteAccountAlias

To delete 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. In the **AWS Account** section, next to **Account Alias**, choose **Delete**.

Note

The only sign-in URL for your account is based off your account ID. Any attempts to connect to the alias URL are not redirected.

Creating, deleting, and listing aliases (AWS CLI)

Note

To use the following commands, you must have at least the following IAM permissions:

- `iam>ListAccountAliases`
- `iam>CreateAccountAlias`
- `iam>DeleteAccountAlias`

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`](#)

Example Alias commands

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

```
$ aws iam list-account-aliases
{
    "AccountAliases": [
        "myaccountalias"
    ]
}
```

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

```
$ aws iam create-account-alias \
--account-alias myaliasname
```

This command produces no output if it's successful.

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

```
$ aws iam delete-account-alias \
--account-alias myaliasname
```

This command produces no output if it's successful.

Creating, deleting, and listing aliases (AWS API)

Note

To use the following API operations, you must have at least the following IAM permissions:

- `iam>ListAccountAliases`
- `iam>CreateAccountAlias`
- `iam>DeleteAccountAlias`

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](#)

Getting started with IAM

Use this tutorial to get started with AWS Identity and Access Management (IAM). You'll learn how to create roles, users, and policies using the AWS Management Console.

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

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](#).

Contents

- [Prerequisites \(p. 30\)](#)
- [Create your first IAM user \(p. 30\)](#)
- [Create your first role \(p. 31\)](#)
- [Create your first IAM policy \(p. 33\)](#)
- [Getting access keys for programmatic access \(p. 33\)](#)

Prerequisites

Before you begin, be sure that you've completed the steps in [Getting set up with IAM \(p. 23\)](#). This tutorial uses the administrator account you created in that procedure.

Create your first IAM user

An [IAM user](#) is an identity within your AWS account that has specific permissions for a single person or application. Users can be organized into groups that share the same permissions.

Note

As a security best practice, we recommend that you provide access to your resources through identity federation instead of creating IAM users. For information about specific situations where an IAM user is required, see [When to create an IAM user \(instead of a role\)](#) in the *IAM User Guide*.

For the purpose of familiarizing yourself with the process of creating a IAM user, this tutorial will step you through creating an IAM user and group for emergency access.

To create your first IAM user

1. Follow the sign-in procedure appropriate to your user type as described in the topic [How to sign in to AWS](#) in the *AWS Sign-In User Guide*.
2. On the **Console Home** page, select the IAM service.
3. In the navigation pane, select **Users** and then select **Add users**.

Note

If you have IAM Identity Center enabled, the AWS Management Console displays a reminder that it is best to manage users' access in IAM Identity Center. In this tutorial, the IAM user

you create is specifically for use only when your user in IAM Identity Center credentials are unavailable.

4. For **User name**, enter **EmergencyAccess**. Names cannot contain spaces.
5. Select the check box next to **Provide user access to the AWS Management Console—optional** and then choose **I want to create an IAM user**.
6. Under **Console password**, select **Autogenerated password**.
7. Clear the check box next to **User must create a new password at next sign-in (recommended)**. Because this IAM user is for emergency access, a trusted administrator retains the password and only provides it when needed.
8. On the **Set permissions** page, under **Permissions options**, select **Add user to group**. Then, under **User groups**, select **Create group**.
9. On the **Create user group** page, in **User group name**, enter **EmergencyAccessGroup**. Then, under **Permissions policies**, select **AdministratorAccess**.
10. Select **Create user group** to return to the **Set permissions** page.
11. Under **User groups**, select the name of the **EmergencyAccessGroup** you created previously.
12. Select **Next** to proceed to the **Review and create** page.
13. On the **Review and create** page, review the list of user group memberships to be added to the new user. When you are ready to proceed, select **Create user**.
14. On the **Retrieve password** page, select **Download .csv file** to save a .csv file with the user credential information (Connection URL, user name, and password).
15. Save this file to use if you need to sign-in to IAM and do not have access to your federated identity provider.

The new IAM user is displayed in the **Users** list. Select the **User name** link to view the user details. Under **Summary**, copy the **ARN** of the user to the clipboard. Paste the **ARN** into a text document, so that you can use it in the next procedure.

Create your first role

IAM roles are a secure way to grant permissions to entities you trust. An IAM role has some similarities to an IAM user. Roles and users are both principals 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. Using roles helps you follow the IAM best practices. You can use a role to:

- Enable workforce identities and Identity Center enabled applications access to the AWS Management Console using AWS IAM Identity Center (successor to AWS Single Sign-On).
- Delegate permission to an AWS service to carry out actions on your behalf.
- Enable application code running on an Amazon EC2 instance to access or modify AWS resources.
- Grant access to another AWS account.

Note

You can use AWS Identity and Access Management Roles Anywhere to give access to machine identities. Using IAM Roles Anywhere means you don't need to manage long-term credentials for workloads running outside of AWS. For more information, see [What is AWS Identity and Access Management Roles Anywhere?](#) in the *AWS Identity and Access Management Roles Anywhere User Guide*.

IAM Identity Center and other AWS services automatically create roles for their services. If you are using IAM users, we recommend that you create roles for your users to assume when they sign-in. This will give them temporary permissions during the session instead of long-term permissions.

The AWS Management Console wizard that guides you through the steps for creating a role displays slightly different steps depending on whether you're creating a role for an IAM user, AWS service, or for a federated user. Regular access to AWS accounts within an organization should be provided using federated access. If you are creating IAM users for specific purposes, such as emergency access or programmatic access, only grant those IAM users permission to assume a role and put those IAM users into role specific groups.

In this procedure, you create a role that provides **SupportUser** access for the **EmergencyAccess** IAM user. Before starting this procedure, copy the ARN of the IAM user to the clipboard.

To create a role for an IAM user

1. Follow the sign-in procedure appropriate to your user type as described in the topic [How to sign in to AWS in the AWS Sign-In User Guide](#).
2. On the **Console Home** page, select the IAM service.
3. In the navigation pane of the IAM console, choose **Roles** and then choose **Create role**.
4. Choose **AWS account** role type.
5. In **Select trusted entity**, under **Trusted entity type**, choose **Custom trust policy**.
6. In the **Custom trust policy** section, review the basic trust policy. This is the one we will use for this role. Use the **Edit statement** editor to update the trust policy:
 1. In **Add actions for STS**, select **Assume Role**.
 2. Next to **Add a principal**, select **Add**. The **Add principal** window opens.

Under **Principal type**, select **IAM Users**.

Under **ARN**, paste the IAM user ARN you copied to the clipboard.

Select **Add principal**.

3. Verify that the **Principal** line in the trust policy now contains the ARN you specified:

`"Principal": { "AWS": "arn:aws:iam::123456789012:user/username" }`

7. Resolve any security warnings, errors, or general warnings generated during [policy validation \(p. 588\)](#), and then choose **Next**.
8. In **Add permissions**, select the check box next to the permissions policy to apply. For this tutorial we are going to select the **SupportUser** trust policy. You can then use this role to troubleshoot and resolve issues with the AWS account and open support cases with AWS. We are not going to set a [permissions boundary \(p. 501\)](#) at this time.
9. Choose **Next**.
10. In **Name, review, and create** complete these settings:
 - For **Role name**, enter a name that identifies this role, such as *SupportUserRole*.
 - For **Description**, explain the intended use of the role.

Because other AWS resources might reference the role, you cannot edit the name of the role after it has been created.

11. Select **Create role**.

After the role is created, share the role information with the people who require the role. You can share the role information by:

- **Role link:** Send users a link that takes them to the **Switch Role** page with all the details already filled in.
- **Account ID or alias:** Provide each user with the role name along with the account ID number or account alias. The user then goes to the **Switch Role** page and adds the details manually.
- Saving the role link information along with the EmergencyAccess user credentials.

For details, see [Providing information to the user \(p. 278\)](#).

Create your first IAM policy

IAM policies are attached 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.

To create your first IAM policy

1. Follow the sign-in procedure appropriate to your user type as described in the topic [How to sign in to AWS in the AWS Sign-In User Guide](#).

2. On the **Console Home** page, select the IAM service.

3. In the navigation pane, choose **Policies**.

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

4. Choose **Create policy**.

5. On the **Create policy** page, select the **JSON** tab and then select **Import managed policy**.

6. In the **Import managed policies** window, in the **Search** box, enter **power** to reduce the list of policies. Select the **PowerUserAccess** policy.

7. Select **Import**. The policy displays in the **JSON** tab.

8. Select **Next: Tags** and then **Next: Review**.

9. On the **Review policy** page, for **Name**, enter **PowerUserExamplePolicy**. For **Description**, enter **Allows full access to all services except those for user management**. Then choose **Create policy** to save the policy.

You can attach this policy to a role to provide users who assume that role the permissions associated with this policy. The **PowerUserAccess** policy is commonly used to provide access to developers.

Getting access keys for programmatic access

Users need programmatic access if they want to interact with AWS outside of the AWS Management Console. The way to grant programmatic access depends on the type of user that's accessing AWS:

- If you manage identities in IAM Identity Center, the AWS APIs require a profile, and the AWS Command Line Interface requires a profile or an environment variable.
- If you have IAM users, the AWS APIs and the AWS Command Line Interface require access keys. Whenever possible, create temporary credentials that consist of an access key ID, a secret access key, and a security token that indicates when the credentials expire.

To grant users programmatic access, choose one of the following options.

Which user needs programmatic access?	To	By
Workforce identity (Users managed in IAM Identity Center)	Use short-term credentials to sign programmatic requests to the AWS CLI or AWS APIs (directly or by using the AWS SDKs).	<p>Following the instructions for the interface that you want to use:</p> <ul style="list-style-type: none"> For the AWS CLI, follow the instructions in Getting IAM role credentials for CLI access in the <i>AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide</i>. For the AWS APIs, follow the instructions in SSO credentials in the <i>AWS SDKs and Tools Reference Guide</i>.
IAM	Use short-term credentials to sign programmatic requests to the AWS CLI or AWS APIs (directly or by using the AWS SDKs).	Following the instructions in Using temporary credentials with AWS resources .
IAM	<p>Use long-term credentials to sign programmatic requests to the AWS CLI or AWS APIs (directly or by using the AWS SDKs).</p> <p>(Not recommended)</p>	Following the instructions in Managing access keys for IAM users .

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: Grant access to the billing console \(p. 35\)](#)
- [IAM tutorial: Delegate access across AWS accounts using IAM roles \(p. 40\)](#)
- [IAM tutorial: Create and attach your first customer managed policy \(p. 49\)](#)
- [IAM tutorial: Define permissions to access AWS resources based on tags \(p. 51\)](#)
- [IAM tutorial: Permit users to manage their credentials and MFA settings \(p. 66\)](#)

IAM tutorial: Grant access to the billing console

The AWS account owner ([AWS account root user \(p. 468\)](#)) can grant IAM users and roles access to the AWS Billing and Cost Management data for their AWS account. The instructions in this tutorial 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.

[Step 1: Activate access to billing data on your AWS test account \(p. 36\)](#)

In this scenario, you grant general access to allow IAM users and roles access to the AWS Billing and Cost Management console. This setting doesn't grant IAM users and roles the necessary permissions for these console pages, it enables access for IAM users or roles that have the required IAM policies. If policies are already attached to IAM users or roles, but this setting isn't enabled, the permissions granted by those policies aren't in effect.

Note

Member accounts created using AWS Organizations have this access enabled by default.

[Step 2: Sign-in as the root user on your AWS test account and create test users and groups \(p. 36\)](#)

In this scenario, you grant IAM users access to the billing console and you create two users:

- Pat Candella

Pat is a member of the finance department and works with billing and payments. Pat requires full access to the billing information in your AWS account.

- Terry Whitlock

Terry is part of your IT support department. Most of the time Terry doesn't require access to the billing console, but sometimes needs access to answer questions for employees in the finance department.

[Step 3: Create a role to grant access to the AWS Billing console \(p. 38\)](#)

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's an AWS identity with permission policies that determine what the identity can and can't 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 doesn't 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. In this scenario you create a role that Terry Whitlock can assume to access the billing console.

[**Step 4: Test access to the console \(p. 38\)**](#)

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. By testing the access of each user you can compare the user experiences.

Prerequisites

Make the following preparations before performing the steps in this tutorial:

- Create a test AWS account
- Record the AWS account number of your test account so that you can use it in the tutorial.

In this tutorial we use the example account number 111122223333. Whenever a step uses that account number, replace it with your test account number.

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

In this scenario, you grant general access to allow IAM users and roles access to the AWS Billing and Cost Management console. This setting doesn't grant IAM users and roles the necessary permissions for these console pages, it just enables access for IAM users or roles that have the required IAM policies.

Note

Member accounts created using AWS Organizations have this access enabled by default.

To activate IAM user and role access to the Billing and Cost Management console

1. Sign in to the AWS Management Console with your root user credentials (specifically, the email address and password that you used to create your AWS account).
2. On the navigation bar, select your account name, and then select [Account](#).
3. Scroll down the page until you find the section **IAM User and Role Access to Billing Information**, then select **Edit**.
4. Select the **Activate IAM Access** check box to activate access to the Billing and Cost Management console pages.
5. Choose **Update**.

The page displays the message **IAM user/role access to billing information is activated**.

In the next step of this tutorial you attach IAM policies to grant or deny access to specific billing features.

Step 2: Sign-in as the root user on your AWS test account and create test users and groups

Your test AWS account doesn't have any identities defined except for the root user. To provide access to billing information we create additional identities to whom we can grant permission to access billing information.

Sign-in as the root user on your AWS test account and create test users and groups

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

As the root user, you can't sign in to the **Sign in as IAM user** page. If you see the **Sign in as IAM user** page, choose **Sign in using root user email** near the bottom of the page. For help signing in as the root user, see [Signing in to the AWS Management Console as the root user](#) in the *AWS Sign-In User Guide*.

2. In the navigation pane, select **Users** and then select **Add users**.

Note

If you have IAM Identity Center enabled, the AWS Management Console displays a reminder that it's best to manage users' access in IAM Identity Center. In this tutorial, the IAM users we create are to learn about providing access to billing information. If you have created users in IAM Identity Center you assign the **Billing** permission set to those users or groups using IAM Identity Center instead of IAM.

3. For **User name**, enter **pcandella**. Names can't contain spaces.
4. Select the select box next to **Provide user access to the AWS Management Console—optional** and then choose **want to create an IAM user**.
5. Under **Console password**, select **Autogenerated password**.
6. Clear the select box next to **User must create a new password at next sign-in (recommended)** and then select **Next**. Because this IAM user is for testing, we're going to download the password for use during the verification procedure.
7. On the **Set permissions** page, under **Permissions options**, select **Add user to group**. Then, under **User groups**, select **Create group**.
8. On the **Create user group** page, in **User group name**, enter **BillingGroup**. Then, under **Permissions policies**, select the AWS managed job function policy **Billing**.
9. Select **Create user group** to return to the **Set permissions** page.
10. Under **User groups**, select the select box of the **BillingGroup** you created.
11. Select **Next** to proceed to the **Review and create** page.
12. On the **Review and create** page, review the list of user group memberships for the new user. When you are ready to proceed, select **Create user**.
13. On the **Retrieve password** page, select **Download .csv file** to save a .csv file with the user sign-in information (Connection URL, user name, and password).

Save this file to use as a reference when you sign in to AWS as this IAM user

14. Select **Return to users list**
15. Repeat this procedure using the following modifications to create the user for Terry Whitlock and a group for support users.
 - a. In step 3, for **User name**, enter **twhitlock**.
 - b. In step 8, for **User group name**, enter **SupportGroup**. Then, under **Permissions policies**, select the AWS managed-job function policy **SupportUser**.

You can review the new IAM users, groups and roles in the console lists. For each item you created you can select the name to view its details. When you view the user details, the console displays **Billing** listed under **Permissions policies** for **pcandella** and **SupportUser** listed under **Permissions policies** for **twhitlock**.

For more information about using policies to grant IAM users access to AWS Billing and Cost Management features, see [Using identity-based policies \(IAM policies\) for AWS Billing](#) in the *AWS Billing User Guide*.

Step 3: Create a role to grant access to the AWS Billing console

You can use a role to grant IAM users access to the billing console. Roles provide temporary credentials that users can assume when needed. In this tutorial, the user **twhitlock** needs to be able to access billing information when a support request from the finance department requires he investigate an issue.

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

As the root user, you can't sign in to the **Sign in as IAM user** page. If you see the **Sign in as IAM user** page, choose **Sign in using root user email** near the bottom of the page. For help signing in as the root user, see [Signing in to the AWS Management Console as the root user](#) in the [AWS Sign-In User Guide](#).

2. In the navigation pane, select **Users** and then select the **twhitlock** user to view the user details. Copy the ARN for the **twhitlock** user to the clipboard.
3. In the navigation pane, select **Roles** and then select **Create role**.
4. On the **Select trusted entity** page, select **Custom trust policy** and then under **Edit statement** complete the following items:
 - **Add actions for STS** - Verify that **AssumeRole** is selected.
 - **Add a principal** select **Add** to display the **Add principal** dialog box. For **Principal type** select **IAM users** then for **ARN** paste the ARN for the **twhitlock** user that you copied to the clipboard in step 16. Then select **Add principal**.
5. Select **Next** to go to the **Add permissions** page.
6. Under **Permissions policies** in the filter box, enter **Billing** and then select the AWS managed-job function policy **Billing**.
7. Select **Next** to go to the **Name, review, and create** page. Under **Role name**, enter **TempBillingAccess** then select **Create role**.

You are notified that the role has been created. View the role to display the details about the role. In the **Summary** section take note of the following information:

- **Maximum session duration** is 1 hour by default. After that time the user who assumed the role reverts to their base account permissions. If the user wants to continue using the role permissions, they must switch roles again. You can edit the role to increase the maximum duration. The longest session duration possible is 12 hours.
- **Link to switch roles in console.** You can copy the link to provide it directly to the users that you add as principals in the trust policy. You can view and edit the trust policy from the **Trust relationships** tab.

Step 4: Test access to the console

We recommend that you test access by signing in as 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 users

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 user 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 **pcandella**.
- b. On the navigation bar, choose **pcandella@111122223333**, and then choose **Billing Dashboard**.
- c. Browse through the pages and choose the various buttons to make sure that you have full modify permissions.

No access

- a. Sign in to your AWS account as the user **twhitlock**.
- b. On the navigation bar, choose **twhitlock@111122223333**, and then choose **Billing Dashboard**.
- c. A message displays stating **You need permissions**. No billing data is visible.

Switch role to elevate access

- a. Sign in to your AWS account as the user **twhitlock**.
- b. On the navigation bar, choose **twhitlock@111122223333**, and then choose **Switch role**.

The **Switch role** page opens. Complete the information as follows:

- **Account-111122223333**
- **Role-TempBillingAccess**

Select **Switch role**

Alternatively, you could use the URL provided in **Link to switch roles in console** to open the **Switch role** page.

- c. The console displays the **AWS Billing Dashboard** and the navigation bar displays **TempBillingAccess@111122223333**.

Summary

You've now completed the steps necessary to provide IAM users access to the AWS Billing console. As a result, you've seen firsthand what your users billing console experience is like. You can now proceed to implement this logic in your production environment at your convenience.

Related resources

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

- [Activating Access to the AWS Billing console](#)
- [AWS Billing policy examples](#)
- [Using identity-based policies \(IAM policies\) for AWS Billing](#)
- [Migrating access control for AWS Billing](#)

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

- [Managed policies and inline policies \(p. 494\)](#)
- [Controlling IAM users access to the AWS Management Console \(p. 80\)](#)
- [Attaching a policy to an IAM user group \(p. 180\)](#)

IAM tutorial: Delegate access across AWS accounts using IAM roles

This tutorial teaches you how to use a role to delegate access to resources in different AWS accounts that you own called **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 have to create individual IAM users in each account. In addition, users don't have to sign out of one account and sign in to another account 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.

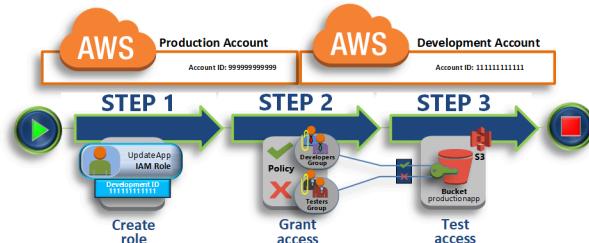
In this tutorial, the **Production** account manages live applications. Developers and testers use the **Development** account as a sandbox to freely test applications. In each account, you store application information in Amazon S3 buckets. You manage IAM users in the **Development** account, where you have two IAM user groups: **Developers** and **Testers**. Users in both user 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. The developers store these applications 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) allowed to assume a specific role in the **Production** account.
- A role in the **Production** account (the trusting account) 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 authenticated by temporary credentials provided by the role. Similar attempts by a Tester to use the role fail.

This workflow has three basic steps:



Create a role in the Production Account (p. 41)

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.

Grant access to the role (p. 44)

In this section, you modify the IAM user group policy to deny Testers access to the *UpdateApp* role. Because Testers have PowerUser access in this scenario, and you must explicitly *deny* the ability to use the role.

Test access by switching roles (p. 45)

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 user groups in the **Development** account created and configured as follows:

User	User group	Permissions
David	Developers	Both users can sign in and use the AWS Management Console in the Development account.
Jane	Testers	

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

Create a role in the Production Account

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 permissions to only read and write access to the productionapp bucket. Anyone 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. Each AWS account has a unique account ID identifier assigned to it.

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, you can use the account ID 111111111111 for the **Development** account. However, you should use a valid account ID if you use this 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 for the role requirements. 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, 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 for your Amazon 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": [  
                "s3GetObject",  
                "s3PutObject",  
                "s3DeleteObject"  
            ],  
            "Resource": "arn:aws:s3:::productionapp/*"  
        }  
    ]  
}
```

The `ListAllMyBuckets` action grants permission to list all buckets owned by the authenticated sender of the request. 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. Resolve any security warnings, errors, or general warnings generated during [policy validation \(p. 588\)](#), and then choose **Next**.

Note

You can switch between the **Visual** and **JSON** editor options anytime. However, if you make changes or choose **Next** in the **Visual** editor, IAM might restructure your policy to optimize it for the visual editor. For more information, see [Policy restructuring \(p. 1181\)](#).

5. On the **Review and create** page, type **read-write-app-bucket** for the policy name. Review 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, choose **Roles** and then choose **Create role**.
7. Choose the **An 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. Leave these options unselected. For more information, see [Using multi-factor authentication \(MFA\) in AWS \(p. 114\)](#).

9. Choose **Next: Permissions** to set the permissions associated with the role.
10. Select the check 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 created. This hides the AWS created policies and makes it much easier to find the one you need.

Then, choose **Next**.

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 resources \(p. 399\)](#).
12. (Optional) For **Description**, enter a description for the new role.
13. After reviewing the role, choose **Create role**.

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

Now you must obtain the Amazon Resource Name (ARN) of the role, a unique identifier for the role. When you modify the Developers and Testers user group policy, you specify the role 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 provide the real AWS account ID for the 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 the users who switch to the **UpdateApp** role can do.

Next, modify the permissions for the user groups.

Grant access to the role

At this point, both Testers and Developers user group members have permissions that allow them to freely test applications in the **Development** account. Use the following required steps for adding permissions to allow switching to the role.

To modify the Developers user 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 **User groups**, and then choose **Developers**.
3. Choose the **Permissions** tab, choose **Add permissions**, and then choose **Create inline policy**.
4. Choose the **JSON** tab.
5. 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 succeeds.

6. Choose **Review policy**.
7. Type a **Name** such as **allow-assume-S3-role-in-production**.
8. Choose **Create policy**.

In most environments, you may not need the following procedure. If, however, you use PowerUserAccess 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 you do not need this procedure 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 you do not have a better option.

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

1. Choose **User groups**, and then choose **Testers**.
2. Choose the **Permissions** tab, choose **Add permissions**, and then choose **Create inline policy**.
3. Choose the **JSON** tab.
4. 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 receives an access denied message.

5. Choose **Review policy**.
6. Type a **Name** like **deny-assume-S3-role-in-production**.
7. Choose **Create policy**.

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

Next, you can see 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.

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 user group in the **Development** account with users allowed to use that role. 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 after you sign 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 sign in as the AWS account root user.

Switch roles (console)

If David needs to work within 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 uses 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 only works 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 \(p. 191\)](#), and [How to Enable Cross-Account Access to the AWS Management Console](#) in the AWS Security Blog.

IAM provides two ways that David can use to enter the **Switch Role** page:

- David receives a link from their administrator that points to a predefined 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 Roles**.
- The administrator does not send the link in email, but instead sends the **Account ID** number and **Role Name** values. To switch roles, David must manually enter the values. This is illustrated in the following procedure.

To assume a role

1. David signs into the AWS Management Console using his normal user in the **Development** user group.
2. They choose the link that the administrator emailed to them. This takes David to the **Switch Role** page with the account ID or alias and the role name information already filled in.

—or—

David chooses their name (the Identity menu) on the navigation bar, and then chooses **Switch Roles**.

If this is the first time that David tries to access the Switch Role page this way, he first lands on a first-run **Switch Role** page. This page provides additional information on how switching roles can permit users to manage resources across AWS accounts. David must choose **Switch Role** 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 currently active in IAM. To keep track of this information, he types PRODUCTION in the **Display Name** text box, chooses 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 done, David can return to their original permissions. To do that, they choose the **PRODUCTION** role display name on the navigation bar and then choose **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 reentering 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 uses the role, he cannot use his power-user privileges in the **Development** account, because only one set of permissions can be in effect at a time.

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

To 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 received 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
+Im5ZEXAMPLEeYjs1M2FUiG1Jx9tQqNMBEXAMPLE
CvSRyh0FW7jEXAMPLEW+vE/7s1HRpXviG7b+qYf4nD00EXAMPLEmj4wxS04L/
uZEXAMPLEcihzFB51TYLto9dyBgSDy
EXAMPLE9/
g7QRUhZp4qbEXAMPLEnGPy0j59pFA41NKCIkVgkREXAMPLEj1zxQ7y52gekeVEXAMPLEDiB9ST3Uuysg
sKdEXAMPLE1TVastU1A0SKFEXAMPLEiywCC/Cs8EXAMPLEpZgOs+6hz4AP4KEXAMPLErbASP
+4eZScEXAMPLEsnf87e
NhYDHq6ikBQ==",
        "Expiration": "2014-12-11T23:08:07Z",
        "AccessKeyId": "AKIAIOSFODNN7EXAMPLE"
    }
}
```

3. David sees the three pieces that they need in the Credentials section of the output.

- AccessKeyId
- SecretAccessKey
- SessionToken

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

4. To add the three values to the environment, David cuts and pastes the output of the previous step into the following commands. 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.

For details on using Tools for Windows Powershell, see [Switching to an IAM role \(Tools for Windows PowerShell\) \(p. 289\)](#)

```
set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfICYEXAMPLEKEY
set AWS_SESSION_TOKEN=AQoDYXdzEGcaEXAMPLE2gsYULo
+Im5ZEXAMPLEeYjs1M2FUiG1Jx9tQqNMBEXAMPLEcvS
Ryh0FW7jEXAMPLEW+vE/7s1HRpXviG7b+qYf4nD00EXAMPLEmj4wxS04L/
uZEXAMPLEcihzFB51TYLto9dyBgSDyEXA
MPLEKEY9/
g7QRUhZp4qbEXAMPLEnGPy0j59pFA41NKCIkVgkREXAMPLEj1zxQ7y52gekeVEXAMPLEDiB9ST3UusKd
```

```
EXAMPLE1TVastU1A0SKFEXAMPLEiywCC/Cs8EXAMPLEpZgOs+6hz4AP4KEXAMPLERbASP
+4eZScEXAMPLENhkxiHen
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.

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

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

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

Using AssumeRole (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. They 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. They 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 \(AWS API\) \(p. 291\)](#).

Related resources

- For more information about IAM users and user groups, see [IAM Identities \(users, user groups, and roles\) \(p. 70\)](#).
- For more information about Amazon S3 buckets, see [Create a Bucket](#) in the *Amazon Simple Storage Service User 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 user 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 \(p. 496\)](#) 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 \(p. 49\)](#)

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 \(p. 50\)](#)

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.

[Step 3: Test user access \(p. 50\)](#)

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** option 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. Resolve any security warnings, errors, or general warnings generated during [policy validation \(p. 588\)](#), and then choose **Next**.

Note

You can switch between the **Visual** and **JSON** editor options 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 \(p. 1181\)](#).

6. On the **Review and create** page, type **UsersReadOnlyAccessToIAMConsole** for the policy name. Review 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 choose the radio button next to **UsersReadOnlyAccessToIAMConsole** in the list.
3. Choose the **Actions** button, and then choose **Attach**.
4. In IAM entities choose the option to filter for **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

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 \(p. 494\)](#)
- [Controlling IAM users access to the AWS Management Console \(p. 80\)](#)

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 resources, including IAM entities (users or roles) and to AWS resources. You can 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? \(p. 15\)](#).

Note

You must pass a single value for each session tag. AWS Security Token Service does not support multi-valued session tags.

Topics

- [Tutorial overview \(p. 51\)](#)
- [Prerequisites \(p. 52\)](#)
- [Step 1: Create test users \(p. 53\)](#)
- [Step 2: Create the ABAC policy \(p. 54\)](#)
- [Step 3: Create roles \(p. 57\)](#)
- [Step 4: Test creating secrets \(p. 57\)](#)
- [Step 5: Test viewing secrets \(p. 59\)](#)
- [Step 6: Test scalability \(p. 60\)](#)
- [Step 7: Test updating and deleting secrets \(p. 61\)](#)
- [Summary \(p. 62\)](#)
- [Related resources \(p. 63\)](#)
- [IAM tutorial: Use SAML session tags for ABAC \(p. 63\)](#)

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 \(p. 1224\)](#). 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 \(p. 417\)](#) 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 \(p. 63\)](#).

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 \(p. 63\)](#).
- 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 a user with administrative permissions.
- 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 \(p. 582\)](#).

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.

To use this policy, replace the *italicized placeholder text* with your account information.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "TutorialAssumeRole",  
            "Effect": "Allow",  
            "Action": "sts:AssumeRole",  
            "Resource": "arn:aws:iam::account-ID-without-hyphens: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 user groups \(p. 177\)](#) and [Adding and removing users in an IAM user group \(p. 179\)](#).

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	access-project = peg access-team = eng cost-center = 987654
access-Mary-peg-qas	access-project = peg access-team = qas cost-center = 987654
access-Saanvi-uni-eng	access-project = uni access-team = eng cost-center = 123456
access-Carlos-uni-qas	access-project = uni access-team = qas cost-center = 123456

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 \(p. 582\)](#).

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

- [Controlling access for IAM principals \(p. 522\)](#)
- [Amazon EC2: Allows starting or stopping EC2 instances a user has tagged, programmatically and in the console \(p. 552\)](#)
- [EC2: Start or stop instances based on matching principal and resource tags \(p. 553\)](#)
- [EC2: Start or stop instances based on tags \(p. 552\)](#)
- [IAM: Assume roles that have a specific tag \(p. 557\)](#)

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": {
                "ForAnyValue: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 is 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 [Multivalued context keys \(p. 1294\)](#).
- 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 \(p. 1288\)](#) 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 \(p. 251\)](#). If you choose to use federation instead of IAM users and roles, see [IAM tutorial: Use SAML session tags for ABAC \(p. 63\)](#).

ABAC roles

Job function	Role tags	Role name	Role description
Project Pegasus Engineering	access-project = peg access-team = eng cost-center = 987654	access-peg-engineering	Allows engineers to read all engineering resources and create and manage Pegasus engineering resources.
Project Pegasus Quality Assurance	access-project = peg access-team = qas cost-center = 987654	access-peg-quality-assurance	Allows the QA team to read all QA resources and create and manage all Pegasus QA resources.
Project Unicorn Engineering	access-project = uni access-team = eng cost-center = 123456	access-uni-engineering	Allows engineers to read all engineering resources and create and manage Unicorn engineering resources.
Project Unicorn Quality Assurance	access-project = uni access-team = qas cost-center = 123456	access-uni-quality-assurance	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-Arvind-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 \(console\) \(p. 282\)](#).

This operation fails because the access-project and cost-center tag values do not match for the access-Arvind-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 \(console\) \(p. 282\)](#).
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 peg.
uni	eng	987654	(none)	Denied because the access-project tag value does not match the role's value of peg.
peg	qas	987654	(none)	Denied because the access-team tag value does not match the role's value of eng.
peg	eng	123456	(none)	Denied because the cost-center tag value does not match the role's value of 987654.
peg	eng	987654	owner = Jane	Denied because the additional tag owner 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 Name 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

User name	Role name	Secret name	Secret tags
			access-team = qas 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-uni-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 \(console\) \(p. 282\)](#).

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

7. 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 **Centaur**. An engineer named Saanvi Sarkar will be the lead engineer for **Centaur** while continuing to work on the **Unicorn** project. Saanvi will also review work for the **Peg** project. There are also several newly hired engineers, including Nikhil Jayashankar, who will work on only the **Centaur** project.

To add the new project to AWS

1. Sign in as the IAM administrator user and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. 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 \(p. 57\)](#) and [Step 5: Test viewing secrets \(p. 59\)](#). 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 the user 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 \(console\) \(p. 600\)](#).

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 \(p. 400\)](#). Instead, you must manually specify the two roles that she can assume.

To use this policy, replace the *italicized placeholder text* with your account information.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "TutorialAssumeSpecificRoles",  
            "Effect": "Allow",  
            "Action": "sts:AssumeRole",  
            "Resource": [  
                "arn:aws:iam::account-ID-without-hyphens:role/access-peg-engineering",  
                "arn:aws:iam::account-ID-without-hyphens:role/access-cen-engineering"  
            ]  
        }  
    ]  
}
```

9. Use the procedures in [Step 4: Test creating secrets \(p. 57\)](#) and [Step 5: Test viewing secrets \(p. 59\)](#). 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-uni-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 \(console\) \(p. 282\)](#).

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 \(p. 1309\)](#).

Related resources

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

- [What is ABAC for AWS? \(p. 15\)](#)
- [AWS global condition context keys \(p. 1338\)](#)
- [Creating IAM users \(console\) \(p. 77\)](#)
- [Creating a role to delegate permissions to an IAM user \(p. 251\)](#)
- [Tagging IAM resources \(p. 399\)](#)
- [Controlling access to AWS resources using tags \(p. 523\)](#)
- [Switching to a role \(console\) \(p. 282\)](#)
- [IAM tutorial: Use SAML session tags for ABAC \(p. 63\)](#)

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 resources, including IAM entities (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 \(p. 417\)](#) 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? \(p. 15\)](#).

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 \(p. 51\)](#) by passing SAML attributes to your session principal, complete the tasks in [IAM tutorial: Define permissions to access AWS resources based on tags \(p. 51\)](#), 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.
- The ability to sign in as a user with administrative permissions.

- 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 \(p. 422\)](#).

Step 1: Create test users

Skip the instructions in [Step 1: Create test users \(p. 53\)](#). 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 \(p. 54\)](#) 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 \(p. 57\)](#).

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 \(console\) \(p. 268\)](#).

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 \(console\) \(p. 385\)](#).

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 \(p. 422\)](#).

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 3C: 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 \(p. 230\)](#).

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 \(p. 230\)](#). Then follow the instructions in [Step 4: Test creating secrets \(p. 57\)](#) 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 \(p. 57\)](#).

Step 5: Test viewing secrets

Follow the instructions in [Step 5: Test viewing secrets \(p. 59\)](#) 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 \(p. 60\)](#) 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 \(p. 61\)](#) 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 \(p. 1309\)](#).

IAM tutorial: Permit users to manage their credentials and MFA settings

You can permit 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 \(p. 67\)**](#)

Create a customer managed policy that prohibits all actions **except** the few IAM actions. These exceptions 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 \(console\) \(p. 102\)](#).

[**Step 2: Attach policies to your test user group \(p. 68\)**](#)

Create a user group whose members have full access to all Amazon EC2 actions if they sign in with MFA. To create such a user 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 \(p. 68\)**](#)

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 \(p. 116\)](#), [FIDO security key \(p. 120\)](#), or [hardware-based MFA device \(p. 129\)](#).
- A test IAM user who is a member of a user group as follows:

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

Important

IAM [best practices \(p. 1032\)](#) recommend that you require human users to use federation with an identity provider to access AWS using temporary credentials instead of using IAM users with long-term credentials.

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 \(p. 533\)](#).
5. Paste the policy text into the **JSON** text box. Resolve any security warnings, errors, or general warnings generated during policy validation, and then choose **Next**.

Note

You can switch between the **Visual editor** and **JSON** options 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 **JSON** to continue working with this policy.

This example policy does not allow users to reset a password while signing in to the AWS Management Console for the first time. We recommend that you do not grant permissions to new users until after they sign in and reset their password.

6. On the **Review and create** 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**. In the **Tags** area, you can

optionally add tag key-value pairs to the customer managed policy. Review 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 user group

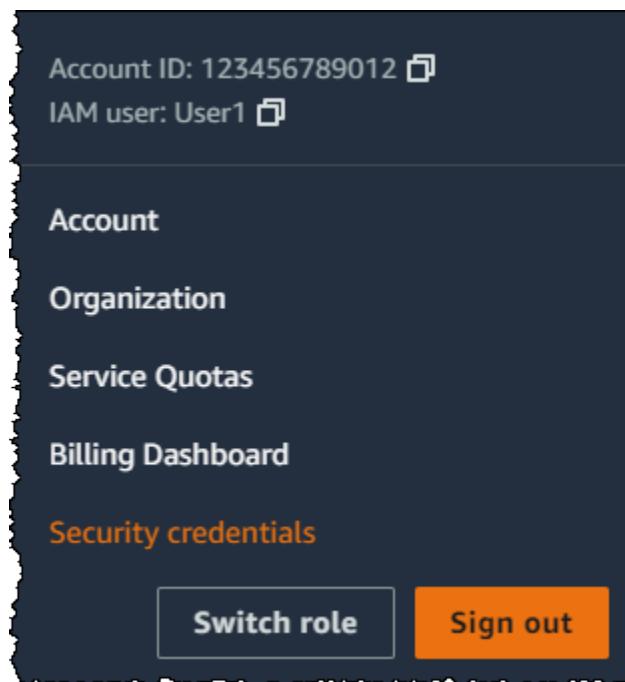
Next you attach two policies to the test IAM user group, which will be used to grant the MFA-protected permissions.

1. In the navigation pane, choose **User groups**.
2. In the search box, type **EC2MFA**, and then choose the group name (not the check box) in the list.
3. Choose the **Permissions** tab, choose **Add permissions**, and then choose **Attach policies**.
4. On the **Attach permission policies to EC2MFA group** page, in the search box, type **EC2Full**. 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 policies**.

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 **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 unassigned virtual MFA device. For more information, see [I am not authorized to perform: iam>DeleteVirtualMFADevice \(p. 1172\)](#).

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

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 **Set up 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 **Set up device** wizard, in the **Enter the code from your authenticator app** box, type the one-time password that currently appears in the virtual MFA device. Choose **Register MFA**.

Important

Submit your request immediately after generating the code. 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 \(p. 137\)](#).

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 additional information, see the following topics:

- [Using multi-factor authentication \(MFA\) in AWS \(p. 114\)](#)
- [Enabling MFA devices for users in AWS \(p. 115\)](#)
- [Using MFA devices with your IAM sign-in page \(p. 83\)](#)

IAM Identities (users, user groups, and roles)

Tip

Having trouble signing in to AWS? Make sure that you're on the correct sign-in page.

- To sign in as the AWS account root user (account owner), use the credentials that you set up when you created the AWS account.
- To sign in as an IAM user, use the credentials your account administrator gave you to sign in to AWS.
- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

For help signing in using an IAM Identity Center user, see [Signing in to the AWS access portal](#) in the *AWS Sign-In User Guide*.

For sign-in tutorials, see [How to sign in to AWS](#) in the *AWS Sign-In User Guide*.

Note

If you need to request support, do not use the **Feedback** link on this page. Feedback you enter is received by the AWS Documentation team, not AWS Support. Instead, choose the **Contact Us** link at the top of this page. There, you'll find links to resources to help you get the support that you need.

The AWS account root user or an administrative user for the account can create *IAM identities*. An IAM identity provides access to an AWS account. An *IAM user group* is a collection of IAM users managed as a unit. An IAM identity represents a human user or programmatic workload, 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 user group can perform, on which AWS resources, and under what conditions.

[AWS account root user \(p. 468\)](#)

When you first create an AWS account, you begin with one 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 don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see [Tasks that require root user credentials](#) in the *AWS Account Management Reference Guide*.

[IAM users \(p. 74\)](#)

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. Where possible, [best practices \(p. 1032\)](#) recommend relying on temporary credentials

instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see [Rotate access keys regularly for use cases that require long-term credentials \(p. 1034\)](#). To add IAM users to your AWS account, see [Creating an IAM user in your AWS account \(p. 76\)](#).

IAM user groups (p. 175)

An [IAM group](#) is an identity that specifies a collection of IAM users. You can't use a group to sign-in. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMPublishers* and give that group the types of permissions that publishing workloads typically need.

IAM roles (p. 183)

An [IAM role](#) is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by [switching roles](#). You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see [Using IAM roles \(p. 274\)](#).

IAM roles with temporary credentials are used in the following situations:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see [Creating a role for a third-party Identity Provider](#) in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see [Permission sets](#) in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*.
- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see [Cross account resource access in IAM \(p. 526\)](#).
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
 - **Principal permissions** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see [Actions, resources, and condition keys for AWS Identity and Access Management](#) in the *Service Authorization Reference*.
 - **Service role** – A service role is an [IAM role](#) that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see [Creating a role to delegate permissions to an AWS service](#) in the *IAM User Guide*.

- **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see [Using an IAM role to grant permissions to applications running on Amazon EC2 instances](#) in the *IAM User Guide*.

Temporary credentials in IAM (p. 426)

As a [best practice \(p. 1032\)](#), use temporary credentials with both human users and workloads. 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 use IAM Identity Center users?

We recommend that all human users use IAM Identity Center to access AWS resources. IAM Identity Center enables significant improvements over accessing AWS resources as an IAM user. IAM Identity Center provides:

- A central set of identities and assignments
- Access to accounts across an entire AWS Organization
- Connection to your existing identity provider
- Temporary credentials
- Multi-factor authentication (MFA)
- Self-service MFA configuration for end-users
- Administrative enforcement of MFA usage
- Single sign-on to all AWS account entitlements

For more information, see [What is IAM Identity Center](#) in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*.

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

We recommend you only use IAM users for use cases not supported by federated users. Some of the use cases include the following:

- **Workloads that cannot use IAM roles** – You might run a workload from a location that needs to access AWS. In some situations, you can't use IAM roles to provide temporary credentials, such as for WordPress plugins. In these situations, use IAM user long-term access keys for that workload to authenticate to AWS.

- **Third-party AWS clients** – If you are using tools that don't support access with IAM Identity Center, such as third-party AWS clients or vendors that are not hosted on AWS, use IAM user long-term access keys.
- **AWS CodeCommit access** – If you are using CodeCommit to store your code, you can use an IAM user with either SSH keys or service-specific credentials for CodeCommit to authenticate to your repositories. We recommend that you do this in addition to using a user in IAM Identity Center for normal authentication. Users in IAM Identity Center are the people in your workforce who need access to your AWS accounts or to your cloud applications. To give users access to your CodeCommit repositories without configuring IAM users, you can configure the `git-remote-codecommit` utility. For more information about IAM and CodeCommit, see [Using IAM with CodeCommit: Git credentials, SSH keys, and AWS access keys \(p. 168\)](#). For more information about configuring the `git-remote-codecommit` utility, see [Connecting to AWS CodeCommit repositories with rotating credentials](#) in the [AWS CodeCommit User Guide](#).
- **Amazon Keyspaces (for Apache Cassandra) access** – In a situation where you are unable to use users in IAM Identity Center, such as for testing purposes for Cassandra compatibility, you can use an IAM user with service-specific credentials to authenticate with Amazon Keyspaces. Users in IAM Identity Center are the people in your workforce who need access to your AWS accounts or to your cloud applications. You can also connect to Amazon Keyspaces using temporary credentials. For more information, see [Using temporary credentials to connect to Amazon Keyspaces using an IAM role and the SigV4 plugin](#) in the [Amazon Keyspaces \(for Apache Cassandra\) Developer Guide](#).
- **Emergency access** – In a situation where you cannot access your identity provider and you must take action in your AWS account. Establishing emergency access IAM users can be part of your resiliency plan. We recommend that the emergency user credentials be tightly controlled and secured using multi-factor authentication (MFA).

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 \(p. 374\)](#).

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 User Guide](#)
- [About web identity federation \(p. 199\)](#)

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 \(p. 205\)](#).

- 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 \(p. 233\)](#).

IAM users

Important

IAM [best practices \(p. 1032\)](#) recommend that you require human users to use federation with an identity provider to access AWS using temporary credentials instead of using IAM users with long-term credentials.

An AWS Identity and Access Management (IAM) *user* is an entity that you create in AWS. The IAM user represents the human user or workload who uses the IAM user 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 \(p. 468\)](#).

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 an IAM user, IAM creates these ways to identify that user:

- A "friendly name" for the IAM user, which is the name that you specified when you created the IAM user, such as Richard or Anaya. These are the names you see in the AWS Management Console.
- An Amazon Resource Name (ARN) for the IAM user. You use the ARN when you need to uniquely identify the IAM user across all of AWS. For example, you could use an ARN to specify the IAM 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 IAM user. This ID is returned only when you use the API, Tools for Windows PowerShell, or AWS CLI to create the IAM user; you do not see this ID in the console.

For more information about these identifiers, see [IAM identifiers \(p. 1213\)](#).

IAM users and credentials

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

- [Console password \(p. 93\)](#): A password that the IAM user can type to sign in to interactive sessions such as the AWS Management Console. Disabling the password (console access) for an IAM user prevents them from signing in to the AWS Management Console using their sign-in credentials. It does not change their permissions or prevent them from accessing the console using an assumed role.
- [Access keys \(p. 103\)](#): Used to make programmatic calls to AWS. However, there are more secure alternatives to consider before you create access keys for IAM users. For more information, see [Considerations and alternatives for long-term access keys](#) in the *AWS General Reference*. If the IAM user has active access keys, they continue to function and allow access through the AWS CLI, Tools for Windows PowerShell, AWS API, or the AWS Console Mobile Application.

- [SSH keys for use with CodeCommit \(p. 168\)](#): An SSH public key in the OpenSSH format that can be used to authenticate with CodeCommit.
- [Server certificates \(p. 171\)](#): 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 an IAM 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 your use case.

You have the following options to administer passwords, access keys, and multi-factor authentication (MFA) devices:

- [Manage passwords for your IAM users \(p. 93\)](#). 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 \(p. 103\)](#). Create and update access keys for programmatic access to the resources in your account.
- [Enable multi-factor authentication \(MFA\) for the IAM user \(p. 114\)](#). As a [best practice \(p. 1032\)](#), we recommend that you require multi-factor authentication for all IAM users in your account. With MFA, users must 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.
- [Find unused passwords and access keys \(p. 161\)](#). 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 \(p. 164\)](#). 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.

IAM users and permissions

By default, a new IAM user has no [permissions \(p. 484\)](#) to do anything. They are 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 create a password for him and attach permissions 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 \(p. 88\)](#). For procedures on how to change the user's password or access keys, see [Managing user passwords in AWS \(p. 93\)](#) and [Managing access keys for IAM users \(p. 103\)](#).

You can also add a permissions boundary to your IAM 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 an IAM user or role. For more information about policy types and uses, see [Policies and permissions in IAM \(p. 485\)](#).

IAM users and accounts

Each IAM user is associated with one and only one AWS account. Because IAM 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 IAM 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 AWS STS quotas \(p. 1220\)](#).

IAM 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



Important

IAM [best practices \(p. 1032\)](#) recommend that you require human users to use federation with an identity provider to access AWS using temporary credentials instead of using IAM users with long-term credentials.

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](#).

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 \(users, user groups, and roles\) \(p. 70\)](#).

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:
 - **Enable console access – optional:** If the user needs to access the AWS Management Console, [create a password for the user \(p. 97\)](#). Disabling console access for a user prevents them from signing in to the AWS Management Console using their user name and password. It does not change their permissions or prevent them from accessing the console using an assumed role.

Tip

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 \(p. 501\)](#) 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 resources \(p. 399\)](#).
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 \(MFA\) \(p. 114\)](#) 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 \(p. 100\)](#).

For information about the permissions that you need in order to create a user, see [Permissions required to access IAM resources \(p. 662\)](#).

Topics

- [Creating IAM users \(console\) \(p. 77\)](#)
- [Creating IAM users \(AWS CLI\) \(p. 79\)](#)
- [Creating IAM users \(AWS API\) \(p. 80\)](#)

Creating IAM users (console)

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

To create an IAM user (console)

1. Follow the sign-in procedure appropriate to your user type as described in the topic [How to sign in to AWS in the AWS Sign-In User Guide](#).
2. On the **Console Home** page, select the IAM service.
3. In the navigation pane, select **Users** and then select **Add users**.
4. On the **Specify user details** page, under **User details**, in **User name**, enter the name for the new user. This is their sign-in name for AWS.

Note

The number and size of IAM resources in an AWS account are limited. For more information, see [IAM and AWS STS quotas \(p. 1220\)](#). 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*. When a user name is used in a policy or as part of an ARN, the name is case sensitive. When a user name appears to customers in the console, such as during the sign-in process, the user name is case insensitive.

5. Select **Provide user access to the – AWS Management Console optional** This produces AWS Management Console sign-in credentials for the new user.

You are asked whether you are providing console access to a person. We recommend that you create users in IAM Identity Center rather than IAM.

- To switch to creating the user in IAM Identity Center, select **Specify a user in Identity Center**.

If you have not enabled IAM Identity Center, selecting this option takes you to the service page in the console so that you can enable the service. For details on this procedure, see <https://>

docs.aws.amazon.com/singlesignon/latest/userguide/getting-started.html in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*

If you have enabled IAM Identity Center, selecting this option takes you to the **Specify user details** page in IAM Identity Center. For details on this procedure, see <https://docs.aws.amazon.com/singlesignon/latest/userguide/addusers.html> in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*

- If you cannot use IAM Identity Center, select **I want to create an IAM user** and continue following this procedure.

- a. For **Console password**, select one of the following:

- **Autogenerated password** – The user gets a randomly generated password that meets the [account password policy \(p. 94\)](#). You can view or download the password when you get to the **Retrieve password** page.
- **Custom password** – The user is assigned the password that you enter in the box.

- b. (Optional) **Users must create a new password at next sign-in (recommended)** is selected by default to ensure that the user is 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.

6. Select **Next**.
7. On the **Set permissions** page, specify how you want to assign permissions for this user. Select one of the following three options:
 - **Add user to group** – Select this option if you want to assign the user 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 select **Create group** to create a new group. For more information, see [Changing permissions for an IAM user \(p. 88\)](#).
 - **Copy permissions** – Select this option to copy all of the group memberships, attached managed policies, embedded inline policies, and any existing [permissions boundaries \(p. 501\)](#) from an existing user to the new user. IAM displays a list of the users in your account. Select the one whose permissions most closely match the needs of your new user.
 - **Attach policies directly** – Select 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 user or select **Create policy** to open a new browser tab and create a new policy. For more information, see step 4 in the procedure [Creating IAM policies \(p. 582\)](#). After you create the policy, close that tab and return to your original tab to add the policy to the user.

Tip

Whenever possible, attach your policies to a group and then make users members of the appropriate groups.

8. (Optional) Set a [permissions boundary \(p. 501\)](#). This is an advanced feature.

Open the **Permissions boundary** section and select **Use a permissions boundary to control the maximum 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 select **Create policy** to open a new browser tab and create a new policy. For more information, see step 4 in the procedure [Creating IAM policies \(p. 582\)](#). After you create the policy, close that tab and return to your original tab to select the policy to use for the permissions boundary.

9. Select **Next**.

10. (Optional) On the **Review and create** page, under **Tags**, select **Add new tag** to add metadata to the user by attaching tags as key-value pairs. For more information about using tags in IAM, see [Tagging IAM resources \(p. 399\)](#).
11. Review all of the choices you made up to this point. When you are ready to proceed, select **Create user**.
12. On the **Retrieve password** page, get the password assigned to the user:
 - Select **Show** next to the password to view the user's password so that you can record it manually.
 - Select **Download .csv** to download the user's sign in credentials as a .csv file that you can save to a safe location.
13. Select **Email sign-in instructions**. Your local mail client opens with a draft that you can customize and send to the user. 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
```

Important

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

14. If the user also requires access keys, refer to [Managing access keys for IAM users \(p. 103\)](#).

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 users \(AWS CLI or AWS API\) \(p. 404\)](#).
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 \(p. 533\)](#).

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 users \(AWS CLI or AWS API\) \(p. 404\)](#).
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 \(p. 533\)](#).

Controlling IAM users access to the AWS Management Console

IAM 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 IAM users access to your

AWS account resources through the AWS Management Console. It also shows how IAM 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 IAM user who needs access to the AWS Management Console. Users access the console through your IAM-enabled AWS account sign-in page. For information about accessing the sign-in page, see [How to sign in to AWS](#) in the *AWS Sign-In User Guide*. For information about creating passwords, see [Managing user passwords in AWS \(p. 93\)](#).

You can prevent an IAM user from accessing the AWS Management Console by removing their password. This prevents them from signing into the AWS Management Console using their sign-in credentials. It does not change their permissions or prevent them from accessing the console using an assumed role. If the user has active access keys, they continue to function and allow access through the AWS CLI, Tools for Windows PowerShell, AWS API, or the AWS Console Mobile Application.

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

Even if your IAM users have passwords, they still need permission to access your AWS resources. When you create an IAM user, that user has no permissions by default. To give your IAM users the permissions they need, you attach policies to them. If you have many IAM users who perform the same tasks with the same resources, you can assign those IAM users to a group. Then assign the permissions to that group. For information about creating IAM users and groups, see [IAM Identities \(users, user groups, and roles\) \(p. 70\)](#). For information about using policies to set permissions, see [Access management for AWS resources \(p. 484\)](#).

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 your AWS account billing and usage information. For more information, see [Controlling Access to Your Billing Information](#) in the *AWS Billing 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 \(p. 484\)](#) and [Example IAM identity-based policies \(p. 529\)](#).

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 \(p. 233\)](#).

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 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. However, there are more secure alternatives to consider before you create access keys for users. For more information, see [Considerations and alternatives for long-term access keys](#) in the *AWS General Reference*.

Using MFA devices with your IAM sign-in page

Users who are configured with [multi-factor authentication \(MFA\) \(p. 114\)](#) devices must use their MFA devices to sign in to the AWS Management Console. After the user enters their sign-in credentials, 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 multiple MFA devices enabled \(p. 83\)](#)
- [Signing in with a FIDO security key \(p. 83\)](#)
- [Signing in with a virtual MFA device \(p. 83\)](#)
- [Signing in with a hardware TOTP token \(p. 84\)](#)

[Signing in with multiple MFA devices enabled](#)

If a user signs in to the AWS Management Console as an AWS account root user or IAM user with multiple MFA devices enabled for that account, they only need to use one MFA device to sign in. After the user authenticates with the user's password, they select which MFA device type they would like to use to finish authenticating. Then the user is prompted to authenticate with the type of device that they selected.

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

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

Note

Google Chrome users should not choose any of the available options on the pop-up that asks to **Verify your identity with amazon.com**. You only need to tap on the security key.

Unlike other MFA devices, FIDO security keys do not go out of sync. Administrators can deactivate a FIDO security key if it's lost or broken. For more information, see [Deactivating MFA devices \(console\) \(p. 142\)](#).

For information on browsers that support WebAuthn and FIDO-compliant devices that AWS supports, see [Supported configurations for using FIDO security keys \(p. 125\)](#).

[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 \(p. 137\)](#).

Signing in with a hardware TOTP token

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 TOTP token.

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 TOTP token can go out of sync. If a user can't 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 \(p. 137\)](#).

Managing IAM users

Note

As a [best practice \(p. 1032\)](#), we recommend that you require human users to use federation with an identity provider to access AWS using temporary credentials. If you follow the best practices, you are not managing IAM users and groups. Instead, your users and groups are managed outside of AWS and are able to access AWS resources as a *federated identity*. A federated identity is a user from your enterprise user directory, a web identity provider, the AWS Directory Service, the Identity Center directory, or any user that accesses AWS services by using credentials provided through an identity source. Federated identities use the groups defined by their identity provider. If you are using AWS IAM Identity Center (successor to AWS Single Sign-On), see [Manage identities in IAM Identity Center](#) in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide* for information about creating users and groups in IAM Identity Center.

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 user group, or list all user groups that a user is a member of. You can rename or change the path of an IAM user. If you are moving to using federated identities instead of IAM users, you can delete an IAM user from your AWS account, or deactivate the user.

For more information about adding, changing, or removing managed policies for an IAM user, see [Changing permissions for an IAM user \(p. 88\)](#). For information about managing inline policies for IAM users, see [Adding and removing IAM identity permissions \(p. 598\)](#), [Editing IAM policies \(p. 609\)](#), and [Deleting IAM policies \(p. 613\)](#). As a best practice, use managed policies instead of inline policies. *AWS managed policies* grant permissions for many common use cases. Keep in mind that AWS managed policies might not grant least-privilege permissions for your specific use cases because they are available for use by all AWS customers. As a result, we recommend that you reduce permissions further by defining [customer managed policies](#) that are specific to your use cases. For more information, see [AWS managed policies \(p. 495\)](#). For more information about AWS managed policies that are designed for specific job functions, see [AWS managed policies for job functions \(p. 1329\)](#).

To learn about validating IAM policies, see [Validating IAM policies \(p. 588\)](#).

Tip

[IAM Access Analyzer](#) can analyze the services and actions that your IAM roles use, and then generate a fine-grained policy that you can use. After you test each generated policy, you can deploy the policy to your production environment. This ensures that you grant only the required

permissions to your workloads. For more information about policy generation, see [IAM Access Analyzer policy generation](#).

For information about managing IAM user passwords, see [Managing passwords for IAM users \(p. 97\)](#).

Topics

- [View user access \(p. 85\)](#)
- [Listing IAM users \(p. 85\)](#)
- [Renaming an IAM user \(p. 85\)](#)
- [Deleting an IAM user \(p. 86\)](#)
- [Deactivating an IAM user \(p. 88\)](#)

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 \(p. 616\)](#).

Listing IAM users

You can list the IAM users in your AWS account or in a specific IAM user group, and list all the user 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 \(p. 662\)](#).

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 user group

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

To list all the user 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 \(p. 662\)](#).

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 user groups under the new name.
- The unique ID for the user remains the same. For more information about unique IDs, see [Unique identifiers \(p. 1218\)](#).
- 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 AWS account if that user quits your company. If the user is away temporarily, you can deactivate the user's access instead of deleting them from the account as described in [Deactivating an IAM user \(p. 88\)](#).

Topics

- [Deleting an IAM user \(console\) \(p. 86\)](#)
- [Deleting an IAM user \(AWS CLI\) \(p. 87\)](#)

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 user group memberships—that is, the user is removed from any IAM user 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 user group permissions are not affected)

Note

IAM removes any managed policies attached to the user when you delete the user, but does not delete managed policies.

- 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.
3. At the top of the page, choose **Delete**.
4. In the confirmation dialog box, enter the username in the text input field to confirm the deletion of the user. Choose **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.

[aws iam list-service-specific-credentials](#) (to list the user's git credentials) and [aws iam delete-service-specific-credential](#)

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 user groups.

[aws iam list-groups-for-user](#) (to list the user groups to which the user belongs) and [aws iam remove-user-from-group](#)

10. Delete the user.

[aws iam delete-user](#)

Deactivating an IAM user

You might need to deactivate an IAM user while they are temporarily away from your company. You can leave their IAM user credentials in place and still block their AWS access.

To deactivate a user, create and attach a policy to deny the user access to AWS. You can restore the user's access later.

Here are two examples of deny policies that you can attach to a user to deny their access.

The following policy does not include a time limit. You must remove the policy to restore the user's access.

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

The following policy includes a condition that starts the policy on December 24, 2024 at 11:59 PM (UTC) and ends it on February 28, 2025 at 11:59 PM (UTC).

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Deny",  
            "Action": "*",  
            "Resource": "*",  
            "Condition": {  
                "DateGreaterThanOrEqualTo": {"aws:CurrentTime": "2024-12-24T23:59:59Z"},  
                "DateLessThanOrEqualTo": {"aws:CurrentTime": "2025-02-28T23:59:59Z"}  
            }  
        }  
    ]  
}
```

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 \(p. 501\)](#). 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 \(p. 662\)](#).

Topics

- [View user access \(p. 89\)](#)
- [Generate a policy based on a user's access activity \(p. 89\)](#)
- [Adding permissions to a user \(console\) \(p. 89\)](#)
- [Changing permissions for a user \(console\) \(p. 91\)](#)
- [Removing a permissions policy from a user \(console\) \(p. 92\)](#)
- [Removing the permissions boundary from a user \(console\) \(p. 93\)](#)
- [Adding and removing a user's permissions \(AWS CLI or AWS API\) \(p. 93\)](#)

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 \(p. 616\)](#).

Generate a policy based on a user's access activity

You might sometimes grant permissions to an IAM entity (user or role) beyond what they require. To help you refine the permissions that you grant, you can generate an IAM policy that is based on the access activity for an entity. IAM Access Analyzer reviews your AWS CloudTrail logs and generates a policy template that contains the permissions that have been used by the entity in your specified date range. You can use the template to create a managed policy with fine-grained permissions and then attach it to the IAM entity. That way, you grant only the permissions that the user or role needs to interact with AWS resources for your specific use case. To learn more, see [Generate policies based on access activity \(p. 589\)](#).

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. For easier permissions management, 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 **User 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 AWS STS quotas \(p. 1220\)](#). 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 \(p. 581\)](#).
- c. Choose **Create user group**.
- d. Return to the original tab, refresh your list of groups. Then select the check box for your new group.
8. Choose **Next** 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 doesn'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** 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 policies directly**.
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 \(p. 581\)](#).
5. Choose **Next** 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 permissions 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 **Policy usage** 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 and resolve any [policy validation \(p. 588\)](#) recommendations. For more information, see [Editing IAM policies \(p. 609\)](#).
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 **Set boundary**.

Removing a permissions policy from a user (console)

Removing a policy affects the user immediately.

To remove 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 remove permissions by removing an existing policy, view the **Type** to understand how the user is getting that policy before choosing **Remove** to remove the policy:
 - If the policy applies because of group membership, then choosing **Remove** 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 **Remove** 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 boundary** 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 user group \(p. 179\)](#)
- [Adding and removing IAM identity permissions \(p. 598\)](#)

Managing user passwords in AWS

You can manage passwords 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, you have the option of assigning IAM users [access keys \(p. 103\)](#). However, there are other more secure alternatives to access keys that we recommend you consider first. For more information, see [AWS security credentials \(p. 1019\)](#).

Contents

- [Setting an account password policy for IAM users \(p. 94\)](#)
- [Managing passwords for IAM users \(p. 97\)](#)
- [Permitting IAM users to change their own passwords \(p. 100\)](#)
- [How an IAM user changes their own password \(p. 102\)](#)

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 \(p. 95\)](#).

Topics

- [Rules for setting a password policy \(p. 94\)](#)
- [Permissions required to set a password policy \(p. 94\)](#)
- [Default password policy \(p. 95\)](#)
- [Custom password policy options \(p. 95\)](#)
- [Setting a password policy \(console\) \(p. 96\)](#)
- [Setting a password policy \(AWS CLI\) \(p. 97\)](#)
- [Setting a password policy \(AWS API\) \(p. 97\)](#)

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 \(p. 114\)](#).

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 using the JSON editor" \(p. 583\)](#).

```
{  
    "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 \(p. 100\)](#).

Default password policy

If an administrator does not set a custom password policy, IAM user passwords must meet the default AWS password policy.

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 in this section. 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.

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 non-alphanumeric character (! @ # \$ % ^ & * () _ + - = [] { } | ')
- Not be identical to your AWS account name or email address
- Never expire password

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 the Latin alphabet (A-Z)
 - Require at least one lowercase letter from the Latin alphabet (a-z)
 - Require at least one number
 - Require at least one nonalphanumeric character ! @ # \$ % ^ & * () _ + - = [] { } | '
- **Turn on 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, if you specify an expiration of 90 days, it immediately impacts all of your users. For users with passwords older than 90 days, when they log into the console after the change, they must set a new password. Users with passwords 75-89 days old receive an AWS Management Console warning about their 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 using the AWS Management Console to update 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. Administrators with `iam:UpdateLoginProfile` permission can reset IAM user passwords. IAM users with `iam:ChangePassword` permission and active access keys can reset their own IAM user console password 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 change their own password. This gives users access to the `iam:ChangePassword` action for only their user and to the `iam:GetAccountPasswordPolicy` action. This option does not attach a permissions policy to each user. Rather, IAM applies the permissions at the account-level for all users. Alternatively, you can allow only some users to manage their own passwords. 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 \(p. 100\)](#).
- **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 **Edit**.
4. Choose **Custom** to use a custom password policy.
5. Select the options that you want to apply to your password policy and choose **Save changes**.
6. Confirm that you want to set a custom password policy by choosing **Set custom**.

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 **Edit**.
4. Select the options that you want to apply to your password policy and choose **Save changes**.
5. Confirm that you want to set a custom password policy by choosing **Set custom**.

To delete 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 **Edit**.

4. Choose **IAM default** to delete the custom password policy and choose **Save changes**.
5. Confirm that you want to set the IAM default password policy by choosing **Set default**.

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 [How to sign in to AWS](#) in the *AWS Sign-In User Guide*.

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 \(users, user groups, and roles\) \(p. 70\)](#). For information about using policies to set permissions, see [Changing permissions for an IAM user \(p. 88\)](#).

You can grant users permission to change their own passwords. For more information, see [Permitting IAM users to change their own passwords \(p. 100\)](#). For information about how users access your account sign-in page, see [How to sign in to AWS](#) in the *AWS Sign-In User Guide*.

Topics

- [Creating, changing, or deleting an IAM user password \(console\) \(p. 98\)](#)
- [Creating, changing, or deleting an IAM user password \(AWS CLI\) \(p. 99\)](#)

- [Creating, changing, or deleting an IAM user password \(AWS API\) \(p. 100\)](#)

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 **Console sign-in**, choose **Enable console access**.
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 \(p. 94\)](#).

7. To require the user to create a new password when signing in, choose **User must create a new password at next sign-in**. Then choose **Apply**.

Important

If you select the **User must create a new password at next sign-in** 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 \(p. 100\)](#).

8. If you choose the option to generate a password, choose **Show** in the **Console 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 **Console sign-in**, choose **Manage console access**.
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 \(p. 94\)](#), if one is currently set.

7. To require the user to create a new password when signing in, choose **User must create a new password at next sign-in**. Then choose **Apply**.

Important

If you select the **User must create a new password at next sign-in** 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 \(p. 100\)](#).

8. If you choose the option to generate a password, choose **Show** in the **Console 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 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 **Console sign-in**, choose **Manage console access**.
5. For **Console access**, choose **Disable**, and then choose **Apply**.

Important

You can prevent an IAM user from accessing the AWS Management Console by removing their password. This prevents them from signing in to the AWS Management Console using their sign-in credentials. It does not change their permissions or prevent them from accessing the console using an assumed role. If the user has active access keys, they continue to function and allow access through the AWS CLI, Tools for Windows PowerShell, AWS API, or the AWS Console Mobile Application.

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 \(AWS CLI\) \(p. 87\)](#).

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 \(AWS CLI\) \(p. 87\)](#).

Permitting IAM users to change their own passwords

Note

Users with federated identities will use the process defined by their identity provider to change their passwords. As a [best practice \(p. 1032\)](#), require human users to use federation with an identity provider to access AWS using temporary credentials.

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 \(p. 101\).](#)

- [Allow only selected IAM users to change their own passwords \(p. 101\)](#). 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 \(p. 94\)](#) 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 choose **Save changes**. This allows all users in the account access to the iam:ChangePassword action for only their user and to the iam:GetAccountPasswordPolicy action.
5. Provide users with the following instructions for changing their passwords: [How an IAM user changes their own password \(p. 102\)](#).

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 \(AWS CLI\) \(p. 97\)](#).

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. (Optional) 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 [Managing IAM user groups \(p. 178\)](#).
6. Assign the following policy to the group. For more information, see [Managing IAM policies \(p. 581\)](#).

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "iam:GetAccountPasswordPolicy",  
            "Resource": "*"  
        },  
        {  
            "Effect": "Allow",  
            "Action": "iam:ChangePassword",  
            "Resource": "arn:aws:iam::*: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 \(p. 102\)](#).

For more information

For more information on managing credentials, see the following topics:

- [Permitting IAM users to change their own passwords \(p. 100\)](#)
- [Managing user passwords in AWS \(p. 93\)](#)
- [Setting an account password policy for IAM users \(p. 94\)](#)
- [Managing IAM policies \(p. 581\)](#)
- [How an IAM user changes their own password \(p. 102\)](#)

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 \(p. 102\)](#)
- [How IAM users change their own password \(console\) \(p. 102\)](#)
- [How IAM users change their own password \(AWS CLI or AWS API\) \(p. 103\)](#)

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 \(p. 541\)](#).

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](#).

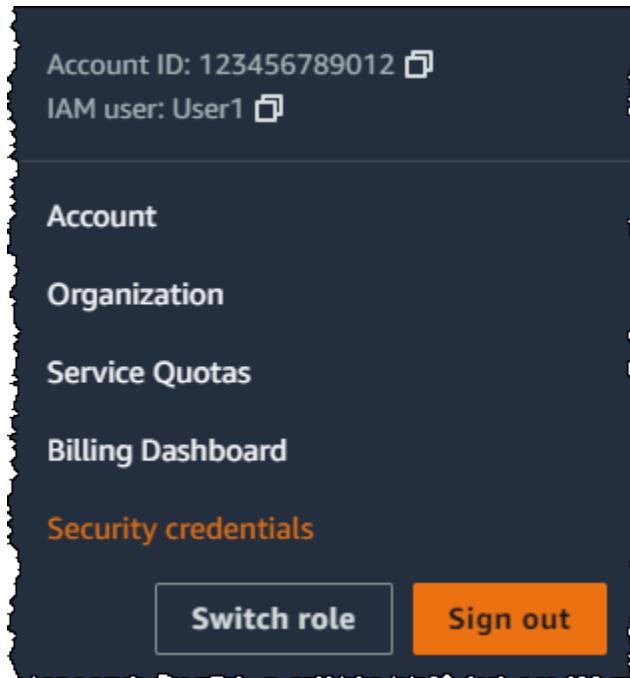
Note

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 **Security credentials**.



3. On the **AWS IAM credentials** tab, choose **Update password**.
4. For **Current password**, enter your current password. Enter a new password for **New password** and **Confirm new password**. Then choose **Update 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 \(p. 94\)](#).

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*.

Important

As a [best practice \(p. 1032\)](#), use temporary security credentials (IAM roles) instead of creating long-term credentials like access keys, and don't create AWS account root user access keys. We don't recommend generating access keys for your root user, because they allow full access to all your resources for all AWS services, including your billing information. For more information, see [Best Practices for AWS accounts](#) in the *AWS Account Management Reference Guide*.

Access keys consist of two parts: an access key ID (for example, AKIAIOSFODNN7EXAMPLE) and a secret access key (for example, wJalrXUtnFEMI/K7MDENG/bPxRfCYEXAMPLEKEY). You must use both the access key ID and secret access key together to authenticate your requests.

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. To follow best practices, rotate the access keys regularly. For more information, see [Rotating access keys \(p. 108\)](#).

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 \(p. 112\)](#).

Important

Manage your access keys securely. Do not provide your access keys to unauthorized parties, even to help [find your account identifiers](#). By doing this, you might give someone permanent access to your account.

Topics

- [Permissions required \(p. 104\)](#)
- [Managing access keys \(console\) \(p. 105\)](#)
- [Managing access keys \(AWS CLI\) \(p. 108\)](#)
- [Managing access keys \(AWS API\) \(p. 108\)](#)
- [Rotating access keys \(p. 108\)](#)
- [Auditing access keys \(p. 112\)](#)

Permissions required

Note

`iam:TagUser` is an optional permission for adding and editing descriptions for the access key. For more information, see [Tagging IAM users \(p. 402\)](#)

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",  
                "iam:TagUser"  
            ]  
        }  
    ]  
}
```

```
        ],
      "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",
        "iam:TagUser"
      ],
      "Resource": "arn:aws:iam::*:user/${aws:username}"
    }
  ]
}
```

Managing access keys (console)

You can use the AWS Management Console to manage the access keys of an IAM user.

To create, modify, or delete your own IAM user access keys (console)

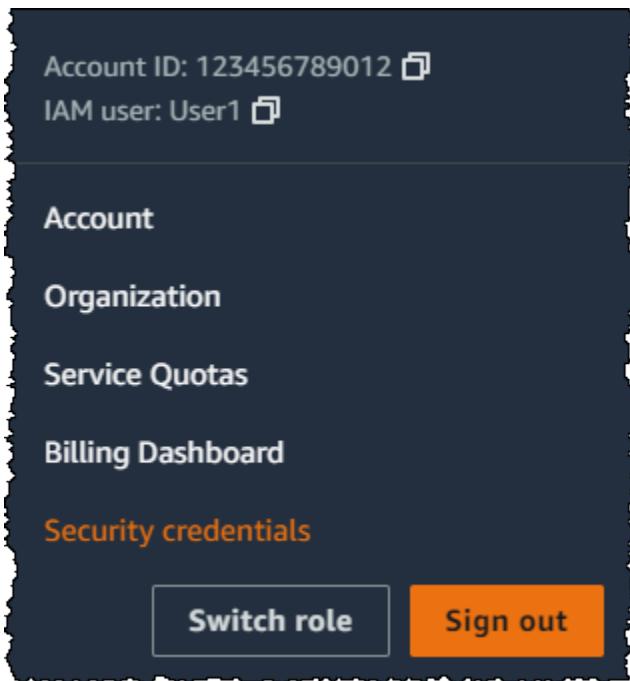
1. Use your AWS account ID or account alias, your IAM user name, and your password to sign in to the [IAM console](#).

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 **Security credentials**.



Do one of the following:

To create an access key

1. In the **Access keys** section, choose **Create access key**. If you already have two access keys, this button is deactivated and you must delete an access key before you can create a new one.
2. On the **Access key best practices & alternatives** page, choose your use case to learn about additional options which can help you avoid creating a long-term access key. If you determine that your use case still requires an access key, choose **Other** and then choose **Next**.
3. (Optional) Set a description tag value for the access key. This adds a tag key-value pair to your IAM user. This can help you identify and rotate access keys later. The tag key is set to the access key id. The tag value is set to the access key description that you specify. When you are finished, choose **Create access key**.
4. On the **Retrieve access keys** page, choose either **Show** to reveal the value of your user's secret access key, or **Download .csv file**. This is your only opportunity to save your secret access key. After you've saved your secret access key in a secure location, choose **Done**.

To deactivate an access key

- In the **Access keys** section find the key you want to deactivate, then choose **Actions**, then choose **Deactivate**. When prompted for confirmation, choose **Deactivate**. A deactivated access key still counts toward your limit of two access keys.

To activate an access key

- In the **Access keys** section, find the key to activate, then choose **Actions**, then choose **Activate**.

To delete an access key when you no longer need it

- In the **Access keys** section, find the key you want to delete, then choose **Actions**, then choose **Delete**. Follow the instructions in the dialog to first **Deactivate** and then confirm the deletion. We recommend that you verify that the access key is no longer in use before you permanently delete it.

To create, modify, or delete the access keys of 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 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**. If the button is deactivated, then you must delete one of the existing keys before you can create a new one. On the **Access key best practices & alternatives** page, review the best practices and alternatives. Choose your use case to learn about additional options which can help you avoid creating a long-term access key. If you determine that your use case still requires an access key, choose **Other** and then choose **Next**. On the **Retrieve access key page**, choose **Show** to reveal the value of your user's secret access key. To save the access key ID and secret access key to a .csv file to a secure location on your computer, choose the **Download .csv file** button. When you create an access key for your user, that key pair is active by default, and your user can use the pair right away.
 - To deactivate an active access key, choose **Actions**, and then choose **Deactivate**.
 - To activate an inactive access key, choose **Actions**, and then choose **Activate**.
 - To delete your access key, choose **Actions**, and then choose **Delete**. Follow the instructions in the dialog to first **Deactivate** and then confirm the deletion. 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. In the **Access keys** section, you will see the user's access keys and the status of each key 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 the IAM user access keys from the AWS CLI, run the following commands.

- To create an access key: [aws iam create-access-key](#)
- To deactivate or activate 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 the access keys of an IAM user from the AWS API, call the following operations.

- To create an access key: [CreateAccessKey](#)
- To deactivate or activate 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. Regularly rotating long-term credentials helps you familiarize yourself with the process. This is useful in case you are ever in a situation where you must rotate credentials, such as when an employee leaves your

company. 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 \(p. 542\)](#). 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 \(p. 94\)](#).

Important

When you create an AWS account, you begin with one 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 don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see [Tasks that require root user credentials](#) in the *AWS Account Management Reference Guide*. Because the AWS account root user credentials are long-term 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. You must use the root user credentials to change the root user credentials.

Topics

- [Rotating IAM user access keys \(console\) \(p. 109\)](#)
- [Rotating access keys \(AWS CLI\) \(p. 110\)](#)
- [Rotating access keys \(AWS API\) \(p. 111\)](#)

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. In the **Access keys** section, choose **Create access key**. On the **Access key best practices & alternatives** page, choose **Other**, then choose **Next**.
 - e. (Optional) Set a description tag value for the access key to add a tag key-value pair to this IAM user. This can help you identify and rotate access keys later. The tag key is set to the access key id. The tag value is set to the access key description that you specify. When you are finished, choose **Create access key**.
 - f. On the **Retrieve access keys** page, choose either **Show** to reveal the value of your user's secret access key, or **Download .csv file**. This is your only opportunity to save your secret access key. After you've saved your secret access key in a secure location, choose **Done**.

When you create an access key for your user, that key pair is active by default, and your user can use the pair right away. 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** information 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** information indicates that the old key has never been used, we recommend that you do not immediately delete the first access key. Instead, choose **Actions** and then choose **Deactivate** 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 reactivate the first access key. Then return to [Step 3 \(p. 109\)](#) 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. In the **Access keys** section for the access key you want to delete, choose **Actions**, and then choose **Delete**. Follow the instructions in the dialog to first **Deactivate** and then confirm the deletion.

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 reactivate 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 \(p. 1032\)](#). 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 reactivate 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 \(p. 1032\)](#). 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 \(p. 164\)](#) 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 \(p. 473\)](#) to learn who performed an action in AWS. You can use the `sts:SourceIdentity` condition key in the role trust policy to require users to specify an identity when they assume a role. For example, you can require that IAM users specify their own user name as their source identity. This can help you determine which user performed a specific action in AWS. For more information, see [sts:SourceIdentity \(p. 1382\)](#).

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

Important

Having trouble signing in to AWS? Make sure that you're on the correct [AWS sign-in page](#) 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 choose **Still unable to log into your AWS account** and then choose one of the available support options.

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](#).

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” \(p. 113\)](#) 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 \(p. 470\)](#) and [Deleting access keys for the root user \(p. 470\)](#).
- **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 \(p. 97\)](#).
- **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 \(console\) \(p. 105\)](#). 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 \(p. 103\)](#).

You should follow the AWS [best practice \(p. 1034\)](#) 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 \(p. 108\)](#).

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?**.

Note

If you are an IAM user, this option is not available. The **Forgot your password?** option is only available for the root user account. You must ask your administrator to reset 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 @verify.signin.aws. 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](#) in the [AWS Sign-In User Guide](#).

Using multi-factor authentication (MFA) in AWS

 [Follow us on Twitter](#)

For increased security, we recommend that you configure multi-factor authentication (MFA) to help protect your AWS resources. You can enable MFA for the AWS account root user and IAM users. 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. You can register up to eight MFA devices of any combination of the currently supported MFA types with your AWS account root user and IAM users. For more information about supported MFA types see [What is MFA? \(p. 114\)](#). With multiple MFA devices, only one MFA device is needed to sign in to the AWS Management Console or create a session through the AWS CLI as that user.

Note

We recommend that you require your human users to use temporary credentials when accessing AWS. Have you considered using AWS IAM Identity Center (successor to AWS Single Sign-On)? You can use IAM Identity Center to centrally manage access to multiple AWS accounts and provide users with MFA-protected, single sign-on access to all their assigned accounts from one place. With IAM Identity Center, you can create and manage user identities in IAM Identity Center or easily connect to your existing SAML 2.0 compatible identity provider. For more information, see [What is IAM Identity Center?](#) in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*.

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:

- **FIDO security key** – FIDO Certified hardware security keys are provided by third-party providers. The FIDO Alliance maintains a list of all [FIDOCertified products](#) that are compatible with FIDO specifications. FIDO authentication standards are based on public key cryptography, which enables strong, phishing-resistant authentication that is more secure than passwords. FIDO security keys support multiple root accounts and IAM users using a single security key. For more information about enabling FIDO security keys, see [Enabling a FIDO security key \(console\) \(p. 120\)](#).
- **Virtual MFA devices** – A virtual authenticator application that runs on a phone or other device and emulates a physical device. Virtual authenticator apps implement the [time-based one-time password](#) (TOTP) algorithm and support multiple tokens on a single device. 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 can't 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 FIDO security keys. 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 \(console\) \(p. 116\)](#).
- **Hardware TOTP token** – A hardware device that generates a six-digit numeric code based on the [time-based one-time password](#) (TOTP) 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 TOTP token with AWS, see [Enabling a hardware TOTP token \(console\) \(p. 129\)](#).

Note

SMS text message-based MFA – AWS ended support for enabling SMS multi-factor authentication (MFA). We recommend that customers who have IAM users that use SMS text message-based MFA switch to one of the following alternative methods: [FIDO](#)

[security key \(p. 120\)](#), [virtual \(software-based\) MFA device \(p. 116\)](#), or [hardware MFA device \(p. 129\)](#). You can identify the 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.

Topics

- [Enabling MFA devices for users in AWS \(p. 115\)](#)
- [Checking MFA status \(p. 136\)](#)
- [Resynchronizing virtual and hardware MFA devices \(p. 137\)](#)
- [Deactivating MFA devices \(p. 141\)](#)
- [What if an MFA device is lost or stops working? \(p. 143\)](#)
- [Configuring MFA-protected API access \(p. 145\)](#)
- [Sample code: Requesting credentials with multi-factor authentication \(p. 151\)](#)

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 \(p. 115\)](#)
- [Enabling a virtual multi-factor authentication \(MFA\) device \(console\) \(p. 116\)](#)
- [Enabling a FIDO security key \(console\) \(p. 120\)](#)
- [Enabling a hardware TOTP token \(console\) \(p. 129\)](#)
- [Enabling and managing virtual MFA devices \(AWS CLI or AWS API\) \(p. 134\)](#)

General steps for enabling MFA devices

The following overview procedure describes how to set up and use MFA and provides links to related information.

Note

You can also watch this English-language video, [How to Setup AWS Multi-Factor Authentication \(MFA\) and AWS Budget Alerts](#), for more information.

1. *Get an MFA device such as one of the following.* You can enable up to eight MFA devices per AWS account root user or IAM user of any combination of the following types.
 - 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 FIDO security key with an [AWS supported configuration \(p. 125\)](#). The FIDO Alliance maintains a list of all [FIDO Certified products](#) that are compatible with FIDO specifications.
 - A hardware-based MFA device from a third-party provider like a token device. These tokens are used exclusively with AWS accounts. For more information, see [Enabling a hardware TOTP token \(console\) \(p. 129\)](#). You can purchase these tokens directly from the manufacturers as a key fob or display card device.
2. *Enable the MFA device.*
 - **Virtual or Hardware TOTP tokens** –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.

- **FIDO security keys** – Root users and IAM users with FIDO security keys can enable from the AWS Management Console only, not from the AWS CLI or AWS API.

For information about enabling each type of MFA device, see the following pages:

- Virtual MFA device: [Enabling a virtual multi-factor authentication \(MFA\) device \(console\) \(p. 116\)](#)
- FIDO security key: [Enabling a FIDO security key \(console\) \(p. 120\)](#)
- Hardware TOTP token: [Enabling a hardware TOTP token \(console\) \(p. 129\)](#)

3. *Enable Multiple MFA devices (recommended)*

- We recommend that you enable multiple MFA devices to the AWS account root user and IAM users in your AWS accounts. This allows you to raise the security bar in your AWS accounts and simplify managing access to highly privileged users, such as the AWS account root user.
- You can register up to **eight** MFA devices of any combination of the [currently supported MFA types](#) with your AWS account root user and IAM users. With multiple MFA devices, you only need one MFA device to sign in to the AWS Management Console or create a session through the AWS CLI as that user. An IAM user must authenticate with an existing MFA device to enable or disable an additional MFA device.
- In the event of a lost, stolen, or inaccessible MFA device you can use one of the remaining MFA devices to access the AWS account without performing the AWS account recovery procedure. If an MFA device is lost or stolen, it should be disassociated from the IAM principal with which it is associated.
- The use of multiple MFAs allows your employees in geographically dispersed locations or working remotely to use hardware-based MFA to access AWS without having to coordinate the physical exchange of a single hardware device between employees.
- The use of additional MFA devices for IAM principals allows you to use one or more MFAs for everyday usage, while also maintaining physical MFA devices in a secure physical location such as a vault or safe for backup and redundancy.

4. *Use the MFA device when you log in to or access AWS resources.* Note the following:

- **FIDO security keys** – To access an AWS website, enter your credentials and then tap the FIDO security key when prompted.
- **Virtual MFA devices and hardware TOTP tokens** – To access an AWS website, you need an MFA code from the device in addition to your user name and password.

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 device defined in AWS)
- The usual access key ID and secret access key

Notes

- You cannot pass the MFA information for a FIDO security key to AWS STS API operations to request temporary credentials.
- You cannot use AWS CLI commands or AWS API operations to enable [FIDO security keys \(p. 120\)](#).
- You cannot use the same name for more than one root or IAM MFA device.

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 FIDO security keys. 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. You can register up to **eight** MFA devices of any combination of the [currently supported MFA types](#) with your AWS account root user and IAM users. With multiple MFA devices, you only need one MFA device to sign in to the AWS Management Console or create a session through the AWS CLI as that 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 \(p. 117\)](#)
- [Enable a virtual MFA device for an IAM user \(console\) \(p. 117\)](#)
- [Enable a virtual MFA device for your AWS account root user \(console\) \(p. 118\)](#)
- [Replace or "rotate" a virtual MFA device \(p. 120\)](#)

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 \(p. 539\)](#).

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. You can attach tags to your IAM resources, including virtual MFA devices, to identify, organize, and control access to them. You can tag virtual MFA devices only when you use the AWS CLI or AWS API. To enable and manage an MFA device using the AWS CLI or AWS API, see [Enabling and managing virtual MFA devices \(AWS CLI or AWS API\) \(p. 134\)](#). For more information about tagging IAM resources, see [Tagging IAM resources \(p. 399\)](#).

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 the [IAM tutorial: Permit users to manage their credentials and MFA settings \(p. 66\)](#) and example policy [AWS: Allows MFA-authenticated IAM users to manage their own MFA device on the My security credentials page \(p. 539\)](#).

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 **Users** list, choose the name of the IAM user.
4. Choose the **Security Credentials** tab. Under **Multi-factor authentication (MFA)**, choose **Assign MFA device**.
5. In the wizard, type a **Device name**, choose **Authenticator app**, and then choose **Next**.

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.
- From the 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. On the **Set up device** page, 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 **Add 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 \(p. 137\)](#).

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 \(p. 83\)](#).

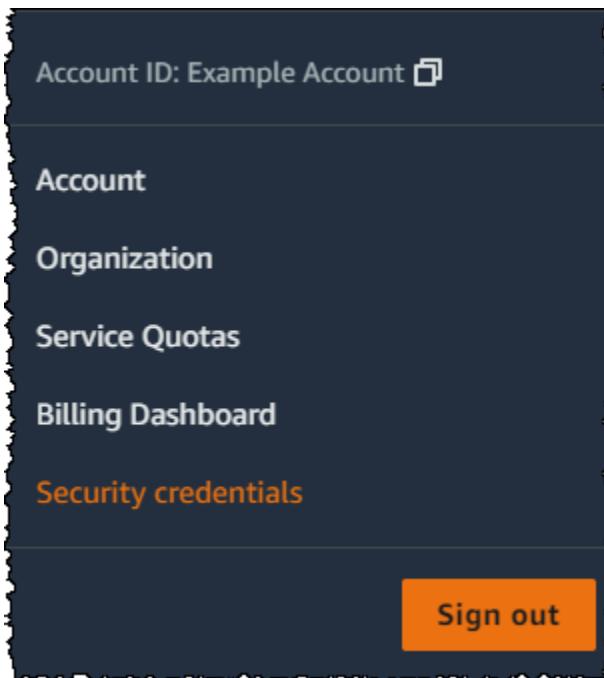
Enable a virtual MFA device for your AWS account root user (console)

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? \(p. 143\)](#).

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 **Security credentials**. If necessary, choose **Continue to Security credentials**.



3. In the **Multi-Factor Authentication (MFA)** section, choose **Assign MFA device**.
4. In the wizard, type a **Device name**, choose **Authenticator app**, and then choose **Next**.

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 **Set up 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 MFA devices for your account. You can register up to **eight** MFA devices of any combination of the [currently supported MFA types](#) with your AWS account root user and IAM users. A virtual MFA device might become unavailable, for example, if you lose the smartphone where the virtual MFA device is hosted. If that happens and you are not able to sign in to your account with no additional MFA devices attached to the user or even by [Recovering a root user MFA device \(p. 143\)](#), 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 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 **Add MFA**.

Important

Submit your request immediately after generating the code. 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 \(p. 137\)](#).

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 register up to **eight** MFA devices of any combination of the [currently supported MFA types](#) with your AWS account root user and IAM users. 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 \(p. 141\)](#).
- 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 \(console\) \(p. 117\)](#) 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 \(console\) \(p. 118\)](#) earlier in this topic.

Enabling a FIDO security key (console)

FIDO security keys are a type of [multi-factor authentication \(MFA\) device \(p. 114\)](#) that you can use to protect your AWS resources. You plug your FIDO 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 FIDO security key with other services, and it has an [AWS supported configuration \(p. 125\)](#) (for example, the YubiKey 5 Series from Yubico), you can also use it with AWS. Otherwise, you need to purchase a FIDO security key if you want to use WebAuthn for MFA in AWS. For specifications and purchase information, see [Multi-Factor Authentication](#).

FIDO2 is an open authentication standard and an extension of FIDO U2F, offering the same high level of security based on public key cryptography. FIDO2 consists of the W3C Web Authentication specification (WebAuthn API) and the FIDO Alliance Client-to-Authenticator Protocol (CTAP), an application layer protocol. CTAP enables communication between client or platform, like a browser or operating system, with an external authenticator. When you enable a FIDO Certified authenticator in AWS, the FIDO security key creates a new key pair for use with only AWS. First, you enter your credentials. When prompted, you tap the FIDO security key, which responds to the authentication challenge issued by AWS. To learn more about the FIDO2 standard, see the [FIDO2 Project](#).

You can register up to **eight** MFA devices of any combination of the [currently supported MFA types](#) with your AWS account root user and IAM users. With multiple MFA devices, you only need one MFA device to sign in to the AWS Management Console or create a session through the AWS CLI as that user.

Note

We recommend that you require your human users to use temporary credentials when accessing AWS. Your users can federate into AWS with an identity provider where they authenticate with

their corporate credentials and MFA configurations. To manage access to AWS and business applications, we recommend that you use IAM Identity Center. For more information, see the [The IAM Identity Center User Guide](#).

Topics

- [Permissions required \(p. 121\)](#)
- [Enable a FIDO security key for your own IAM user \(console\) \(p. 121\)](#)
- [Enable a FIDO security key for another IAM user \(console\) \(p. 123\)](#)
- [Enable a FIDO security key for the AWS account root user \(console\) \(p. 123\)](#)
- [Replace a FIDO security key \(p. 124\)](#)
- [Supported configurations for using FIDO security keys \(p. 125\)](#)

Permissions required

To manage a FIDO security key for your own IAM user while protecting sensitive MFA-related actions, you must have the permissions from the following policy:

Note

The ARN values are static values and are not an indicator of what protocol was used to register the authenticator. We have deprecated U2F, so all new implementations use WebAuthn.

```
{  
    "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": "*",  
            "Condition": {  
                "BoolIfExists": {  
                    "aws:MultiFactorAuthPresent": "false"  
                }  
            }  
        }  
    ]  
}
```

Enable a FIDO security key for your own IAM user (console)

You can enable a FIDO 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 FIDO security key, you must have physical access to the device.

Note

You should not choose any of the available options on the Google Chrome pop-up that asks to **Verify your identity with amazon.com**. You only need to tap on the security key.

To enable a FIDO security key for your own IAM user (console)

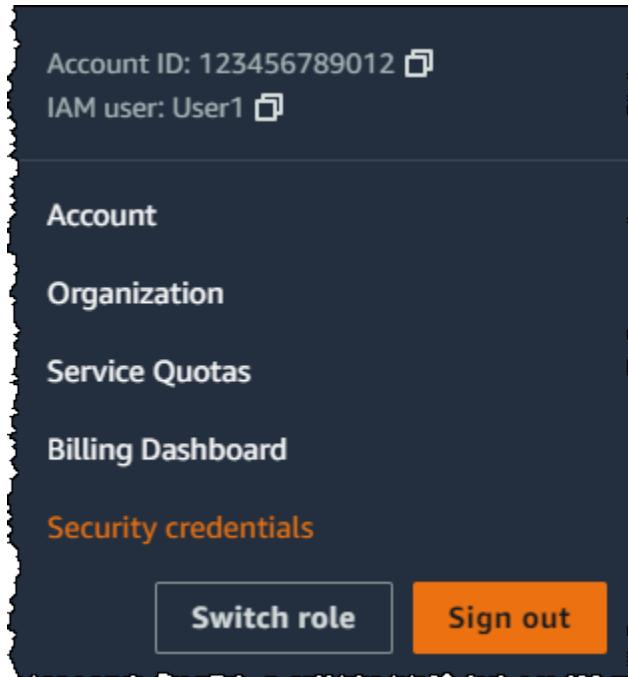
1. Use your AWS account ID or account alias, your IAM user name, and your password to sign in to the [IAM console](#).

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 **Security credentials**.



3. On the AWS IAM credentials tab, in the **Multi-factor authentication (MFA)** section, choose **Assign MFA device**.
4. In the wizard, type a **Device name**, choose **Security Key**, and then choose **Next**.
5. Insert the FIDO security key into your computer's USB port.



6. Tap the FIDO security key.

The FIDO 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 FIDO security key for another IAM user (console)

You can enable a FIDO security key for another IAM user from the AWS Management Console only, not from the AWS CLI or AWS API.

To enable a FIDO 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.
4. Choose the **Security Credentials** tab. Under **Multi-factor authentication (MFA)**, choose **Assign MFA device**.
5. In the wizard, type a **Device name**, choose **Security Key**, and then choose **Next**.
6. Insert the FIDO security key into your computer's USB port.



7. Tap the FIDO security key.

The FIDO 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 FIDO 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 FIDO security key is lost, stolen, or not working, you can still sign in using another MFA device registered to the same AWS account root user. If you only have a single MFA device registered, you can sign in using alternate factors of identification. To learn about signing in using alternative factors of authentication, see [What if an MFA device is lost or stops working? \(p. 143\)](#). To disable this feature, contact [AWS Support](#).

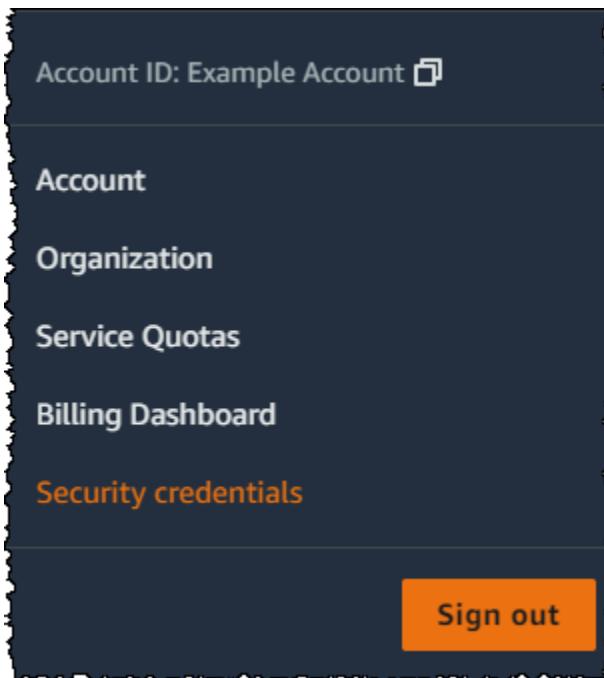
To enable the FIDO key for your root user (console)

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

As the root user, you can't sign in to the **Sign in as IAM user** page. If you see the **Sign in as IAM user** page, choose **Sign in using root user email** near the bottom of the page. For help signing in as the root user, see [Signing in to the AWS Management Console as the root user](#) in the [AWS Sign-In User Guide](#).

2. On the right side of the navigation bar, choose your account name, and then choose **Security credentials**. If necessary, choose **Continue to Security credentials**.



3. Expand the **Multi-factor authentication (MFA)** section.
4. Choose **Assign MFA device**.
5. In the wizard, type a **Device name**, choose **Security Key**, and then choose **Next**.
6. Insert the FIDO security key into your computer's USB port.



7. Tap the FIDO security key.

The FIDO security key is ready for use with AWS. The next time you use your root user credentials to sign in, you must tap your FIDO security key to complete the sign-in process.

Replace a FIDO security key

You can have up to eight MFA devices of any combination of the [currently supported MFA types](#) assigned to a user at a time with your AWS account root user and IAM users. If the user loses a FIDO authenticator or needs to replace it for any reason, you must first deactivate the old FIDO authenticator. Then you can add a new MFA device for the user.

- To deactivate the device currently associated with an IAM user, see [Deactivating MFA devices \(p. 141\)](#).
- To add a new FIDO security key for an IAM user, see [Enable a FIDO security key for your own IAM user \(console\) \(p. 121\)](#).

If you don't have access to a new FIDO security key, you can enable a new virtual MFA device or hardware TOTP token. See one of the following for instructions:

- [Enabling a virtual multi-factor authentication \(MFA\) device \(console\) \(p. 116\)](#)

- [Enabling a hardware TOTP token \(console\) \(p. 129\)](#)

Supported configurations for using FIDO security keys

You can use WebAuthn as a multi-factor authentication (MFA) method with IAM using currently supported configurations. These include FIDO Certified devices supported by IAM and browsers that support WebAuthn.

FIDO devices supported by AWS

IAM currently supports FIDO Certified security devices that connect to your devices through USB, Bluetooth, or NFC. We don't support platform authenticators such as TouchID, FaceID, or Windows Hello.

Note

AWS requires access to the physical USB port on your computer to verify your FIDO Certified device. WebAuthn MFA will not work with a virtual machine, a remote connection, or a browser's incognito mode.

The FIDO Alliance maintains a list of all [FIDO Certified products](#) that are compatible with FIDO specifications.

Browsers that support WebAuthn

The following browsers currently support the use of FIDO Certified security keys:

	macOS 10.15+	Windows 10	Linux
Chrome	Yes	Yes	Yes
Safari	Yes	No	No
Edge	Yes	Yes	Yes
Firefox	Yes	Yes	Yes

Note

Most Firefox versions that currently support WebAuthn don't enable support by default. For instructions on enabling WebAuthn support in Firefox, see [Troubleshooting FIDO security keys \(p. 1194\)](#).

Browser plugins

AWS supports only browsers that natively support the FIDO2 WebAuthn standard. AWS doesn't support using plugins to add FIDO2 WebAuthn browser support. Some browser plugins are incompatible with the FIDO U2F standard and can cause unexpected results with FIDO2 security keys.

For information on disabling browser plugins and other troubleshooting tips, see [I can't enable my FIDO security key \(p. 1194\)](#).

Mobile environments

The following browsers currently support the use of FIDO Certified security keys:

	iOS 14.5+	Android 7+
Chrome	Yes	Yes

	iOS 14.5+	Android 7+
Safari	Yes	No
Edge	No	No
Firefox	Yes	No

Note

The AWS Console Mobile App doesn't support using FIDO Certified security keys for MFA.

Device certifications

We capture and assign device-related certifications, such as FIPS validation and FIDO certification level, only during the registration of a FIDO security key. Your device certification is retrieved from the [FIDO Alliance Metadata Service \(MDS\)](#). If the certification status or level of your FIDO security key changes, it will not be reflected in the device tags automatically. To update the certification information of a device, register the device again to fetch the updated certification information.

AWS provides the following certification types as condition keys during device registration, obtained from the FIDO MDS: FIPS-140-2, FIPS-140-3, and FIDO certification levels. You have the ability to specify the registration of specific authenticators in their IAM policies, based on your preferred certification type and level. For more information, see the policies below.

Example policies for device certifications

The following use cases show sample policies that allow you to register MFA devices with FIPS certifications.

Topics

- [Use case 1: Allow registering only devices that have FIPS-140-2 L2 certifications \(p. 126\)](#)
- [Use case 2: Allow registering devices that have FIPS-140-2 L2 and FIDO L1 certifications \(p. 127\)](#)
- [Use case 3: Allow registering devices that have either FIPS-140-2 L2 or FIPS-140-3 L2 certifications \(p. 127\)](#)
- [Use case 4: Allow registering devices that have FIPS-140-2 L2 certification and support other MFA types like virtual authenticators and hardware TOTP \(p. 128\)](#)

Use case 1: Allow registering only devices that have FIPS-140-2 L2 certifications

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:EnableMFADevice",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:RegisterSecurityKey" : "Create"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "iam:EnableMFADevice",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:RegisterSecurityKey" : "Activate",
                }
            }
        }
    ]
}
```

```
        "iam:FIDO-FIPS-140-2-certification": "L2"
    }
}
]
```

Use case 2: Allow registering devices that have FIPS-140-2 L2 and FIDO L1 certifications

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:EnableMFADevice",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:RegisterSecurityKey" : "Create"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "iam:EnableMFADevice",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:RegisterSecurityKey" : "Activate",
                    "iam:FIDO-FIPS-140-2-certification": "L2",
                    "iam:FIDO-certification": "L1"
                }
            }
        }
    ]
}
```

Use case 3: Allow registering devices that have either FIPS-140-2 L2 or FIPS-140-3 L2 certifications

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:EnableMFADevice",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:RegisterSecurityKey" : "Create"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "iam:EnableMFADevice",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:RegisterSecurityKey" : "Activate",
                    "iam:FIDO-FIPS-140-2-certification": "L2"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "iam:EnableMFADevice",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:RegisterSecurityKey" : "Activate",
                    "iam:FIDO-FIPS-140-3-certification": "L2"
                }
            }
        }
    ]
}
```

```
        "Effect": "Allow",
        "Action": "iam:EnableMFADevice",
        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "iam:RegisterSecurityKey" : "Activate",
                "iam:FIDO-FIPS-140-3-certification": "L2"
            }
        }
    ]
}
```

Use case 4: Allow registering devices that have FIPS-140-2 L2 certification and support other MFA types like virtual authenticators and hardware TOTP

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:EnableMFADevice",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:RegisterSecurityKey": "Create"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "iam:EnableMFADevice",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:RegisterSecurityKey": "Activate",
                    "iam:FIPS-140-2-certification": "L2"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "iam:EnableMFADevice",
            "Resource": "*",
            "Condition": {
                "Null": {
                    "iam:RegisterSecurityKey": "true"
                }
            }
        }
    ]
}
```

AWS CLI and AWS API

AWS supports using FIDO certified security keys only in the AWS Management Console. Using FIDO Certified security keys for MFA is not supported in the [AWS CLI](#) and [AWS API](#), or for access to [MFA-protected API operations \(p. 145\)](#).

Additional resources

- For more information on using FIDO Certified security keys in AWS, see [Enabling a FIDO security key \(console\) \(p. 120\)](#).

- For help with troubleshooting FIDO Certified security keys in AWS, see [Troubleshooting FIDO security keys \(p. 1194\)](#).
 - For general industry information on FIDO2 support, see [FIDO2 Project](#).

Enabling a hardware TOTP token (console)

A hardware TOTP token generates a six-digit numeric code based upon a time-based one-time password (TOTP) 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 TOTP tokens and [FIDO security keys \(p. 120\)](#) are both physical devices that you purchase. The difference is that hardware MFA devices generate a TOTP code that you view and then enter when prompted when signing it to AWS. A FIDO security key is a strong phishing-resistant MFA option where you don't see or type an authentication code. Instead, the FIDO 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 TOTP token 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 TOTP token for the AWS account root user \(console\) \(p. 132\)](#).

You can register up to **eight** MFA devices of any combination of the [currently supported MFA types](#) with your AWS account root user and IAM users. With multiple MFA devices, you only need one MFA device to sign in to the AWS Management Console or create a session through the AWS CLI as that user.

Important

We recommend that you enable multiple MFA devices for your users for continued access to your account in case of a lost or inaccessible MFA device.

Note

If you want to enable the MFA device from the command line, use [aws iam enable-mfa-device](#). To enable the MFA device with the IAM API, use the [EnableMFADevice](#) operation.

Topics

- [Permissions required \(p. 129\)](#)
 - [Enable a hardware TOTP token for your own IAM user \(console\) \(p. 130\)](#)
 - [Enable a hardware TOTP token for another IAM user \(console\) \(p. 131\)](#)
 - [Enable a hardware TOTP token for the AWS account root user \(console\) \(p. 132\)](#)
 - [Replace or "rotate" a physical MFA device \(p. 134\)](#)

Permissions required

To manage a hardware TOTP token 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 TOTP token for your own IAM user (console)

You can enable your own hardware TOTP token from the AWS Management Console.

Note

Before you can enable a hardware TOTP token, you must have physical access to the device.

To enable a hardware TOTP token for your own IAM user (console)

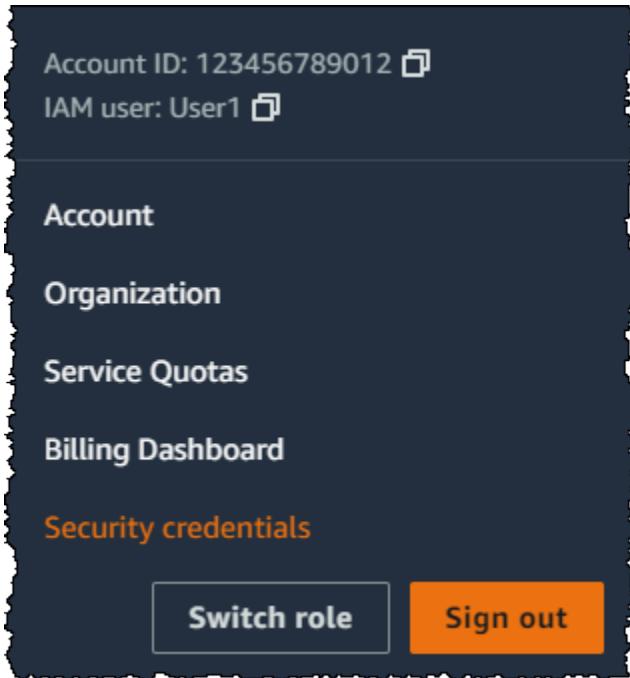
1. Use your AWS account ID or account alias, your IAM user name, and your password to sign in to the [IAM console](#).

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 **Security credentials**.



3. On the **AWS IAM credentials** tab, in the **Multi-factor authentication (MFA)** section, choose **Assign MFA device**.
4. In the wizard, type a **Device name**, choose **Hardware TOTP token**, and then choose **Next**.
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 **Add 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 \(p. 137\)](#).

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 TOTP token for another IAM user \(console\)](#)

You can enable a hardware TOTP token for another IAM user from the AWS Management Console.

To enable a hardware TOTP token 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.
4. Choose the **Security Credentials** tab. Under **Multi-factor authentication (MFA)**, choose **Assign MFA device**.
5. In the wizard, type a **Device name**, choose **Hardware TOTP token**, and then choose **Next**.
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 **Add 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 \(p. 137\)](#).

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 TOTP token for the AWS account root user (console)

You can configure and enable a physical 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? \(p. 143\)](#). 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)

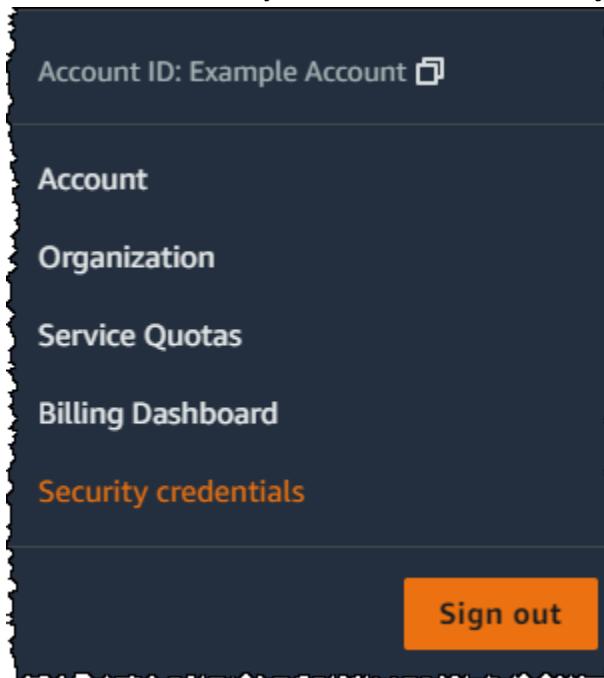
1. Sign in to the [IAM console](#) as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

Note

As the root user, you can't sign in to the **Sign in as IAM user** page. If you see the **Sign in as IAM user** page, choose **Sign in using root user email** near the bottom of the page. For help

signing in as the root user, see [Signing in to the AWS Management Console as the root user](#) in the *AWS Sign-In User Guide*.

2. On the right side of the navigation bar, choose on your account name, and then choose **Security credentials**. If necessary, choose **Continue to Security credentials**.



3. Expand the **Multi-factor authentication (MFA)** section.
4. Choose **Assign MFA device**.
5. In the wizard, type a **Device name**, choose **Hardware TOTP token**, and then choose **Next**.
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 **Add 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 \(p. 137\)](#).

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 up to eight MFA devices of any combination of the [currently supported MFA types](#) assigned to a user at a time with your AWS account root user and IAM users. 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 \(p. 141\)](#).
- To add a replacement hardware TOTP token for an IAM user, follow the steps in the procedure [Enable a hardware TOTP token for another IAM user \(console\) \(p. 131\)](#) earlier in this topic.
- To add a replacement hardware TOTP token for the AWS account root user, follow the steps in the procedure [Enable a hardware TOTP token for the AWS account root user \(console\) \(p. 132\)](#) earlier in this topic.

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.

You can attach tags to your IAM resources, including virtual MFA devices, to identify, organize, and control access to them. You can tag virtual MFA devices only when you use the AWS CLI or AWS API.

An IAM user using the SDK or CLI can enable an additional MFA device by calling [EnableMFADevice](#) or deactivate an existing MFA device by calling [DeactivateMFADevice](#). To do this successfully, they must first call [GetSessionToken](#) and submit MFA codes with an existing MFA device. This call returns temporary security credentials that can then be used to sign API operations that require MFA authentication. For an example request and response, see [GetSessionToken—temporary credentials for users in untrusted environments](#).

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. 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 tag a virtual MFA device

Use these commands to tag a virtual MFA device.

- AWS CLI: [aws iam tag-mfa-device](#)
- AWS API: [TagMFADevice](#)

To list tags for a virtual MFA device

Use these commands to list the tags attached to a virtual MFA device.

- AWS CLI: [aws iam list-mfa-device-tags](#)
- AWS API: [ListMFADeviceTags](#)

To untag a virtual MFA device

Use these commands to remove tags attached to a virtual MFA device.

- AWS CLI: [aws iam untag-mfa-device](#)
- AWS API: [UntagMFADevice](#)

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, a 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. The user must contact an administrator to deactivate the device. For more information, see [What if an MFA device is lost or stops working? \(p. 143\)](#).

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 **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 \(console\) \(p. 118\)](#)
- [Enable a FIDO security key for the AWS account root user \(console\) \(p. 123\)](#)
- [Enable a hardware TOTP token for the AWS account root user \(console\) \(p. 132\)](#)

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 **None**. If the user has an MFA device enabled, the **MFA** column shows the type of device that is enabled with a value of **Virtual**, **Security key**, **Hardware**, or **SMS**.

Note

AWS ended support for enabling SMS multi-factor authentication (MFA). We recommend that customers who have IAM users that use SMS text message-based MFA switch to one of the following alternative methods: [virtual \(software-based\) MFA device \(p. 116\)](#), [FIDO security key \(p. 120\)](#), or [hardware MFA device \(p. 129\)](#). You can identify the 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.
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 MFA devices. Assign an MFA device to improve the security of your AWS environment** in the **Multi-factor authentication (MFA)** section. If the user has MFA devices enabled, the **Multi-factor authentication (MFA)** section shows details about the devices:
 - The device name
 - The device type
 - The identifier for the device, such as a serial number for a physical device or the ARN in AWS for a virtual device
 - When the device was created

To remove or resync a device, choose the radio button next to the device and choose **Remove** or **Resync**.

For more information on enabling MFA, see the following:

- [Enabling a virtual multi-factor authentication \(MFA\) device \(console\) \(p. 116\)](#)
- [Enabling a FIDO security key \(console\) \(p. 120\)](#)
- [Enabling a hardware TOTP token \(console\) \(p. 129\)](#)

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

FIDO security keys do not go out of sync. If a FIDO 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 \(console\) \(p. 142\)](#).

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. If you aren't able to successfully resynchronize your device, you may need to de-associate and re-associate it. For more information on how to do this, see [Deactivating MFA devices \(p. 141\)](#) and [Enabling MFA devices for users in AWS \(p. 115\)](#).

Topics

- [Permissions required \(p. 137\)](#)
- [Resynchronizing virtual and hardware MFA devices \(IAM console\) \(p. 138\)](#)
- [Resynchronizing virtual and hardware MFA devices \(AWS CLI\) \(p. 141\)](#)
- [Resynchronizing virtual and hardware MFA devices \(AWS API\) \(p. 141\)](#)

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)

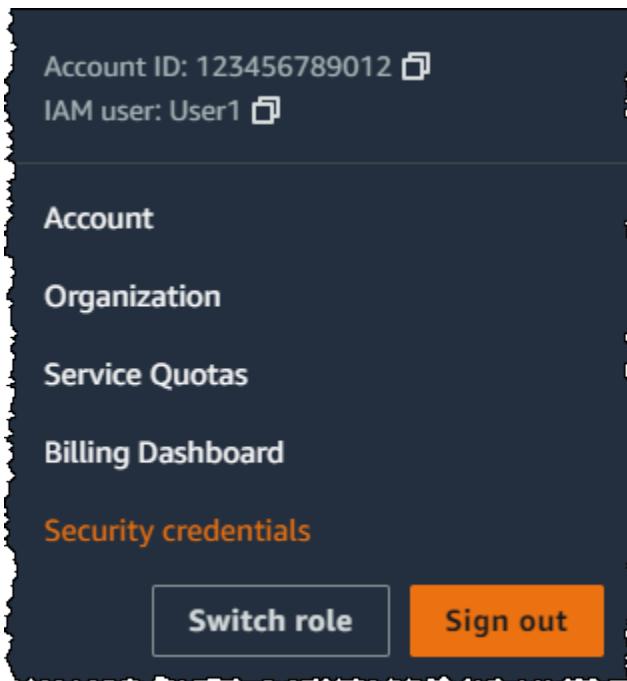
1. Use your AWS account ID or account alias, your IAM user name, and your password to sign in to the [IAM console](#).

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 **Security credentials**.



3. On the **AWS IAM credentials** tab, in the **Multi-factor authentication (MFA)** section, choose the radio button next to the MFA device and choose **Resync**.
4. Type the next two sequentially generated codes from the device into **MFA code 1** and **MFA code 2**. Then choose **Resync**.

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. In the **Multi-factor authentication (MFA)** section, choose the radio button next to the MFA device and choose **Resync**.
4. Type the next two sequentially generated codes from the device into **MFA code 1** and **MFA code 2**. Then choose **Resync**.

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.

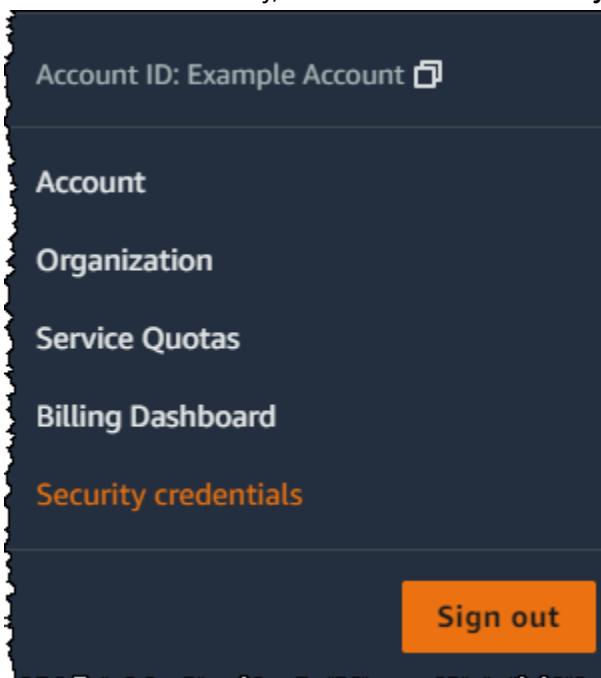
To resynchronize your root user MFA device after signing in (console)

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

As the root user, you can't sign in to the **Sign in as IAM user** page. If you see the **Sign in as IAM user** page, choose **Sign in using root user email** near the bottom of the page. For help signing in as the root user, see [Signing in to the AWS Management Console as the root user](#) in the *AWS Sign-In User Guide*.

2. On the right side of the navigation bar, choose on your account name, and then choose **Security credentials**. If necessary, choose **Continue to Security credentials**.



3. Expand the **Multi-factor authentication (MFA)** section on the page.
4. Choose the radio button next to the device and choose **Resync**.
5. In the **Resync MFA device** dialog box, type the next two sequentially generated codes from the device into **MFA code 1** and **MFA code 2**. Then choose **Resync**.

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 are having 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" \(p. 115\)](#).

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 \(console\) \(p. 142\)](#)
- [Deactivating MFA devices \(AWS CLI\) \(p. 143\)](#)
- [Deactivating MFA devices \(AWS API\) \(p. 143\)](#)

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.
5. Under **Multi-factor authentication (MFA)**, choose the radio button next to the MFA device, choose **Remove**, and then choose **Remove**.

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.

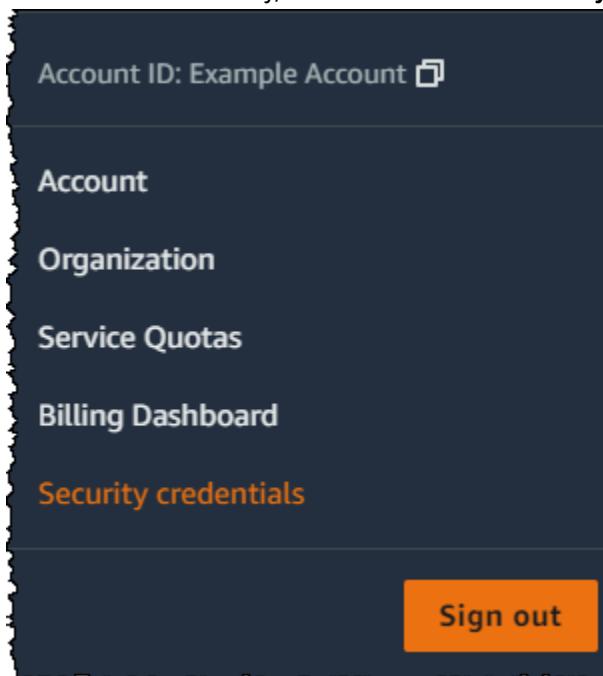
To deactivate the MFA device for your AWS account root user (console)

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

As the root user, you can't sign in to the **Sign in as IAM user** page. If you see the **Sign in as IAM user** page, choose **Sign in using root user email** near the bottom of the page. For help signing in as the root user, see [Signing in to the AWS Management Console as the root user](#) in the [AWS Sign-In User Guide](#).

2. On the right side of the navigation bar, choose on your account name, and then choose **Security credentials**. If necessary, choose **Continue to Security credentials**.



3. In the **Multi-factor authentication (MFA)** section, choose the radio button next the MFA device that you want to deactivate and choose **Remove**.
4. Choose **Remove**.

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 `@signin.aws` or `@verify.signin.aws`.

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 \(p. 116\)](#) or [hardware TOTP token \(p. 129\)](#) appears to be functioning properly, but you can't 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 \(p. 137\)](#). [FIDO security keys \(p. 120\)](#) do not go out of sync.

If your AWS account root user [multi-factor authentication \(MFA\) device \(p. 114\)](#) is lost, damaged, or not working, you can recover access to your account. IAM users must contact an administrator to deactivate the device.

Important

We recommend that you enable multiple MFA devices for your IAM users to ensure continued access to your account in case of lost or inaccessible MFA device. You can register up to eight MFA devices of any combination of the currently supported MFA types with your AWS account root user and IAM users.

Recovering a root user MFA device

If your AWS account root user [multi-factor authentication \(MFA\) device \(p. 114\)](#) is lost, damaged, or not working, you can sign in using another MFA device registered to the same AWS account root user. If the root user only has one MFA device enabled, you can use 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 the primary contact phone number registered with your account.

Before you use alternative factors of authentication to sign in as a root user, you must be able to access the email and primary contact phone number that are associated with your account. If you need to update the primary contact phone number, you can sign in as an IAM user with *Administrator* access instead of the root user. For additional instructions on updating the account contact information, see [Editing contact information](#) in the *AWS Billing User Guide*. If you do not have access to an email and primary contact phone number, you must contact [AWS Support](#).

Important

We recommend that you keep the email address and contact phone number linked to your root user up to date for a successful account recovery.

To sign in using alternative factors of authentication as an AWS account root user

- 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 With Authentication Device** page, choose **Having problems with your authentication device? Click here.**

Note

You might see alternative text, such as **Sign in using MFA**, **Troubleshoot your authentication device**, or **Troubleshoot MFA**, but the functionality is the same. If you can't use alternative factors of authentication to verify your account email address and primary contact phone number, 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 (recover-mfa-no-reply@verify.signin.aws). 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 primary contact 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 see [Lost or unusable Multi-Factor Authentication \(MFA\) device](#) 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 FIDO security key, go to the [AWS Security Credentials](#) page and deactivate the old FIDO security key before enabling a new one.
 - For a hardware TOTP token, 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 FIDO security key and order a new one, you can use virtual MFA or a hardware TOTP token until you receive a new FIDO security key.

Important

If your MFA device is missing or stolen, after signing in using alternative factors of authentication and establishing your replacement MFA device, change your root user password in case an attacker has stolen the authentication device and might also have your current password. For more information, see [Change the password for the AWS account root user](#) in the *AWS Account Management Reference Guide*.

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 \(p. 141\)](#) 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 \(console\) \(p. 116\)](#).
 - For a FIDO security key, contact the third-party provider for help replacing the device. When you receive the new FIDO security key, enable it as described in [Enabling a FIDO security key \(console\) \(p. 120\)](#).
 - For a hardware TOTP token, 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 TOTP token \(console\) \(p. 129\)](#).

Note

You don't have to replace a lost or stolen MFA device with the same type of device. You can have up to eight MFA devices of any combination. For example, if you break your FIDO security key and order a new one, you can use virtual MFA or a hardware TOTP token until you receive a new FIDO security key.

3. If your MFA device is missing or stolen, also change your password in case an attacker has stolen the authentication device and might also have your current password. For more information, see [Managing passwords for IAM users \(p. 97\)](#)

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 \(p. 145\)](#)
- [Scenario: MFA protection for cross-account delegation \(p. 148\)](#)
- [Scenario: MFA protection for access to API operations in the current account \(p. 149\)](#)
- [Scenario: MFA protection for resources that have resource-based policies \(p. 150\)](#)

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 \(p. 115\)](#).
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 \(p. 1338\)](#), [Numeric condition operators \(p. 1284\)](#), and [Condition operator to check existence of condition keys \(p. 1290\)](#).

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 \(p. 184\)](#).)

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"},
      "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": "ListTables",
      "Effect": "Allow",
      "Action": "dynamodb>ListTables",
      "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 \(p. 152\)](#).

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": "*"
    }
  ]
}
```

```

    "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 \(p. 151\)](#) 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 \(p. 151\)](#) 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

The following example shows 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.

The following code example shows how to get a session token with AWS STS and use it to perform a service action that requires an MFA token.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Get a session token by passing an MFA token and use it to list Amazon S3 buckets for the account.

```

def list_buckets_with_session_token_with_mfa(mfa_serial_number, mfa_totp,
                                             sts_client):
    """
    Gets a session token with MFA credentials and uses the temporary session
    credentials to list Amazon S3 buckets.

    Requires an MFA device serial number and token.

    :param mfa_serial_number: The serial number of the MFA device. For a virtual
        MFA
        device, this is an Amazon Resource Name (ARN).
    :param mfa_totp: A time-based, one-time password issued by the MFA device.
    :param sts_client: A Boto3 STS instance that has permission to assume the role.
    """
    if mfa_serial_number is not None:
        response = sts_client.get_session_token(
            SerialNumber=mfa_serial_number, TokenCode=mfa_totp)
    else:
        response = sts_client.get_session_token()
    temp_credentials = response['Credentials']

    s3_resource = boto3.resource(
        's3',
        aws_access_key_id=temp_credentials['AccessKeyId'],
        aws_secret_access_key=temp_credentials['SecretAccessKey'],
        aws_session_token=temp_credentials['SessionToken'])

    print(f"Buckets for the account:")
    for bucket in s3_resource.buckets.all():
        print(bucket.name)

```

- For API details, see [GetSessionToken](#) in *AWS SDK for Python (Boto3) API Reference*.

Calling AssumeRole with MFA authentication

The following examples show 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 \(p. 148\)](#).

The following code examples show how to assume a role with AWS STS.

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```

using Amazon;
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;
using System;
using System.Threading.Tasks;

namespace AssumeRoleExample

```

```
{
    class AssumeRole
    {
        ///<summary>
        /// This example shows how to use the AWS Security Token
        /// Service (AWS STS) to assume an IAM role.
        ///
        /// NOTE: It is important that the role that will be assumed has a
        /// trust relationship with the account that will assume the role.
        ///
        /// Before you run the example, you need to create the role you want to
        /// assume and have it trust the IAM account that will assume that role.
        ///
        /// See https://docs.aws.amazon.com/IAM/latest/UserGuide/id\_roles\_create.html
        /// for help in working with roles.
        /// </summary>

        private static readonly RegionEndpoint REGION = RegionEndpoint.USWest2;

        static async Task Main()
        {
            // Create the SecurityToken client and then display the identity of the
            // default user.
            var roleArnToAssume = "arn:aws:iam::123456789012:role/testAssumeRole";

            var client = new
                Amazon.SecurityToken.AmazonSecurityTokenServiceClient(REGION);

            // Get and display the information about the identity of the default
            user.
            var callerIdRequest = new GetCallerIdentityRequest();
            var caller = await client.GetCallerIdentityAsync(callerIdRequest);
            Console.WriteLine($"Original Caller: {caller.Arн}");

            // Create the request to use with the AssumeRoleAsync call.
            var assumeRoleReq = new AssumeRoleRequest()
            {
                DurationSeconds = 1600,
                RoleSessionName = "Session1",
                RoleArn = roleArnToAssume
            };

            var assumeRoleRes = await client.AssumeRoleAsync(assumeRoleReq);

            // Now create a new client based on the credentials of the caller
            assuming the role.
            var client2 = new AmazonSecurityTokenServiceClient(credentials:
            assumeRoleRes.Credentials);

            // Get and display information about the caller that has assumed the
            defined role.
            var caller2 = await client2.GetCallerIdentityAsync(callerIdRequest);
            Console.WriteLine($"AssumedRole Caller: {caller2.Arн}");
        }
    }
}
```

- For API details, see [AssumeRole in AWS SDK for .NET API Reference](#).

Bash

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
#####
function iecho() {
    if [[ $VERBOSE == true ]]; then
        echo "$@"
    fi
}

#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
    printf "%s\n" "$*" 2>&1
}

#####
# function sts_assume_role
#
# This function assumes a role in the AWS account and returns the temporary
# credentials.
#
# Parameters:
#   -n role_session_name -- The name of the session.
#   -r role_arn -- The ARN of the role to assume.
#
# Returns:
#   [access_key_id, secret_access_key, session_token]
#   And:
#     0 - If successful.
#     1 - If an error occurred.
#####
function sts_assume_role() {
    local role_session_name role_arn response
    local option OPTARG # Required to use getopts command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function sts_assume_role"
        echo "Assumes a role in the AWS account and returns the temporary credentials:"
        echo "  -n role_session_name -- The name of the session."
        echo "  -r role_arn -- The ARN of the role to assume."
        echo ""
    }

    while getopts n:r:h option; do
        case "${option}" in
            n) role_session_name=${OPTARG} ;;
            r) role_arn=${OPTARG} ;;
            h)
                usage
                ;;
        esac
    done
}
```

```

        return 0
    ;;
\?)
    ech o"Invalid parameter"
    usage
    return 1
;;
esac
done

response=$(aws sts assume-role \
--role-session-name "$role_session_name" \
--role-arn "$role_arn" \
--output text \
--query "Credentials.[AccessKeyId, SecretAccessKey, SessionToken]")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports create-role operation failed.\n$response"
    return 1
fi

echo "$response"

return 0
}

```

- For API details, see [AssumeRole](#) in *AWS CLI Command Reference*.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```

bool AwsDoc::STS::assumeRole(const Aws::String &roleArn,
                             const Aws::String &roleSessionName,
                             const Aws::String &externalId,
                             Aws::Auth::AWSCredentials &credentials,
                             const Aws::Client::ClientConfiguration &clientConfig)
{
    Aws::STS::STSCient sts(clientConfig);
    Aws::STS::Model::AssumeRoleRequest sts_req;

    sts_req.SetRoleArn(roleArn);
    sts_req.SetRoleSessionName(roleSessionName);
    sts_req.SetExternalId(externalId);

    const Aws::STS::Model::AssumeRoleOutcome outcome = sts.AssumeRole(sts_req);

    if (!outcome.IsSuccess()) {
        std::cerr << "Error assuming IAM role. " <<
                    outcome.GetError().GetMessage() << std::endl;
    }
    else {
        std::cout << "Credentials successfully retrieved." << std::endl;
        const Aws::STS::Model::AssumeRoleResult result = outcome.GetResult();
    }
}

```

```
    const Aws::STS::Model::Credentials &temp_credentials =
result.GetCredentials();

    // Store temporary credentials in return argument.
    // Note: The credentials object returned by assumeRole differs
    // from the AWS Credentials object used in most situations.
    credentials.SetAWSAccessKeyId(temp_credentials.GetAccessKeyId());
    credentials.SetAWSSecretKey(temp_credentials.GetSecretAccessKey());
    credentials.SetSessionToken(temp_credentials.GetSessionToken());
}

return outcome.IsSuccess();
}
```

- For API details, see [AssumeRole](#) in *AWS SDK for C++ API Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public static void assumeGivenRole(StsClient stsClient, String roleArn, String
roleSessionName) {

    try {
        AssumeRoleRequest roleRequest = AssumeRoleRequest.builder()
            .roleArn(roleArn)
            .roleSessionName(roleSessionName)
            .build();

        AssumeRoleResponse roleResponse = stsClient.assumeRole(roleRequest);
        Credentials myCreds = roleResponse.credentials();

        // Display the time when the temp creds expire.
        Instant exTime = myCreds.expiration();
        String tokenInfo = myCreds.sessionToken();

        // Convert the Instant to readable date.
        DateTimeFormatter formatter =
            DateTimeFormatter.ofLocalizedDateTime(FormatStyle.SHORT)
                .withLocale(Locale.US)
                .withZone(ZoneId.systemDefault());

        formatter.format(exTime);
        System.out.println("The token " + tokenInfo + " expires on " + exTime);

    } catch (StsException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}
```

- For API details, see [AssumeRole](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Create the client.

```
import { STSClient } from "@aws-sdk/client-sts";
// Set the AWS Region.
const REGION = "us-east-1";
// Create an AWS STS service client object.
export const client = new STSClient({ region: REGION });
```

Assume the IAM role.

```
import { AssumeRoleCommand } from "@aws-sdk/client-sts";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Returns a set of temporary security credentials that you can use to
    // access Amazon Web Services resources that you might not normally
    // have access to.
    const command = new AssumeRoleCommand({
      // The Amazon Resource Name (ARN) of the role to assume.
      RoleArn: "ROLE_ARN",
      // An identifier for the assumed role session.
      RoleSessionName: "session1",
      // The duration, in seconds, of the role session. The value specified
      // can range from 900 seconds (15 minutes) up to the maximum session
      // duration set for the role.
      DurationSeconds: 900,
    });
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- For API details, see [AssumeRole in AWS SDK for JavaScript API Reference](#).

SDK for JavaScript (v2)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// Load the AWS SDK for Node.js
const AWS = require('aws-sdk');
// Set the region
AWS.config.update({region: 'REGION'});

var roleToAssume = {RoleArn: 'arn:aws:iam::123456789012:role/RoleName',
  RoleSessionName: 'session1',
```

```

        DurationSeconds: 900,};

var roleCreds;

// Create the STS service object
var sts = new AWS.STS({apiVersion: '2011-06-15'});

//Assume Role
sts.assumeRole(roleToAssume, function(err, data) {
    if (err) console.log(err, err.stack);
    else{
        roleCreds = {accessKeyId: data.Credentials.AccessKeyId,
                     secretAccessKey: data.Credentials.SecretAccessKey,
                     sessionToken: data.Credentials.SessionToken};
        stsGetCallerIdentity(roleCreds);
    }
});

//Get Arn of current identity
function stsGetCallerIdentity(creds) {
    var stsParams = {credentials: creds };
    // Create STS service object
    var sts = new AWS.STS(stsParams);

    sts.getCallerIdentity({}, function(err, data) {
        if (err) {
            console.log(err, err.stack);
        }
        else {
            console.log(data.Arn);
        }
    });
}

```

- For API details, see [AssumeRole](#) in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Assume an IAM role that requires an MFA token and use temporary credentials to list Amazon S3 buckets for the account.

```

def list_buckets_from_assumed_role_with_mfa(
    assume_role_arn, session_name, mfa_serial_number, mfa_totp, sts_client):
    """
    Assumes a role from another account and uses the temporary credentials from
    that role to list the Amazon S3 buckets that are owned by the other account.
    Requires an MFA device serial number and token.

    The assumed role must grant permission to list the buckets in the other
    account.

    :param assume_role_arn: The Amazon Resource Name (ARN) of the role that
                           grants access to list the other account's buckets.
    :param session_name: The name of the STS session.
    :param mfa_serial_number: The serial number of the MFA device. For a virtual
                             MFA
                                         device, this is an ARN.

```

```
:param mfa_totp: A time-based, one-time password issued by the MFA device.  
:param sts_client: A Boto3 STS instance that has permission to assume the role.  
"""  
response = sts_client.assume_role(  
    RoleArn=assume_role_arn,  
    RoleSessionName=session_name,  
    SerialNumber=mfa_serial_number,  
    TokenCode=mfa_totp)  
temp_credentials = response['Credentials']  
print(f"Assumed role {assume_role_arn} and got temporary credentials.")  
  
s3_resource = boto3.resource(  
    's3',  
    aws_access_key_id=temp_credentials['AccessKeyId'],  
    aws_secret_access_key=temp_credentials['SecretAccessKey'],  
    aws_session_token=temp_credentials['SessionToken'])  
  
print(f"Listing buckets for the assumed role's account:")  
for bucket in s3_resource.buckets.all():  
    print(bucket.name)
```

- For API details, see [AssumeRole in AWS SDK for Python \(Boto3\) API Reference](#).

Ruby

SDK for Ruby

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
# Creates an AWS Security Token Service (AWS STS) client with specified  
credentials.  
# This is separated into a factory function so that it can be mocked for unit  
testing.  
#  
# @param key_id [String] The ID of the access key used by the STS client.  
# @param key_secret [String] The secret part of the access key used by the STS  
client.  
def create_sts_client(key_id, key_secret)  
  Aws::STS::Client.new(access_key_id: key_id, secret_access_key: key_secret)  
end  
  
# Gets temporary credentials that can be used to assume a role.  
#  
# @param role_arn [String] The ARN of the role that is assumed when these  
credentials  
#           are used.  
# @param sts_client [AWS::STS::Client] An AWS STS client.  
# @return [Aws::AssumeRoleCredentials] The credentials that can be used to assume  
the role.  
def assume_role(role_arn, sts_client)  
  credentials = Aws::AssumeRoleCredentials.new(  
    client: sts_client,  
    role_arn: role_arn,  
    role_session_name: "create-use-assume-role-scenario"  
  )  
  puts("Assumed role '#{role_arn}', got temporary credentials.")  
  credentials  
end
```

- For API details, see [AssumeRole](#) in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
async fn assume_role(
    config: &SdkConfig,
    region: Region,
    role_name: String,
    session_name: Option<String>,
) -> Result<(), Error> {
    match config.credentials_provider() {
        Some(credential) => {
            let provider = aws_config::sts::AssumeRoleProvider::builder(role_name)
                .region(region)
                .session_name(session_name.unwrap_or_else(|| String::from("rust-
assume-role")))
                .build(credential.clone());
            let local_config = aws_config::from_env()
                .credentials_provider(provider)
                .load()
                .await;
            let client = Client::new(&local_config);
            let req = client.get_caller_identity();
            let resp = req.send().await;
            match resp {
                Ok(e) => {
                    println!("UserID : {}", e.user_id().unwrap_or_default());
                    println!("Account: {}", e.account().unwrap_or_default());
                    println!("Arn : {}", e.arn().unwrap_or_default());
                }
                Err(e) => println!("{}:{}", e),
            }
        }
        None => {
            println!("No config provided");
        }
    }
    Ok(())
}
```

- For API details, see [AssumeRole](#) in *AWS SDK for Rust API reference*.

Swift

SDK for Swift

Note

This is prerelease documentation for an SDK in preview release. It is subject to change.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public func assumeRole(role: IAMClientTypes.Role, sessionName: String)
    async throws -> STSClientTypes.Credentials {
    let input = AssumeRoleInput(
        roleArn: role.arn,
        roleSessionName: sessionName
    )
    do {
        let output = try await stsClient.assumeRole(input: input)

        guard let credentials = output.credentials else {
            throw ServiceHandlerError.authError
        }

        return credentials
    } catch {
        throw error
    }
}
```

- For API details, see [AssumeRole](#) in *AWS SDK for Swift API reference*.

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 **Select visible columns**, select **Console last sign-in**.
 - c. Choose **Confirm** 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 \(AWS CLI\) \(p. 167\)](#).

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 **Select visible columns**, select **Access key last used**.
 - c. Choose **Confirm** 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 – 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 \(AWS CLI\) \(p. 167\)](#).

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 \(p. 164\)](#)
- [Understanding the report format \(p. 164\)](#)
- [Getting credential reports \(console\) \(p. 167\)](#)
- [Getting credential reports \(AWS CLI\) \(p. 167\)](#)
- [Getting credential reports \(AWS API\) \(p. 168\)](#)

Required permissions

The following permissions are needed to create and download reports:

- To create a credential report: `iam:GenerateCredentialReport`
- To download the report: `iam: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 \(p. 1213\)](#).

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 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 \(p. 114\)](#) (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 \(p. 108\)](#).

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 \(p. 103\)](#), which you can use with the credential helper included with the AWS CLI to communicate with CodeCommit repositories over HTTPS.

Note

You cannot use SSH keys or Git credentials to access repositories in another AWS account. To learn how to configure access to CodeCommit repositories for IAM users and groups in another AWS account, see [Configure cross-account access to an AWS CodeCommit repository using roles](#) in the [AWS CodeCommit User Guide](#).

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. See the following topics for more information:

- To learn more about federated access, see [Identity providers and federation \(p. 198\)](#) and [Providing access to externally authenticated users \(identity federation\) \(p. 196\)](#).
- To learn more about temporary credentials, see [Temporary security credentials in IAM \(p. 426\)](#) 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 through the AWS Management Console, or programmatically. To access Amazon Keyspaces programmatically with service-specific credentials, you can use cqlsh or open-source Cassandra drivers. *Service-specific credentials* include a user name and password like those that Cassandra uses for authentication and access management. To access Amazon Keyspaces programmatically with AWS access keys, you can use the AWS SDK, the AWS Command Line Interface (AWS CLI) or open-source Cassandra drivers with the SigV4 plugin. To learn more, see [Connecting programmatically to Amazon Keyspaces](#) in the [Amazon Keyspaces \(for Apache Cassandra\) Developer Guide](#).

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 using the console](#) in the [Amazon Keyspaces \(for Apache Cassandra\) Developer Guide](#).

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 \(AWS API\) \(p. 171\)](#)
- [Retrieving a server certificate \(AWS API\) \(p. 172\)](#)
- [Listing server certificates \(AWS API\) \(p. 173\)](#)
- [Tagging and Untagging Server Certificates \(AWS API\) \(p. 173\)](#)
- [Renaming a server certificate or updating its path \(AWS API\) \(p. 173\)](#)
- [Deleting a server certificate \(AWS API\) \(p. 173\)](#)
- [Troubleshooting \(p. 174\)](#)

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 \(p. 174\)](#).
- The certificate, private key, and certificate chain must all be PEM-encoded. For help converting these items to PEM format, see [Troubleshooting \(p. 174\)](#).

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.
- (Optional) You want to tag the server certificate with a key-value pair. For example, you might add the tag key Department and the tag value Engineering to help you identify and organize your certificates.

To use the following example command, replace these file names with your own. Replace *ExampleCertificate* with a name for your uploaded certificate. If you want to tag the certificate, replace the *ExampleKey* and *ExampleValue* tag key-value pair with your own values. 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
--tags '{"Key": "ExampleKey", "Value": "ExampleValue"}'
```

When the preceding command is successful, it returns metadata about the uploaded certificate, including its [Amazon Resource Name \(ARN\) \(p. 1213\)](#), its friendly name, its identifier (ID), its expiration date, tags, 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](#).

Tagging and Untagging Server Certificates (AWS API)

You can attach tags to your IAM resources to organize and control access to them. To use the IAM API to tag an existing server certificate, send a [TagServerCertificate](#) request. The following example shows how to do this with the AWS CLI.

```
aws iam tag-server-certificate --server-certificate-name ExampleCertificate
                               --tags '{"Key": "ExampleKey", "Value": "ExampleValue"}'
```

When the preceding command is successful, no output is returned.

To use the IAM API to untag a server certificate, send a [UntagServerCertificate](#) request. The following example shows how to do this with the AWS CLI.

```
aws iam untag-server-certificate --server-certificate-name ExampleCertificate
                                   --tag-keys ExampleKeyName
```

When the preceding command is successful, no output is returned.

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-----  
Base64-encoded certificate  
-----END CERTIFICATE-----  
-----BEGIN CERTIFICATE-----  
Base64-encoded 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 user groups

An IAM [user group \(p. 175\)](#) is a collection of IAM users. User groups let you specify permissions for multiple users, which can make it easier to manage the permissions for those users. For example, you

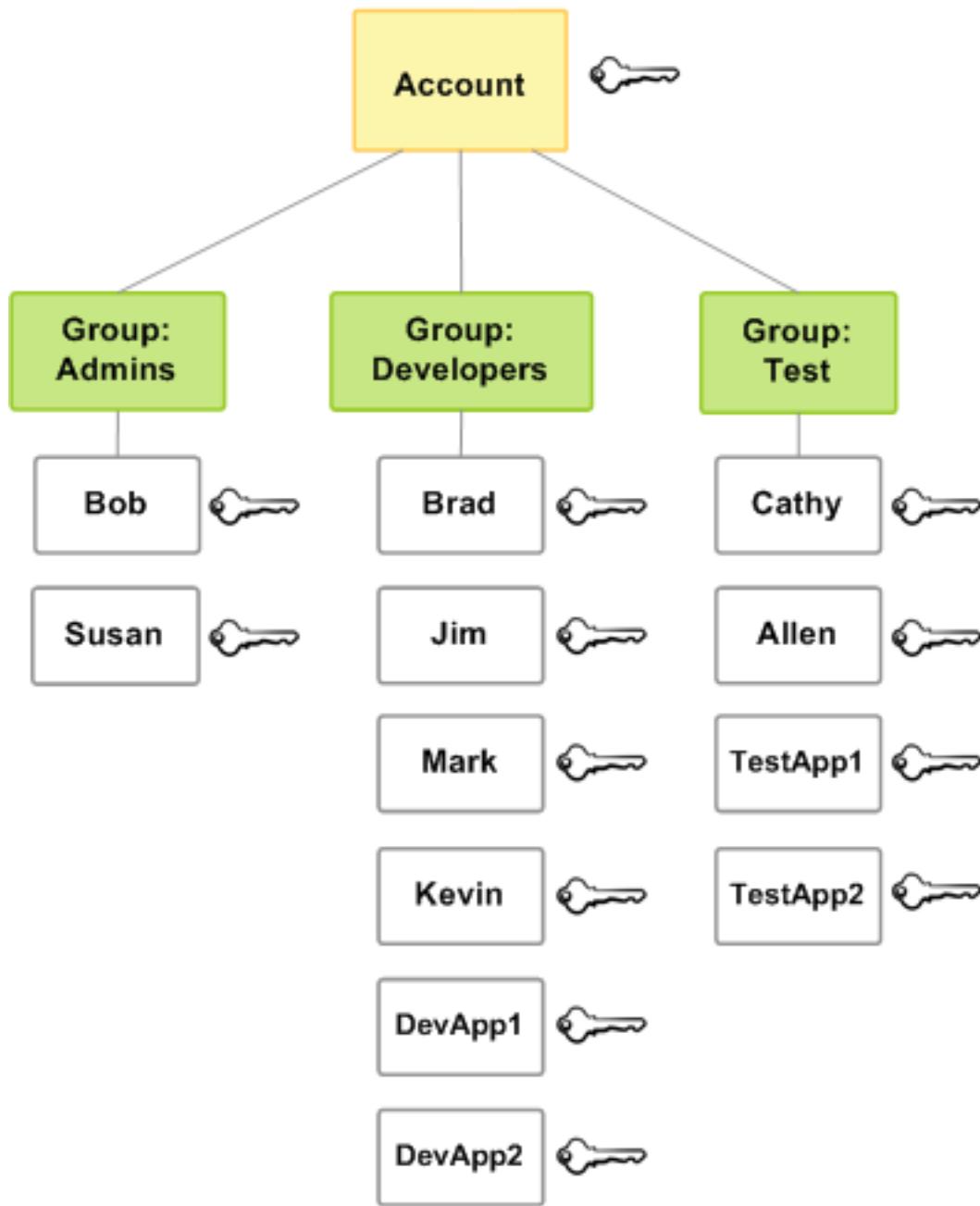
could have a user group called *Admins* and give that user group typical administrator permissions. Any user in that user group automatically has *Admins* group permissions. If a new user joins your organization and needs administrator privileges you can assign the appropriate permissions by adding the user to the *Admins* user group. If a person changes jobs in your organization, instead of editing that user's permissions you can remove them from the old user groups and add them to the appropriate new user groups.

You can attach an identity-based policy to a user group so that all of the users in the user group receive the policy's permissions. You cannot identify a user group as a Principal in a policy (such as a resource-based policy) because groups relate to permissions, not authentication, and principals are authenticated IAM entities. For more information about policy types, see [Identity-based policies and resource-based policies \(p. 511\)](#).

Here are some important characteristics of user groups:

- A user group can contain many users, and a user can belong to multiple user groups.
- User groups can't be nested; they can contain only users, not other user groups.
- There is no default user group that automatically includes all users in the AWS account. If you want to have a user group like that, you must create it and assign each new user to it.
- The number and size of IAM resources in an AWS account, such as the number of groups, and the number of groups that a user can be a member of, are limited. For more information, see [IAM and AWS STS quotas \(p. 1220\)](#).

The following diagram shows a simple example of a small company. The company owner creates an *Admins* user group for users to create and manage other users as the company grows. The *Admins* user group creates a *Developers* user group and a *Test* user group. Each of these user 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 user group. However, users can belong to multiple user groups.



Creating IAM user groups

Note

As a [best practice \(p. 1032\)](#), we recommend that you require human users to use federation with an identity provider to access AWS using temporary credentials. If you follow the best practices, you are not managing IAM users and groups. Instead, your users and groups are managed outside of AWS and are able to access AWS resources as a *federated identity*. A federated identity is a user from your enterprise user directory, a web identity provider, the AWS Directory Service, the Identity Center directory, or any user that accesses AWS services by using

credentials provided through an identity source. Federated identities use the groups defined by their identity provider. If you are using AWS IAM Identity Center (successor to AWS Single Sign-On), see [Manage identities in IAM Identity Center](#) in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide* for information about creating users and groups in IAM Identity Center.

To set up a user 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 user group, see [Permissions required to access IAM resources \(p. 662\)](#).

To create an IAM user 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, choose **User groups** and then choose **Create group**.
3. For **User group name**, type the name of the group.

Note

The number and size of IAM resources in an AWS account are limited. For more information, see [IAM and AWS STS quotas \(p. 1220\)](#). Group names can be a combination of up to 128 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 users, select the check box for each user that you want to add to the group.
5. In the list of policies, select the check box for each policy that you want to apply to all members of the group.
6. Choose **Create group**.

To create IAM user groups (AWS CLI or AWS API)

Use one of the following:

- AWS CLI: [aws iam create-group](#)
- AWS API: [CreateGroup](#)

Managing IAM user groups

Amazon Web Services offers multiple tools for managing IAM user groups. For information about the permissions that you need in order to add and remove users in a user group, see [Permissions required to access IAM resources \(p. 662\)](#).

Topics

- [Listing IAM user groups \(p. 179\)](#)
- [Adding and removing users in an IAM user group \(p. 179\)](#)
- [Attaching a policy to an IAM user group \(p. 180\)](#)
- [Renaming an IAM user group \(p. 181\)](#)
- [Deleting an IAM user group \(p. 182\)](#)

Listing IAM user groups

You can list all the user groups in your account, list the users in a user group, and list the user groups a user belongs to. If you use the AWS CLI or AWS API, you can list all the user groups with a particular path prefix.

To list all the user groups in your account

Do any of the following:

- [AWS Management Console](#): In the navigation pane, choose **User groups**.
- AWS CLI: [aws iam list-groups](#)
- AWS API: [ListGroups](#)

To list the users in a specific user group

Do any of the following:

- [AWS Management Console](#): In the navigation pane, choose **User 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 user 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 user group

Use user groups to apply the same permissions policies across multiple users at once. You can then add users to or remove users from an IAM user 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 \(p. 616\)](#).

Add or remove a user in a user group (console)

You can use the AWS Management Console to add or remove a user from a user group.

To add a user to an IAM user 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 **User groups** and then choose the name of the group.

3. Choose the **Users** tab and then choose **Add users**. 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 **User groups** and then choose the name of the group.
3. Choose the **Users** tab. Select the check box next to the users you want to remove and then choose **Remove users**.

Add or remove a user in a user group (AWS CLI)

You can use the AWS CLI to add or remove a user from a user group.

To add a user to an IAM user group (AWS CLI)

- Use the following command:
 - [aws iam add-user-to-group](#)

To remove a user from an IAM user group (AWS CLI)

- Use the following command:
 - [aws iam remove-user-from-group](#)

Add or remove a user in a user group (AWS API)

You can use the AWS API to add or remove a user in a user group.

To add a user to an IAM group (AWS API)

- Complete the following operation:
 - [AddUserToGroup](#)

To remove a user from an IAM user group (AWS API)

- Complete the following operation:
 - [RemoveUserFromGroup](#)

Attaching a policy to an IAM user group

You can attach an [AWS managed policy \(p. 495\)](#)—that is, a prewritten policy provided by AWS—to a user 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 \(p. 581\)](#).

For more information about permissions and policies, see [Access management for AWS resources \(p. 484\)](#).

To attach a policy to a user 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 **User groups** and then choose the name of the group.
3. Choose the **Permissions** tab.
4. Choose **Add permissions** and then choose **Attach policy**.
5. The current policies attached to the user group are displayed in the **Current permissions policies** list. In the list of **Other permissions policies**, select the check box next to the names of the policies to attach. You can use the search box to filter the list of policies by type and policy name.
6. Choose **Attach policies**.

To attach a policy to a user group (AWS CLI or AWS API)

Do either of the following:

- AWS CLI: [aws iam attach-group-policy](#)
- AWS API: [AttachGroupPolicy](#)

Renaming an IAM user group

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

- Any policies attached to the user group stay with the group under the new name.
- The user group retains all its users under the new name.
- The unique ID for the user group remains the same. For more information about unique IDs, see [Unique identifiers \(p. 1218\)](#).

IAM does not automatically update policies that refer to the user group as a resource to use the new name. Therefore, you must be careful when you rename a user group. Before you rename your user group, you must manually check all of your policies to find any policies where that user 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 user group. If an administrator changes the name of the user 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 user group.

To find policies that refer to a user group as a resource:

1. From the navigation pane of the IAM console, choose **Policies**.
2. Sort by the **Type** column to find your **Customer managed** custom policies.
3. Choose the policy name of the policy to edit.
4. Choose the **Permissions** tab, and then choose **Summary**.
5. Choose **IAM** from the list of services, if it exists.
6. Look for the name of your user group in the **Resource** column.
7. Choose **Edit** to change the name of your user group in the policy.

To change the name of an IAM user group

Do any of the following:

- [AWS Management Console](#): In the navigation pane, choose **User groups** and then select the group name. Choose **Edit**. Type the new user group name and then choose **Save changes**.
- AWS CLI: [aws iam update-group](#)
- AWS API: [UpdateGroup](#)

Deleting an IAM user group

When you delete a user 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 user group as a resource, you must be careful when you delete a user group. Before you delete your user group, you must manually check all of your policies to find any policies that mention the group by name. For example, John, the Test Team manager, has a policy attached to his IAM user entity that lets him add and remove users from the Test user group. If an administrator deletes the group, the administrator must also delete the policy attached to John. Otherwise, if the administrator recreates the deleted group and give it the same name, John's permissions remain in place, even if he left the Test Team.

To find policies that refer to a user group as a resource

1. From the navigation pane of the IAM console, choose **Policies**.
2. Sort by the **Type** column to find your **Customer managed** custom policies.
3. Choose the policy name of the policy to delete.
4. Choose the **Permissions** tab, and then choose **Summary**.
5. Choose **IAM** from the list of services, if it exists.
6. Look for the name of your user group in the **Resource** column.
7. Choose **Delete** to delete the policy.
8. Type the policy name to confirm deletion of the policy and choose **Delete**.

In contrast, when you use the AWS CLI, Tools for Windows PowerShell, or AWS API to delete a user group, you must first remove the users in the group. Then delete any inline policies embedded in the user group. Next, detach any managed policies that are attached to the group. Only then can you delete the user group itself.

Deleting an IAM user group (console)

You can delete an IAM user group from the AWS Management Console.

To delete an IAM user 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 **User groups**.
3. In the list of user groups, select the check box next to the names of the user groups to delete. You can use the search box to filter the list of user groups by type, permissions, and user group name.
4. Choose **Delete**.
5. In the confirmation box, if you want to delete a single user group, type the user group name and choose **Delete**. If you want to delete multiple user groups, type the number of user groups to delete followed by **user groups** and choose **Delete**. For example, if you delete three user groups, type **3 user groups**.

Deleting an IAM user group (AWS CLI)

You can delete an IAM user group from the AWS CLI.

To delete an IAM user group (AWS CLI)

1. Remove all users from the user group.
 - [aws iam get-group](#) (to get the list of users in the user group), and [aws iam remove-user-from-group](#) (to remove a user from the user group)
2. Delete all inline policies embedded in the user group.
 - [aws iam list-group-policies](#) (to get a list of the user group's inline policies), and [aws iam delete-group-policy](#) (to delete the user group's inline policies)
3. Detach all managed policies attached to the user group.
 - [aws iam list-attached-group-policies](#) (to get a list of the managed policies attached to the user group), and [aws iam detach-group-policy](#) (to detach a managed policy from the user group)
4. Delete the user group.
 - [aws iam delete-group](#)

Deleting an IAM user group (AWS API)

You can use the AWS API to delete an IAM user group.

To delete an IAM user group (AWS API)

1. Remove all users from the user group.
 - [GetGroup](#) (to get the list of users in the user group) and [RemoveUserFromGroup](#) (to remove a user from the user group)
2. Delete all inline policies embedded in the user group.
 - [ListGroupPolicies](#) (to get a list of the user group's inline policies) and [DeleteGroupPolicy](#) (to delete the user group's inline policies)
3. Detach all managed policies attached to the user group.
 - [ListAttachedGroupPolicies](#) (to get a list of the managed policies attached to the user group) and [DetachGroupPolicy](#) (to detach a managed policy from the user group)
4. Delete the user 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.

Note

When you first create your AWS account, no roles are created by default. As you add services to your account, they may add service-linked roles to support their use cases.

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

Before you can delete service-linked roles you must first delete their related resources. This protects your resources because you can't inadvertently remove permission to access the resources.

For information about which services support using service-linked roles, see [AWS services that work with IAM \(p. 1224\)](#) 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.

Topics

- [Roles terms and concepts \(p. 184\)](#)
- [Common scenarios for roles: Users, applications, and services \(p. 187\)](#)
- [Identity providers and federation \(p. 198\)](#)
- [Using service-linked roles \(p. 242\)](#)
- [Creating IAM roles \(p. 250\)](#)
- [Using IAM roles \(p. 274\)](#)
- [Managing IAM roles \(p. 384\)](#)

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 service role is an [IAM role](#) that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see [Creating a role to delegate permissions to an AWS service](#) in the *IAM User Guide*.

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 \(p. 374\)](#).

AWS service-linked role

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

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 \(p. 1197\)](#).

For information about which services support using service-linked roles, see [AWS services that work with IAM \(p. 1224\)](#) 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. For more information, see [Using service-linked roles \(p. 242\)](#).

Role chaining

Role chaining is when you use a role to assume a second role through the AWS CLI or API. For example, RoleA has permission to assume RoleB. You can enable User1 to assume RoleA by using their long-term user credentials in the AssumeRole API operation. This returns RoleA short-term credentials. With role chaining, you can use RoleA's short-term credentials to enable User1 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 \(p. 417\)](#).

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 \(p. 276\)](#) 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 \(p. 374\)](#) 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 \(p. 251\)](#) in the trusting account that has two [policies \(p. 186\)](#) 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 part of and ARN in the principal element. 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 \(p. 277\)](#). 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 \(p. 191\)](#) 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 \(p. 198\)](#). For more information about federated users, see [Federated Users and Roles \(p. 14\)](#) in the *IAM User Guide*.

Trust policy

A [JSON policy document \(p. 1323\)](#) in which you define the principals that you *trust* to assume the role. A role trust policy is a required [resource-based policy \(p. 486\)](#) that is attached to a role in IAM. The [principals \(p. 1264\)](#) 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 \(p. 1260\)](#).

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 \(p. 501\)](#).

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 \(p. 15\)](#), 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 (*) to match part of a principal name or ARN in a role's trust policy. For details, see [AWS JSON policy elements: Principal \(p. 1264\)](#).

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 \(p. 1224\)](#). For more information about resource-based policies, see [Cross account resource access in IAM \(p. 526\)](#).

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 \(p. 188\)](#)

- [Provide access to non AWS workloads \(p. 190\)](#)
- [Provide access to IAM users in AWS accounts owned by third parties \(p. 190\)](#)
- [Provide access for services offered by AWS to AWS resources \(p. 193\)](#)
- [Provide access for externally authenticated users \(identity federation\) \(p. 196\)](#)

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 \(p. 190\)](#) 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 \(p. 145\)](#).

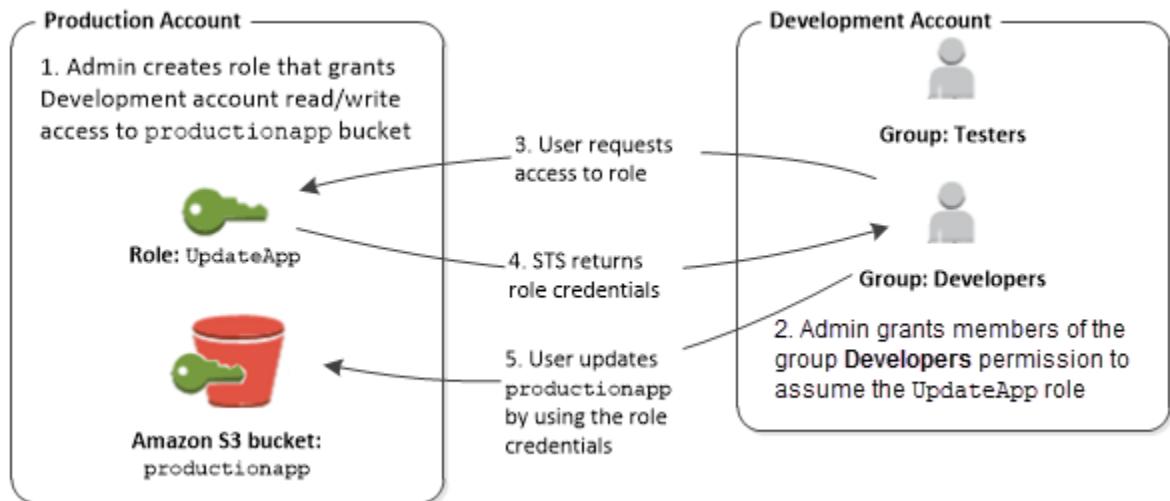
We recommend using this approach to enforce the *principle of least privilege*. That means restricting the use of elevated permissions to only those times when they are needed for specific tasks. With roles you can help prevent accidental changes to sensitive environments, especially if you combine them with [auditing \(p. 471\)](#) 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 \(p. 145\)](#).

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.

More information

For more information, see the following:

- [IAM tutorial: Delegate access across AWS accounts using IAM roles \(p. 40\)](#)

Providing access for non AWS workloads

An [IAM role \(p. 183\)](#) is an object in AWS Identity and Access Management (IAM) that is assigned [permissions \(p. 485\)](#). When you [assume that role \(p. 274\)](#) using an IAM identity or an identity from outside of AWS, it provides you with temporary security credentials for your role session. You might have workloads running in your data center or other infrastructure outside of AWS that need to access your AWS resources. Instead of creating, distributing, and managing long-term access keys, you can use AWS Identity and Access Management Roles Anywhere (IAM Roles Anywhere) to authenticate your non AWS workloads. IAM Roles Anywhere uses X.509 certificates from your certificate authority (CA) to authenticate identities and securely provide access to AWS services with the temporary credentials provided by an IAM role.

To use IAM Roles Anywhere, you set up a CA using [AWS Private Certificate Authority](#) or use a CA from your own PKI infrastructure. After you have set up a CA, you create an object in IAM Roles Anywhere called a *trust anchor* to establish trust between IAM Roles Anywhere and your CA for authentication. You can then configure your existing IAM roles, or create new roles that trust the IAM Roles Anywhere service. When your non AWS workloads authenticate with IAM Roles Anywhere using the trust anchor, they can get temporary credentials for your IAM roles to access your AWS resources.

For more information about configuring IAM Roles Anywhere, see [What is AWS Identity and Access Management Roles Anywhere](#) in the *IAM Roles Anywhere User Guide*.

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 identifier that is known only 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 \(p. 191\)](#).
- 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.

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.

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 (such as an IAM user 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 Security Token Service [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 \(p. 428\)](#).

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 to ensure each external ID is unique.

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 \(p. 193\)](#).

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.

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 \(p. 185\)](#). When a role serves a specialized purpose for a service, it can be categorized as a [service role for EC2 instances \(p. 185\)](#), or a [service-linked role \(p. 185\)](#). 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 \(p. 255\)](#).

The confused deputy problem

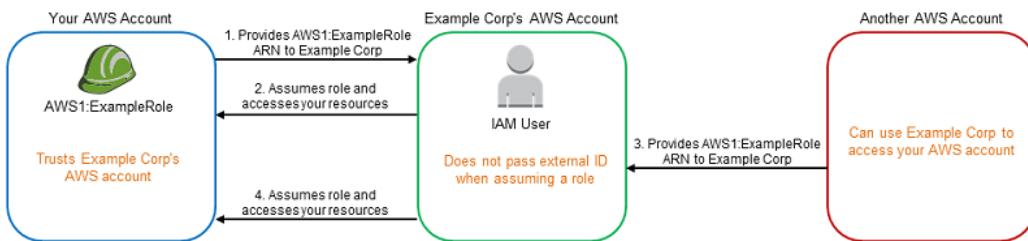
The confused deputy problem is a security issue where an entity that doesn't have permission to perform an action can coerce a more-privileged entity to perform the action. To prevent this, AWS provides tools that help you protect your account if you provide third parties (known as *cross-account*) or other AWS services (known as *cross-service*) access to resources in your account.

At times, you might need to give a third party access to your AWS resources (delegate access). 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. You can use an IAM role to establish a trusted relationship between your AWS account and the Example Corp account. 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. The primary function of the external ID is to address and prevent the confused deputy problem.

In AWS, cross-service impersonation can result in the confused deputy problem. Cross-service impersonation can occur when one service (the *calling service*) calls another service (the *called service*). The calling service can be manipulated to use its permissions to act on another customer's resources in a way it should not otherwise have permission to access.

Cross-account confused deputy prevention

The following diagram illustrates the cross-account confused deputy problem.



This scenario 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."

Example Corp can address the confused deputy problem by requiring that you include the `ExternalId` condition check in the role's trust policy. Example Corp generates a unique `ExternalId` value for each customer and uses that value in its request to assume the role. The `ExternalId` value must be unique among Example Corp's customers and controlled by Example Corp, not its customers. This is why you get it from Example Corp and you don't come up with it on your own. This prevents Example Corp from being a confused deputy and granting access to another account's AWS resources.

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.

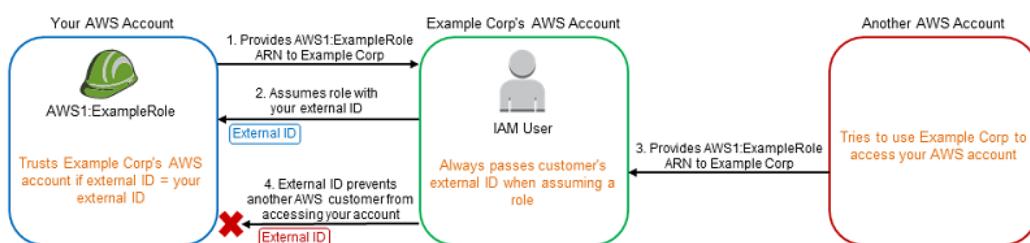
Example Corp gives the external ID value of 12345 to you. You must then add a `Condition` element to the role's trust policy that requires the `sts:ExternalId` value to be 12345, like this:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "Example Corp's AWS Account ID"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "sts:ExternalId": "12345"
        }
      }
    }
  ]
}
```

```
        "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.

The following diagram illustrates this.



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.

Cross-service confused deputy prevention

We recommend using the [aws:SourceArn](#) and [aws:SourceAccount](#) global condition context keys in resource-based policies to limit the permissions that a service has to a specific resource. Use `aws:SourceArn` if you want only one resource to be associated with the cross-service access. Use `aws:SourceAccount` if you want to allow any resource in that account to be associated with the cross-service use.

The most effective way to protect against the confused deputy problem is to use the aws:SourceArn global condition context key with the full ARN of the resource in your resource-based policies. If you don't know the full ARN of the resource or if you are specifying multiple resources, use the aws:SourceArn global condition context key with wildcards (*) for the unknown portions of the ARN. For example, `arn:aws:servicename:*:123456789012:*`.

If the `aws:SourceArn` value does not contain the account ID, such as an Amazon S3 bucket ARN, you must use both global condition context keys to limit permissions.

For non-service-linked role [trust policies \(p. 186\)](#), every service in the trust policy has performed the `iam:PassRole` action to verify that the role is in the same account as the calling service. As a result, using `aws:SourceAccount` with those trust policies is not necessary. Using `aws:SourceArn` in a trust policy allows you to specify resources a role can be assumed on behalf of, such as a Lambda function ARN. Some AWS services use `aws:SourceAccount` and `aws:SourceArn` in trust policies for newly created roles, but using the keys isn't required for existing roles in your account.

[Cross-service confused deputy prevention for AWS Security Token Service](#)

Many AWS services require that you use roles to allow the service to access another service's resources on your behalf. A role that a service assumes to perform actions on your behalf is called a [service role \(p. 185\)](#). A role requires two policies: a role trust policy that specifies the principal that is allowed to assume the role and a permissions policy that specifies what can be done with the role. A role trust policy is the only type of resource-based policy in IAM. Other AWS services have resource-based policies, such as an Amazon S3 bucket policy.

When a service assumes a role on your behalf, the service principal must be allowed to perform the [sts:AssumeRole](#) action in the role trust policy. When a service calls `sts:AssumeRole`, AWS STS returns a set of temporary security credentials that the service principal uses to access the resources that are permitted by the role's permissions policy. When a service assumes a role in your account, you can include the `aws:SourceAccount` and `aws:SourceArn` global condition context keys in your role trust policy to limit access to the role to only requests that are generated by expected resources.

For example, in AWS Systems Manager Incident Manager, you must choose a role to allow Incident Manager to run a Systems Manager automation document on your behalf. The automation document can include automated response plans for incidents that are initiated by CloudWatch alarms or EventBridge events. In the following role trust policy example, you can use the `aws:SourceArn` condition key to restrict access to the service role based on the incident record's ARN. Only incident records that are created from the response plan resource `myresponseplan` are able to use this role.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "ssm-incidents.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:ssm-incidents:*:111122223333:incident-
record/myresponseplan/*"
                }
            }
        }
    ]
}
```

[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).

Note

As a security best practice, we recommend you manage user access in [IAM Identity Center](#) with identity federation instead of creating IAM users. For information about specific situations where an IAM user is required, see [When to create an IAM user \(instead of a role\)](#) in the [IAM User Guide](#).

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 \(p. 200\)](#).

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 \(p. 199\)](#).

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 \(p. 205\)](#).

Federating users by creating a custom identity broker application

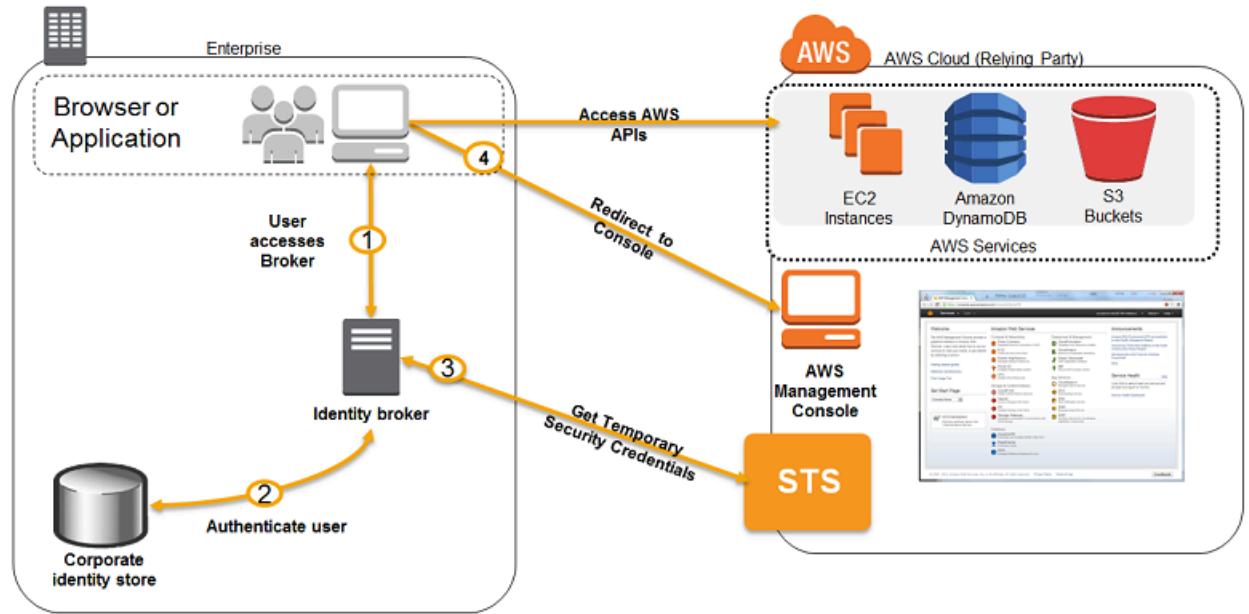
If your identity store is not compatible with SAML 2.0, then you can build a custom identity broker application to perform a similar function. The broker application authenticates users, requests temporary credentials for users from AWS, and then provides them to the user to access AWS resources.

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 \(p. 426\)](#) and [Controlling permissions for temporary security credentials \(p. 441\)](#).) 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 \(p. 428\)](#). 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 \(p. 230\)](#).

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.

Note

As a security best practice, we recommend you manage user access in [IAM Identity Center](#) with identity federation instead of creating IAM users. For information about specific situations where an IAM user is required, see [When to create an IAM user \(instead of a role\)](#) in the [IAM User Guide](#).

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 \(p. 199\)](#)
- [About SAML 2.0-based federation \(p. 205\)](#)

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 \(p. 209\)](#)

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 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 \(p. 200\)](#).

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 \(p. 200\)](#)
- [Using web identity federation API operations for mobile apps \(p. 201\)](#)

- [Identifying users with web identity federation \(p. 203\)](#)
- [Additional resources for web identity federation \(p. 204\)](#)

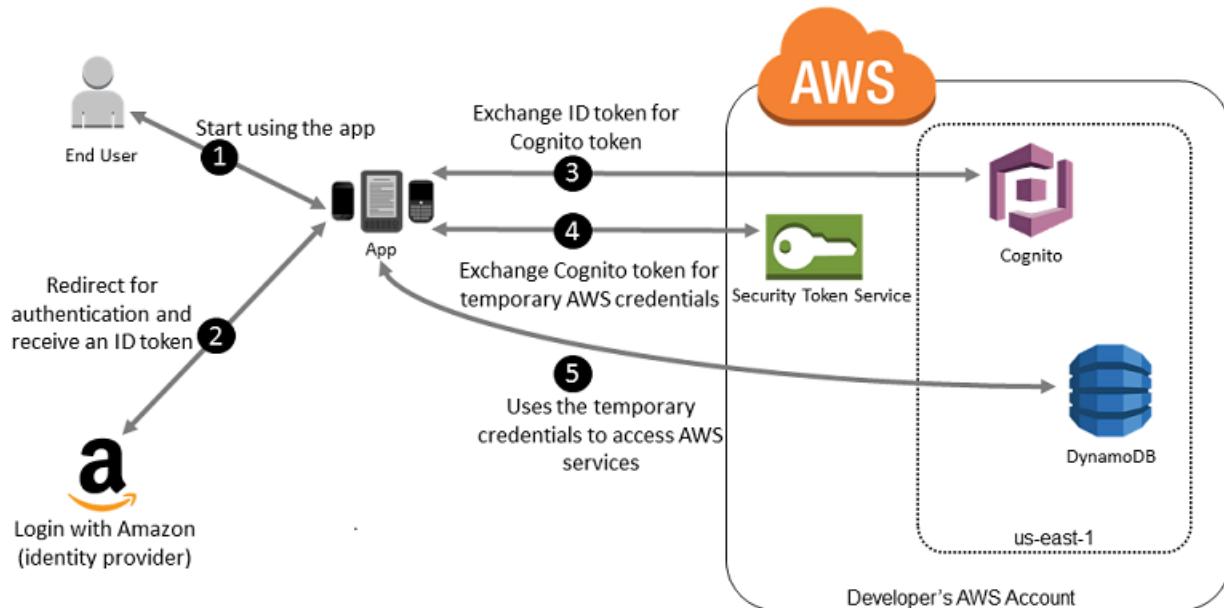
Using Amazon Cognito for mobile apps

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](#), [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 an [Amazon Cognito identity pool](#) in 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 the Amazon Cognito API operations `GetId` and `GetCredentialsForIdentity` to exchange the Login with Amazon ID token for an Amazon Cognito token. Amazon Cognito, which has been configured to trust your Login with Amazon project, generates a token that it exchanges for temporary session credentials with AWS STS.
4. The app receives temporary security credentials from Amazon Cognito. Your app can also use the Basic (Classic) workflow in Amazon Cognito to retrieve tokens from AWS STS using `AssumeRoleWithWebIdentity`. For more information, see [Identity pools \(federated identities\) authentication flow](#) in the Amazon Cognito Developer Guide.
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 the 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. Integrate [AWS Amplify](#) 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:

- [Sign in \(Android\)](#) in the AWS Amplify Framework Documentation.
- [Sign in \(iOS\)](#) in the AWS Amplify Framework Documentation.

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.

Important

If you use an OIDC identity provider from Google, Facebook, or Amazon Cognito, do not create a separate IAM identity provider in the AWS Management Console. AWS has these OIDC identity providers built-in and available for your use. Skip the following step and move directly to creating new roles using your identity provider.

2. If you use an IdP other than Google, Facebook or Amazon Cognito compatible with OIDC, then create an IAM identity provider entity for it.
3. In IAM, [create one or more roles \(p. 260\)](#). For each role, define who can assume the role (the trust policy) and what permissions the app's users have (the permissions policy). Typically, you create one role for each IdP that an app supports. For example, you might create a role assumed by an app if the user signs in through Login with Amazon, a second role for the same app if the user signs in through Facebook, and a third role for the app if 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 in [Creating a role for a third-party Identity Provider \(federation\) \(p. 260\)](#).
4. In your application, authenticate your users with the IdP. The specifics of how to do this vary both according to which IdP you use (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 \(p. 203\)](#) 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 uses 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 signs in using Login with Amazon, and that the user uses 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 \(p. 1372\)](#).

Additional resources for web identity federation

The following resources can help you learn more about web identity federation:

- [Amazon Cognito Identity](#) in the *Amplify Libraries for Android Guide* and [Amazon Cognito Identity](#) in the *Amplify Libraries for Swift 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 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:

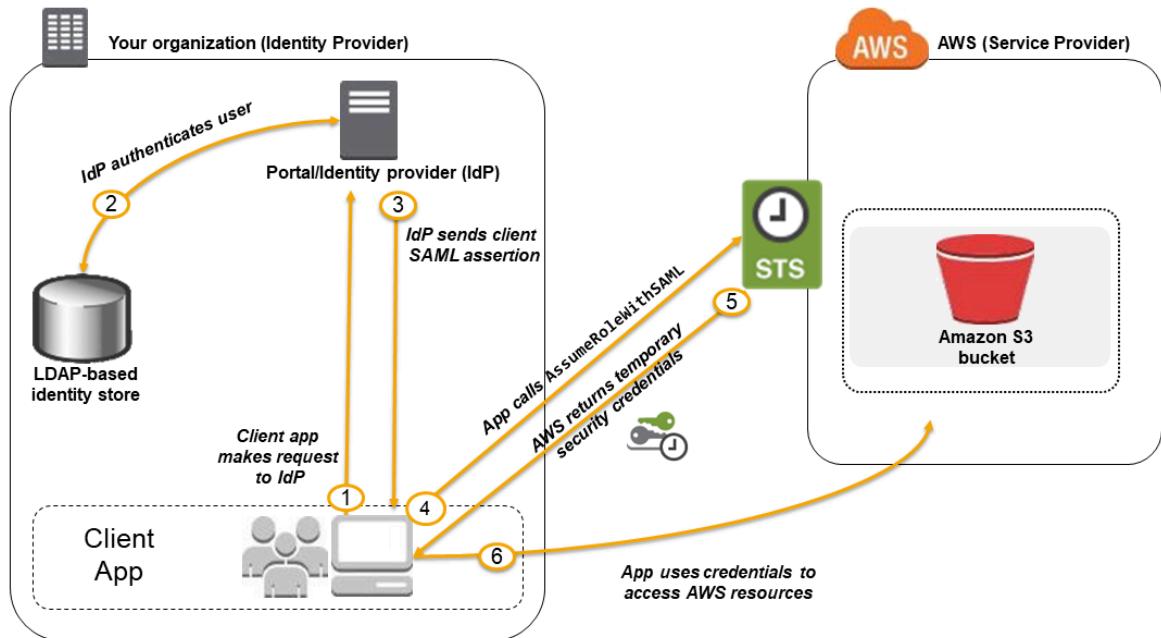
- [Federated access to allow a user or application in your organization to call AWS API operations \(p. 205\)](#). You use a SAML assertion (as part of the authentication response) that is generated in your organization to get temporary security credentials. This scenario is similar to other federation scenarios that IAM supports, like those described in [Requesting temporary security credentials \(p. 428\)](#) and [About web identity federation \(p. 199\)](#). 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 \(p. 230\)](#). 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 \(p. 233\)](#).

Topics

- [Using SAML-based federation for API access to AWS \(p. 205\)](#)
- [Overview of configuring SAML 2.0-based federation \(p. 206\)](#)
- [Overview of the role to allow SAML-federated access to your AWS resources \(p. 207\)](#)
- [Uniquely identifying users in SAML-based federation \(p. 208\)](#)

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 \(p. 223\)](#), like Microsoft Active Directory Federation Service (AD FS, part of Windows Server), Shibboleth, or another compatible SAML 2.0 provider.

Note

To improve federation resiliency, we recommend that you configure your IdP and AWS federation to support multiple SAML sign-in endpoints. For details, see the AWS Security Blog article [How to use regional SAML endpoints for failover](#).

To configure your organization's IdP and AWS to trust each other

1. Register AWS as a service provider (SP) with the IdP of your organization. Use the SAML metadata document from <https://region-code.signin.aws.amazon.com/static/saml-metadata.xml>

For a list of possible `region-code` values, see the **Region** column in [AWS Sign-In endpoints](#).

You can optionally use the SAML metadata document from <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 \(p. 218\)](#).
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 \(federation\) \(p. 260\)](#).

Note

SAML IDPs used in a role trust policy must be in the same account that the role is in.

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 \(p. 205\)](#) 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 \(p. 225\)](#).

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 \(p. 230\)](#).

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:saml-provider/ExampleOrgSSOProvider"},
      "Action": "sts:AssumeRoleWithSAML",
      "Condition": {
        "StringEquals": {
          "sts:RequestorSSO": "https://openidp.feide.no"
        }
      }
    }
  ]
}
```

```
        "StringEquals": {
            "saml:aud": "https://signin.aws.amazon.com/saml",
            "saml:iss": "https://openidp.feide.no"
        },
        "ForAllValues:StringLike": {"saml:edupersonaffiliation": ["staff"]}
    }
}
```

Note

SAML IDPs used in a role trust policy must be in the same account that the role is in.

For more information about the SAML keys that you can check in a policy, see [Available keys for SAML-based AWS STS federation \(p. 1376\)](#).

You can include regional endpoints for the saml:aud attribute at <https://region-code.signin.aws.amazon.com/static/saml-metadata.xml>. For a list of possible *region-code* values, see the **Region** column in [AWS Sign-In endpoints](#).

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 \(p. 1376\)](#).

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 \(p. 1376\)](#).

- `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, `_ccb88bf52c2510eabe00c1642d4643f41430fe25e3`).
- `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 \(p. 225\)](#).

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 \(p. 225\)](#).

Creating IAM identity providers

Note

We recommend that you require your human users to use temporary credentials when accessing AWS. Have you considered using AWS IAM Identity Center (successor to AWS Single Sign-On)? You can use IAM Identity Center to centrally manage access to multiple AWS accounts and provide users with MFA-protected, single sign-on access to all their assigned accounts from one place. With IAM Identity Center, you can create and manage user identities in IAM Identity Center or easily connect to your existing SAML 2.0 compatible identity provider. For more information, see [What is IAM Identity Center?](#) in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*.

You can use an external identity provider (IdP) to manage user identities outside of AWS. An external IdP can provide identity information to AWS using either OpenID Connect (OIDC) or Security Assertion Markup Language (SAML). Examples of well-known OIDC identity providers are: Login with Amazon, Facebook, and Google. Examples of well-known SAML identity providers are: Shibboleth and Active Directory Federation Services.

When you want to configure federation with an external IdP, you create an IAM *identity provider* to inform AWS about the external IdP and its configuration. This establishes "trust" between your AWS account and the external IdP. The following topics include details about how to create an IAM identity provider for each of the external IdP types.

Topics

- [Creating OpenID Connect \(OIDC\) identity providers \(p. 210\)](#)
- [Creating IAM SAML identity providers \(p. 218\)](#)

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" \(p. 199\)](#).

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 \(federation\) \(p. 260\)](#).

Topics

- [Creating and managing an OIDC provider \(console\) \(p. 210\)](#)
- [Creating and managing an IAM OIDC identity provider \(AWS CLI\) \(p. 212\)](#)
- [Creating and managing an OIDC Identity Provider \(AWS API\) \(p. 213\)](#)
- [Obtaining the thumbprint for an OpenID Connect Identity Provider \(p. 215\)](#)

Creating and managing an OIDC provider (console)

Follow these instructions to create and manage an IAM OIDC identity provider in the AWS Management Console.

Important

If you are using an OIDC identity provider from either Google, Facebook, or Amazon Cognito, do not create a separate IAM identity provider using this procedure. These OIDC identity providers are already built-in to AWS and are available for your use. Instead, follow the steps to create new roles for your identity provider, see [Creating a role for web identity or OpenID Connect Federation \(console\) \(p. 262\)](#).

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.

Note

AWS secures communication with some OIDC identity providers (IdPs) through our library of trusted root certificate authorities (CAs) instead of using a certificate thumbprint to verify your IdP server certificate. These OIDC IdPs include Auth0, GitHub, Google, and those that

use an Amazon S3 bucket to host a JSON Web Key Set (JWKS) endpoint. In these cases, your legacy thumbprint remains in your configuration, but is no longer used for validation.

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://**.
 - The URL should not contain a port number.
 - 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 thumbprint for an OpenID Connect Identity Provider \(p. 215\)](#).
7. For **Audience**, type the client ID of the application that you registered with the IdP and received in [Step 1](#), and that 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. (Optional) For **Add tags**, you can add key-value pairs to help you identify and organize your IdPs. You can also use tags to control access to AWS resources. To learn more about tagging IAM OIDC identity providers, see [Tagging OpenID Connect \(OIDC\) identity providers \(p. 409\)](#). Choose **Add tag**. Enter values for each tag key-value pair.
9. Verify the information that you have provided. When you are done choose **Add provider**.
10. 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 \(federation\) \(p. 260\)](#).

Note

SAML IDPs used in a role trust policy must be in the same account that the role is in.

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 tag an existing IAM OIDC identity provider (AWS CLI)

- To tag an existing IAM OIDC identity provider, run the following command:
 - [`aws iam tag-open-id-connect-provider`](#)

To list tags for an existing IAM OIDC identity provider (AWS CLI)

- To list tags for an existing IAM OIDC identity provider, run the following command:
 - [`aws iam list-open-id-connect-provider-tags`](#)

To remove tags on an IAM OIDC identity provider (AWS CLI)

- To remove tags on an existing IAM OIDC identity provider, run the following command:
 - [aws iam untag-open-id-connect-provider](#)

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 tag an existing IAM OIDC identity provider (AWS API)

- To tag an existing IAM OIDC identity provider, call the following operation:
 - [TagOpenIDConnectProvider](#)

To list tags for an existing IAM OIDC identity provider (AWS API)

- To list tags for an existing IAM OIDC identity provider, call the following operation:
 - [ListOpenIDConnectProviderTags](#)

To remove tags on an existing IAM OIDC identity provider (AWS API)

- To remove tags on an existing IAM OIDC identity provider, call the following operation:
 - [UntagOpenIDConnectProvider](#)

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 thumbprint for an OpenID Connect Identity Provider

When you [create an OpenID Connect \(OIDC\) identity provider \(p. 210\)](#) in IAM, you must supply a thumbprint. IAM requires the thumbprint for the top 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 AWS and your IdP. This should be issued by a well-known public root CA, such as AWS Certificate Manager. This enables the client to check the reliability and status of the certificate.
- The second is used to encrypt tokens, and should be signed by a private or public root CA.

You can create an IAM OIDC identity provider with [the AWS Command Line Interface, the Tools for Windows PowerShell, or the IAM API \(p. 212\)](#). 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 \(p. 210\)](#), 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 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 \(p. 217\)](#) and [Configure OpenSSL \(p. 217\)](#).

Note

AWS secures communication with some OIDC identity providers (IdPs) through our library of trusted root certificate authorities (CAs) instead of using a certificate thumbprint to verify your IdP server certificate. These OIDC IdPs include Auth0, GitHub, Google, and those that use an Amazon S3 bucket to host a JSON Web Key Set (JWKS) endpoint. In these cases, your legacy thumbprint remains in your configuration, but is no longer used for validation.

2. Start with the OIDC IdP 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 server name.

3. In the displayed document, use your web browser **Find** feature to locate the text "jwks_uri". Immediately following the text "jwks_uri", there is a colon (:) followed by a URL. Copy the fully qualified domain name of the URL. Do not include `https://` or any path that comes after the top-level domain.

```
{
  "issuer": "https://accounts.example.com",
  "authorization_endpoint": "https://accounts.example.com/o/oauth2/v2/auth",
  "device_authorization_endpoint": "https://oauth2.exampleapis.com/device/code",
  "token_endpoint": "https://oauth2.exampleapis.com/token",
  "userinfo_endpoint": "https://openidconnect.exampleapis.com/v1/userinfo",
  "revocation_endpoint": "https://oauth2.exampleapis.com/revoke",
  "jwks_uri": "https://www.exampleapis.com/oauth2/v3/certs",
  ...
}
```

4. Use the OpenSSL command line tool to run 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 displayed (at the end of the command output). This contains the certificate of the top intermediate CA in the certificate authority chain.

```
-----BEGIN CERTIFICATE-----
MIICiTCCAFICCQD6m7oRw0uX0jANBgkqhkiG9w0BAQUFADCBiDELMAkGA1UEBhMC
VVMxCzAJBgNVBAgTA1dBMRAwDgYDVQQHEwdTZWF0dGx1MQ8wDQYDVQQKEwZBbWF6
b24xFDASBgNVBAsTC01BTSDb25zb2x1MRIwEAYDVQQDEwlUZXN0Q21sYWMxHzAd
BgkqhkiG9w0BCQEWEg5vb251QGFtYXpvbi5jb20wHhcNMTEwNDI1MjA0NTIxWhcN
MTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTA1dBMRAwDgYD
VQQHEwdTZWF0dGx1MQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC01BTSDb25z
b2x1MRIwEAYDVQQDEwlUZXN0Q21sYWMxHzAdBgkqhkiG9w0BCQEWEg5vb251QGFt
YXpvbi5jb20wgZ8wDQYJKoZIhvcNAQEBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ
21uUSfwfEvySWtC2XADZ4nB+BLYgVIk60CpiwsZ3G93vUEI03IyNoH/f0wYK8m9T
rDHudUZg3qX4wALG5M43q7Wgc/MbQITx0USQv7c7ugFFDzQGBzZswY6786m86gpE
Ibb30hjZnzcvQaRHhd1QWIMm2nrAgMBAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4
nUhVvxYUntneD9+h8Mg9q6q+auNKyExzyLwax1Ao07JHidbtS4J5iNmZgXL0Fkb
FFBjvSfpJI1J00zbhNYS5f6GuoEDmFJl0ZxBhjJnyp3780D8uTs7fLvjx79LjSTb
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 run the following command.

```
openssl x509 -in certificate.crt -fingerprint -sha1 -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

1. At the command line, set the OpenSSL_HOME variable to the location of the OpenSSL installation:

```
$ export OpenSSL_HOME=path_to_your_OpenSSL_installation
```

2. Set the path to include the OpenSSL installation:

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

Note

Any changes you make to environment variables 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

1. Open a **Command Prompt** window.
2. Set the `OpenSSL_HOME` variable to the location of the OpenSSL installation:

```
C:\> set OpenSSL_HOME=path_to_your_OpenSSL_installation
```

3. Set the `OpenSSL_CONF` variable to the location of the configuration file in your OpenSSL installation:

```
C:\> set OpenSSL_CONF=path_to_your_OpenSSL_installation\bin\openssl.cfg
```

4. 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 \(p. 205\)](#).

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 \(federation\) \(p. 260\)](#).

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 \(p. 222\)](#).

Topics

- [Creating and managing an IAM SAML identity provider \(console\) \(p. 219\)](#)
- [Creating and managing an IAM SAML Identity Provider \(AWS CLI\) \(p. 220\)](#)
- [Creating and managing an IAM SAML identity provider \(AWS API\) \(p. 221\)](#)
- [Configuring your SAML 2.0 IdP with relying party trust and adding claims \(p. 222\)](#)
- [Integrating third-party SAML solution providers with AWS \(p. 223\)](#)
- [Configuring SAML assertions for the authentication response \(p. 225\)](#)

Creating and managing an IAM SAML 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 \(p. 223\)](#).

Important

This metadata file includes the issuer name, expiration information, and keys that can be used to validate the SAML authentication response (assertions) received from the IdP. The metadata file must be encoded in UTF-8 format without a byte order mark (BOM). To remove the BOM, you can encode the file as UTF-8 using a text editing tool, such as Notepad++.

The x.509 certificate included as part of the SAML metadata document must use a key size of at least 1024 bits. Also, the x.509 certificate must also be free of any repeated extensions. You can use extensions, but the extensions can only appear once in the certificate. If the x.509 certificate does not meet either condition, IdP creation fails and returns an "Unable to parse metadata" error.

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. (Optional) For **Add tags** you can add key-value pairs to help you identify and organize your IdPs. You can also use tags to control access to AWS resources. To learn more about tagging SAML identity providers, see [Tagging IAM SAML identity providers \(p. 411\)](#).

Choose **Add tag**. Enter values for each tag key-value pair.

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 \(federation\) \(p. 260\)](#).

Note

SAML IDPs used in a role trust policy must be in the same account that the role is in.

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 \(p. 223\)](#).

Important

This metadata file includes the issuer name, expiration information, and keys that can be used to validate the SAML authentication response (assertions) received from the IdP. The metadata file must be encoded in UTF-8 format without a byte order mark (BOM). To remove the BOM, you can encode the file as UTF-8 using a text editing tool, such as Notepad++.

The x.509 certificate included as part of the SAML metadata document must use a key size of at least 1024 bits. Also, the x.509 certificate must also be free of any repeated extensions. You can use extensions, but the extensions can only appear once in the certificate. If the x.509 certificate does not meet either condition, IdP creation fails and returns an "Unable to parse metadata" error.

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 tag an existing IAM identity provider (AWS CLI)

- Run this command: [`aws iam tag-saml-provider`](#)

To list tags for existing IAM identity provider (AWS CLI)

- Run this command: [`aws iam list-saml-provider-tags`](#)

To remove tags on an existing IAM identity provider (AWS CLI)

- Run this command: [`aws iam untag-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 \(p. 223\)](#).

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 tag an existing IAM identity provider (AWS API)

- Call this operation: [`TagSAMLProvider`](#)

To list tags for an existing IAM identity provider (AWS API)

- Call this operation: [`ListSAMLProviderTags`](#)

To remove tags on an existing IAM identity provider (AWS API)

- Call this operation: [`UntagSAMLProvider`](#)

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, use `https://region-code.signin.aws.amazon.com/static/saml-metadata.xml` or `https://signin.aws.amazon.com/static/saml-metadata.xml`. For a list of possible `region-code` values, see the **Region** column in [AWS Sign-In endpoints](#).

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 \(p. 207\)](#). 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 \(p. 225\)](#). 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 \(p. 223\)](#). This topic provides links to documentation provided by third-party organizations about how to integrate identity solutions with AWS.

Note

To improve federation resiliency, we recommend that you configure your IdP and AWS federation to support multiple SAML sign-in endpoints. For details, see the AWS Security Blog article [How to use regional SAML endpoints for failover](#).

Integrating third-party SAML solution providers with AWS

Note

We recommend that you require your human users to use temporary credentials when accessing AWS. Have you considered using AWS IAM Identity Center (successor to AWS Single Sign-On)? You can use IAM Identity Center to centrally manage access to multiple AWS accounts and provide users with MFA-protected, single sign-on access to all their assigned accounts from one place. With IAM Identity Center, you can create and manage user identities in IAM Identity Center or easily connect to your existing SAML 2.0 compatible identity provider. For more information, see [What is IAM Identity Center?](#) in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*.

The following links help you configure third-party SAML 2.0 identity provider (IdP) solutions to work with AWS federation.

Tip

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. 417) . For more information, see Auth0 Announces Partnership with AWS for IAM Session Tags .
Azure Active Directory (Azure AD)	Tutorial: Azure AD SSO integration with AWS Single-Account Access – This tutorial on the Microsoft website describes how to set up Azure AD as an identity provider (IdP) using SAML federation.
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.
CyberArk	Configure CyberArk to provide Amazon Web Services (AWS) access to users logging in through SAML single sign-on (SSO) from the CyberArk User Portal.
ForgeRock	The ForgeRock Identity Platform integrates with AWS. You can configure ForgeRock to pass session tags (p. 417) . 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. 417) . For more information, see IBM Cloud Identity IDaaS one of first to support AWS session tags .
JumpCloud	Granting Access via IAM Roles for Single Sign On (SSO) with Amazon AWS – This article on the JumpCloud website

Solution	More information
	describes how to set up and enable SSO based on IAM roles for AWS.
Matrix42	MyWorkspace Getting Started Guide – This guide describes how to integrate AWS identity services with Matrix42 MyWorkspace.
Microsoft Active Directory Federation Services (AD FS)	<p>Field Notes: Integrating Active Directory Federation Service with AWS IAM Identity Center (successor to AWS Single Sign-On) – This post on the AWS Architecture Blog explains the authentication flow between AD FS and AWS IAM Identity Center (successor to AWS Single Sign-On) (IAM Identity Center). IAM Identity Center supports identity federation with SAML 2.0, allowing integration with AD FS solutions. Users can sign in to the IAM Identity Center portal with their corporate credentials reducing the admin overhead of maintaining separate credentials on IAM Identity Center. You can also configure AD FS to pass session tags (p. 417). 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	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.
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. 417). For more information, see Okta and AWS Partner to Simplify Access Via Session Tags.</p>
Okta	AWS Account Federation – This section on the Okta website describes how to set up and enable IAM Identity Center for AWS.
OneLogin	<p>From the OneLogin Knowledgebase, search for SAML AWS for a list of articles that explain how to set up IAM Identity Center functionality between OneLogin and AWS for a single-role and multi-role scenarios. You can configure OneLogin to pass session tags (p. 417). For more information, see OneLogin and Session Tags: Attribute-Based Access Control for AWS Resources.</p>

Solution	More information
Ping Identity	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. 417) . For more information, see Announcing Ping Identity Support for Attribute-Based Access Control in AWS .
RadiantLogic	Radiant Logic Technology Partners – Radiant Logic's RadiantOne Federated Identity Service integrates with AWS to provide an identity hub for SAML-based SSO.
RSA	RSA Link is an online community that facilitates information sharing and discussion. You can configure RSA to pass session tags (p. 417) . For more information, see Simplify Identity Access and Assurance Decisions on AWS with RSA SecurID and Session Tags .
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. 417) .

For more details, see the [IAM Partners](#) page on the AWS website.

Configuring SAML assertions for the authentication response

After you have verified a user's identity in your organization, the external identity provider (IdP) sends an authentication response to the AWS SAML endpoint at <https://region-code.signin.amazonaws.com/saml>. For a list of potential `region-code` replacements, see the **Region** column in [AWS Sign-In endpoints](#). 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 \(p. 229\)](#).

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://region-code.signin.aws.amazon.com/saml`. For a list of possible `region-code` values, see the **Region** column in [AWS Sign-In endpoints](#). For the AWS value, you can also use `https://signin.aws.amazon.com/static/saml`, as shown in the following example.

NameID elements can have the value persistent, transient, or consist of the full Format URI as provided by the IdP solution. A value of persistent indicates that the value in NameID is the same for a user between sessions. If the value is transient, the user has a different NameID value for each session. Single sign-on interactions support the following types of identifiers:

- `urn:oasis:names:tc:SAML:2.0:nameid-format:persistent`
- `urn:oasis:names:tc:SAML:2.0:nameid-format:transient`
- `urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress`
- `urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified`
- `urn:oasis:names:tc:SAML:1.1:nameid-format:X509SubjectName`
- `urn:oasis:names:tc:SAML:1.1:nameid-format:WindowsDomainQualified Name`
- `urn:oasis:names:tc:SAML:2.0:nameid-format:kerberos`
- `urn:oasis:names:tc:SAML:2.0:nameid-format:entity`

```
<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>
```

Important

The `saml:aud` context key comes from the SAML `recipient` attribute because it is the SAML equivalent to the OIDC audience field, for example, `accounts.google.com:aud`.

PrincipalTag SAML 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 \(p. 417\)](#).

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 \(p. 185\)](#). 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>
```

Role SAML 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 choose which role to assume when they use WebSSO to sign in to 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>
```

RoleSessionName SAML 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 temporary credentials that are issued when the role is assumed. You can use this to associate the temporary credentials with the user who is using your application. 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: . , + = @ - (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>
```

SessionDuration SAML 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://region-code.signin.amazonaws.com/saml>. For a list of possible *region-code* values, see the **Region** column in [AWS Sign-In endpoints](#). You can optionally use the following URL: <https://signin.amazonaws.com/static/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 \(p. 382\)](#).

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>
```

SourceIdentity SAML attribute

(Optional) You can use an Attribute element with the Name attribute set to <https://aws.amazon.com/SAML/Attributes/SourceIdentity>. This element contains one AttributeValue element that provides an identifier for the person or application that is using an IAM role. The value for source identity persists when you use the SAML session to assume another role in AWS known as [role chaining \(p. 185\)](#). The value for source identity is present in the request for every action taken during the role session. The value that is set cannot be changed during the role session. Administrators can then use AWS CloudTrail logs to monitor and audit the source identity information to determine who performed actions with shared roles.

The value in the AttributeValue element must be between 2 and 64 characters long, can contain only alphanumeric characters, underscores, and the following characters: . , + = @ - (hyphen). It cannot contain spaces. The value is typically an attribute that is associated with the user such as 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). For more information about using source identity, see [Monitor and control actions taken with assumed roles \(p. 444\)](#).

Important

If your SAML assertion is configured to use the [SourceIdentity \(p. 228\)](#) attribute, then your role trust policy must also include the sts:SetSourceIdentity action, otherwise the assume role operation will fail. For more information about using source identity, see [Monitor and control actions taken with assumed roles \(p. 444\)](#).

To pass a source identity attribute, include the AttributeValue element that specifies the value of the source identity. For example, to pass the source identity DiegoRamirez use the following attribute.

```
<Attribute Name="https://aws.amazon.com/SAML/Attributes/SourceIdentity">
<AttributeValue>DiegoRamirez</AttributeValue>
```

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 \(p. 1294\)](#) 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 (Name key)	Maps to this AWS context key (FriendlyName key)	Type
urn:oid:1.3.6.1.4.1.5923.1.1.1.1	eduPersonAffiliation	List of strings
urn:oid:1.3.6.1.4.1.5923.1.1.1.2	eduPersonNickname	List of strings
urn:oid:1.3.6.1.4.1.5923.1.1.1.3	eduPersonOrgDN	String
urn:oid:1.3.6.1.4.1.5923.1.1.1.4	eduPersonOrgUnitDN	List of strings
urn:oid:1.3.6.1.4.1.5923.1.1.1.5	eduPersonPrimaryAffiliation	String
urn:oid:1.3.6.1.4.1.5923.1.1.1.6	eduPersonPrincipalName	String
urn:oid:1.3.6.1.4.1.5923.1.1.1.7	eduPersonEntitlement	List of strings
urn:oid:1.3.6.1.4.1.5923.1.1.1.8	eduPersonPrimaryOrgUnitDN	String
urn:oid:1.3.6.1.4.1.5923.1.1.1.9	eduPersonScopedAffiliation	List of strings
urn:oid:1.3.6.1.4.1.5923.1.1.1.10	eduPersonTargetedID	List of strings
urn:oid:1.3.6.1.4.1.5923.1.1.1.11	eduPersonAssurance	List of strings
urn:oid:1.3.6.1.4.1.5923.1.2.1.2	eduOrgHomePageURI	List of strings
urn:oid:1.3.6.1.4.1.5923.1.2.1.3	eduOrgIdentityAuthNPolicyURI	List of strings
urn:oid:1.3.6.1.4.1.5923.1.2.1.4	eduOrgLegalName	List of strings
urn:oid:1.3.6.1.4.1.5923.1.2.1.5	eduOrgSuperiorURI	List of strings
urn:oid:1.3.6.1.4.1.5923.1.2.1.6	eduOrgWhitePagesURI	List of strings
urn:oid:2.5.4.3	cn	List of strings

Active Directory attributes

AD attribute	Maps to this AWS context key	Type
<code>http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name</code>	<code>name</code>	String
<code>http://schemas.xmlsoap.org/claims/CommonName</code>	<code>commonName</code>	String
<code>http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname</code>	<code>givenName</code>	String
<code>http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname</code>	<code>surname</code>	String
<code>http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress</code>	<code>mail</code>	String
<code>http://schemas.microsoft.com/ws/2008/06/identity/claims/primarygroupsid</code>	<code>uid</code>	String

X.500 attributes

X.500 attribute	Maps to this AWS context key	Type
2.5.4.3	<code>commonName</code>	String
2.5.4.4	<code>surname</code>	String
2.4.5.42	<code>givenName</code>	String
2.5.4.45	<code>x500UniqueIdentifier</code>	String
0.9.2342.19200300100.1.1	<code>uid</code>	String
0.9.2342.19200300100.1.3	<code>mail</code>	String
0.9.2342.19200300.100.1.45	<code>organizationStatus</code>	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:

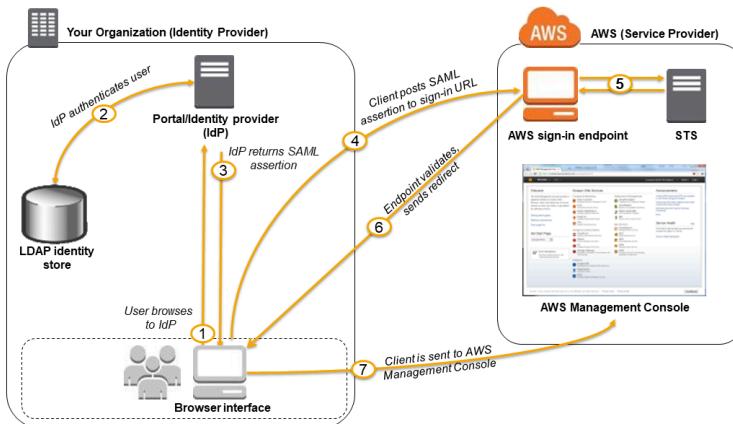
- AWS CLI: [Switching to an IAM role \(AWS CLI\) \(p. 285\)](#)
- Tools for Windows PowerShell: [Switching to an IAM role \(Tools for Windows PowerShell\) \(p. 289\)](#)
- AWS API: [Switching to an IAM role \(AWS API\) \(p. 291\)](#)

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 \(p. 205\)](#) because this workflow opens the AWS Management Console on behalf of the user. This requires the use of the AWS sign-in 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 \(p. 417\)](#). 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 \(p. 223\)](#) 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 \(p. 218\)](#).

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:saml-provider/  
ExampleOrgSSOProvider"},  
        "Action": "sts:AssumeRoleWithSAML",  
        "Condition": {"StringEquals": {  
            "saml:edupersonorgdn": "ExampleOrg",  
            "saml:aud": "https://signin.aws.amazon.com/saml"  
        }}  
    }]  
}
```

Note

SAML IDPs used in a role trust policy must be in the same account that the role is in.

You can include regional endpoints for the saml:aud attribute at <https://region-code.signin.aws.amazon.com/static/saml-metadata.xml>. For a list of possible [region-code](#) values, see the **Region** column in [AWS Sign-In endpoints](#).

For the [permission policy \(p. 485\)](#) 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 \(p. 598\)](#), 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 \(console\) \(p. 268\)](#).

Finish configuration and create SAML assertions

Notify your SAML IdP that AWS is your service provider by installing the saml-metadata.xml file found at <https://region-code.signin.aws.amazon.com/static/saml-metadata.xml> or <https://signin.aws.amazon.com/static/saml-metadata.xml>. For a list of possible [region-code](#) values, see the **Region** column in [AWS Sign-In endpoints](#).

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 \(p. 223\)](#) 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 \(p. 225\)](#).

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 \(p. 230\)](#).

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 \(p. 274\)](#). To learn how to pass optional session tags when you obtain your security credentials, see [Passing session tags in AWS STS \(p. 417\)](#).
 - 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. When you use DurationSeconds in an AssumeRole* operation, you must call it as an IAM user with long-term credentials. Otherwise, the call to the federation endpoint in step 3 fails. To learn how to view or change the maximum value for a role, see [View the maximum session duration setting for a role \(p. 276\)](#).
 - 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.
- Using the credentials for one role to assume a different role is called [role chaining \(p. 185\)](#). 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 \(p. 374\)](#), those applications are not subject to this limitation.

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 \(p. 234\)](#)
- [Example code using Python \(p. 237\)](#)
- [Example code using Java \(p. 239\)](#)
- [Example showing how to construct the URL \(Ruby\) \(p. 241\)](#)

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 session token. For more information about creating temporary credentials, see [Temporary security credentials in IAM \(p. 426\)](#).

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 \(p. 441\)](#).

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": "*** session 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. **Note**

AWS supports POST requests here.

Send your request to the AWS federation endpoint:

`https://region-code.signin.aws.amazon.com/federation`

For a list of possible `region-code` values, see the **Region** column in [AWS Sign-In endpoints](#). You can optionally use the default AWS Sign-In federation endpoint:

`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 ***
```

Note

The following instructions in this step only work using GET requests.

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.
- Using the credentials for one role to assume a different role is called [role chaining \(p. 185\)](#). 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 \(p. 374\)](#), those applications are not subject to this limitation.

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 \(p. 382\)](#).

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%2FL%2FN%2BgYpan5QFz0XUpc8s7HYjRsgcsrsm%22%2C+%22sessionToken%2
2%3A%22FQoDYXdzEBQaDLbj3VWv2u50NN%2F3yyLSASwYtWhPnGPMNmzzFfZsL0Qd3vtYHw5A5dW
Aj0srkdPkghomIe3mJip5%2FdjDBbo7Sm0%2FENDEiCdp$QKodTpIeKA8xQq0CwFg6a69xdEBQT8
FipATnLbKoyS4b%2FebhnsTUjZZQWp0wXqFF7gSm%2FMe2tXe0jzsdP0012obe$9lijPSdF1k2b5
PfGhiuyAR9aD5%2BubM0pY86fKex1qsytjvyTbZ9nXe6DvxVDcnC0h0GETJ7XFkSFdH0v%2FYR25C
UAhJ3nXIkIbG7Ucv9c0EpCf%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 ***"}
```

Note

AWS supports POST requests here.

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 \(p. 235\)](#), 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 ***
```

Note

The following instructions in this step only work using GET API.

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=VCQgs5qZt3Q6fn8Tr5EXAMPLELnwB7JjUc-SHwnUUWabcRdnWsi4DBn-dvC
CZ85wrD0nmlUcZEXAMPLE-vXYH4Q_mleuF_W2BE5HYexbe9y40f-kje53SsjNNecATfjIzpW1
WibbnH6YcYRiBoffZBGExbEXAMPLE5aiKX4THWjQKC6gg6alHu6JFIn0JoK3dtP6I9a6hi6yPgm
i0kPZMmNGhsvVxetKzr8mx3pxhHbMEXAMPLETvipj0rok3IyCR2VvcIjqwfWv32HU2Xl471u
3fu6u0fUComeKiqTGX974xJ0ZbdmX_t_1LrhEXAMPLEDDIisSnyHg2xaZzqudm4mo2uTDk9Pv
915K0ZCqIgEXAMPLEcA6tgLPykEWGUyH6BdSC6166n4M4JkXIQgac7_7821YqixsNxZ6rsrpzwf
nQoS1407R0eJCCJ684EXAMPLEZRdBnNuLbUYpz2Iw3vIN0tQgOujwnwydPscM9F7foaEK3jwMkg
Apeb1-6L_0B12MZhuxFxx5555EXAMPLEhyETEd4Zu1KPdXHkg16T9ZkI1Hz2Uy1RUTUhUxNtSQ
nWC5xkbBoEcXqpoSiEKe7yhje9Vzhd61EXAMPLElbWeouACEMG6-Vd3dAgFYd6i5FYoyFrZLWvm
0LSG7RyYKeYN5VIzUk3YWQpyjP0RiT5KUrsUi-NEXAMPLExMOMdo0DBEgKQsk-iu2ozh6r8bxwC
RNhujg
```

Example code using Python

The following examples show how to use Python to programmatically construct a URL that gives federated users direct access to the AWS Management Console. There are two examples:

- Federate via GET requests to AWS
- Federate via POST requests to AWS

Both examples use the the [AWS SDK for Python \(Boto3\)](#) and [AssumeRole](#) API to obtain temporary security credentials.

Use GET Requests

```
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 Amazon 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: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)
```

Use POST Requests

```
import urllib, json, sys
import requests # 'pip install requests'
import boto3 # AWS SDK for Python (Boto3) 'pip install boto3'
import os
from selenium import webdriver # 'pip install selenium', 'brew install chromedriver'

# Step 1: Authenticate user in your own identity system.

# Step 2: Using the access keys for an IAM user in your AAWS 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 Amazon 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
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)

sts_connection = boto3.client('sts')

assumed_role_object = sts_connection.assume_role(
    RoleArn="arn:aws:iam::account-id:role/ROLE-NAME",
```

```

        RoleSessionName="AssumeRoleDemoSession",
    )

# 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 = {}
request_parameters['Action'] = 'getSigninToken'
request_parameters['SessionDuration'] = '43200'
request_parameters['Session'] = json_string_with_temp_credentials

request_url = "https://signin.aws.amazon.com/federation"
r = requests.post( request_url, data=request_parameters)

# Returns a JSON document with a single element named SigninToken.
signin_token = json.loads(r.text)

# Step 5: Create a POST request where users can use the sign-in token to sign in to
# the console. The POST request must be made within 15 minutes after the
# sign-in token was issued.
request_parameters = {}
request_parameters['Action'] = 'login'
request_parameters['Issuer']='Example.org'
request_parameters['Destination'] = 'https://console.aws.amazon.com/'
request_parameters['SigninToken'] = signin_token['SigninToken']

jsrequest = '''
var form = document.createElement('form');
form.method = 'POST';
form.action = '{request_url}';
request_parameters = {request_parameters}
for (var param in request_parameters) {{
    if (request_parameters.hasOwnProperty(param)) {{
        const hiddenField = document.createElement('input');
        hiddenField.type = 'hidden';
        hiddenField.name = param;
        hiddenField.value = request_parameters[param];
        form.appendChild(hiddenField);
    }}
}}
document.body.appendChild(form);
form.submit();
''.format(request_url=request_url, request_parameters=request_parameters)

driver = webdriver.Chrome()
driver.execute_script(jsrequest);

```

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.
*/
AWSCredentials 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 session 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 simplify the process of setting up a service because you don't have to manually add permissions for the service to complete actions on your behalf.

Note

Remember that service roles are different from service-linked roles. A service role is an [IAM role](#) that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see [Creating a role to delegate permissions to an AWS service](#) in the *IAM User Guide*. A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an

action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

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.

Before you can delete the roles, you must first delete 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 \(p. 1224\)](#) 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 \(p. 279\)](#).

```
{
    "Sid": "PolicyStatementToAllowUserToListRoles",
    "Effect": "Allow",
    "Action": ["iam>ListRoles"],
    "Resource": "*"
},
{
    "Sid": "PolicyStatementToAllowUserToPassOneSpecificRole",
    "Effect": "Allow",
    "Action": "iam:PassRole"
}
```

```
    "Action": [ "iam:PassRole" ],
    "Resource": "arn:aws:iam::account-id: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 originating service can use those permissions in the future. If another user or role has permission to perform actions in the originating service, the originating 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 have the same permissions to 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.

Important

When transferring service linked role permissions, be aware of the "confused deputy" privilege escalation risk. Learn more about mitigating that risk in [Cross-service confused deputy prevention](#).

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 \(p. 1197\)](#).

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 \(p. 1224\)](#) 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.
4. Choose the use case for your service. Use cases are defined by the service to include the trust policy required by the service. Then, choose **Next**.

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 check box next to the policy that assigns the permissions that you want the role to have, and then choose **Next**.

Note

The permissions that you specify are available to any entity that uses the role. By default, a role has no permissions.

6. 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 let you enter an optional suffix.

If possible, enter 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.

7. (Optional) For **Description**, edit the description for the new service-linked role.
8. You cannot attach tags to service-linked roles during creation. For more information about using tags in IAM, see [Tagging IAM resources \(p. 399\)](#).
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 \(p. 1224\)](#) 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. Enter 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 AWS CLI.

For information about which services support using service-linked roles, see [AWS services that work with IAM \(p. 1224\)](#) 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 AWS 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 \(p. 1224\)](#) 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**.
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 \(p. 248\)](#) 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 \(p. 1224\)](#). 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 know the name of the service-linked role that you want to delete, enter the following command to list the role 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. Enter the following command to submit a service-linked role deletion request:

```
aws iam delete-service-linked-role --role-name role-name
```

3. Enter 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 \(p. 248\)](#) 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 \(p. 1224\)](#). 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 \(p. 248\)](#) 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 \(p. 1224\)](#). 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 \(p. 251\)](#)
- [Creating a role to delegate permissions to an AWS service \(p. 255\)](#)
- [Creating a role for a third-party Identity Provider \(federation\) \(p. 260\)](#)
- [Creating a role using custom trust policies \(console\) \(p. 270\)](#)
- [Examples of policies for delegating access \(p. 271\)](#)

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 \(p. 191\)](#).

For information about how to use roles to delegate permissions, see [Roles terms and concepts \(p. 184\)](#). 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 \(p. 255\)](#).

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 high-level information about creating 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 \(p. 188\)](#).

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 **AWS account** role type.
4. To create a role for your account, choose **This account**. To create a role for another account, choose **Another AWS account** and enter the **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, 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 \(p. 191\)](#).

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 \(p. 114\)](#)
7. Choose **Next**.
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 [Creating IAM policies \(p. 582\)](#). 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 \(p. 501\)](#). 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**.
11. For **Role name**, enter a name for your role. Role names must be unique within your AWS account. When a role name is used in a policy or as part of an ARN, the role name is case sensitive. When a role name appears to customers in the console, such as during the sign-in process, the role name is case insensitive. Because various entities might reference the role, you can't edit the name of the role after it is created.
12. (Optional) For **Description**, enter a description for the new role.
13. Choose **Edit** in the **Step 1: Select trusted entities** or **Step 2: Add permissions** sections to edit the use cases and permissions for the role.
14. (Optional) Add metadata to the role by attaching tags as key-value pairs. For more information about using tags in IAM, see [Tagging IAM resources \(p. 399\)](#).
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 \(p. 277\)](#).

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 \(p. 501\)](#) 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 roles \(AWS CLI or AWS API\) \(p. 406\)](#).

4. (Optional) Set the [permissions boundary \(p. 501\)](#) 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 \(p. 114\)](#).

```
{  
    "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:policy/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 \(p. 277\)](#).

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 \(AWS CLI\) \(p. 285\)](#).

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 \(p. 501\)](#) 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 \(p. 277\)](#).

3. (Optional) Add custom attributes to the user by attaching tags: [TagRole](#)

For more information, see [Managing tags on IAM users \(AWS CLI or AWS API\) \(p. 404\)](#).

4. (Optional) Set the [permissions boundary \(p. 501\)](#) 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 \(AWS API\) \(p. 291\)](#).

Creating an IAM role (AWS CloudFormation)

For information about creating an IAM role in AWS CloudFormation, see the [resource and property reference](#) and [examples](#) in the *AWS CloudFormation User Guide*.

For more information about IAM templates in AWS CloudFormation, see [AWS Identity and Access Management template snippets](#) in the *AWS CloudFormation User Guide*.

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 \(p. 185\)](#). When a role serves a specialized purpose for a service, it is categorized as a [service role for EC2 instances \(p. 185\)](#) (for example), or a [service-linked role \(p. 185\)](#). 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 \(p. 1224\)](#). 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 \(p. 184\)](#).

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 following policies 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:ListRolePolicies",  
                "iam:ListRoles",  
                "iam:PutRolePolicy"  
            ]  
        }  
    ]  
}
```

```

        "iam:PutRolePolicy"
    ],
    "Resource": "arn:aws:iam::*:role/SERVICE-ROLE-NAME"
}
]
}
```

To allow an IAM entity to create any service role

AWS recommends that you allow only administrative users to create any service role. A person with permissions to create a role and attach any policy can escalate their own permissions. Instead, create a policy that allows them to create only the roles that they need or have an administrator create the service role on their behalf.

To attach a policy that allows an administrator to access your entire AWS account, use the [AdministratorAccess](#) AWS managed policy.

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": "*"
        }
    ]
}
```

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

AWS recommends that you allow only administrative users to delete any service role. Instead, create a policy that allows them to delete only the roles that they need or have an administrator delete the service role on their behalf.

To attach a policy that allows an administrator to access your entire AWS account, use the [AdministratorAccess](#) AWS managed policy.

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 trusted entity**, choose **AWS service**.
4. Choose the use case for your service. Use cases are defined by the service to include the trust policy required by the service. Then, choose **Next**.
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 [Creating IAM policies \(p. 582\)](#). 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.
 - Choose from a limited set of permissions.
 - Choose from any permissions.
 - Select no policies at this time, create the policies later, and then attach them to the role.
6. (Optional) Set a [permissions boundary \(p. 501\)](#). This is an advanced feature that is available for service roles, but not service-linked roles.

Expand the **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 [Creating IAM policies \(p. 582\)](#). 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**.
8. 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 other AWS resources might reference the role, you cannot edit the name of the role after it has been created.

9. (Optional) For **Description**, enter a description for the new role.
10. Choose **Edit** in the **Step 1: Select trusted entities** or **Step 2: Add permissions** sections to edit the use cases and permissions for the role.
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 resources \(p. 399\)](#).
12. 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 \(p. 501\)](#) for your role.

To create a role for an AWS service from the AWS CLI

1. The following [`create-role`](#) command creates a role named *Test-Role* and attaches a trust policy to it:


```
aws iam create-role --role-name Test-Role --assume-role-policy-document file://Test-Role-Trust-Policy.json
```

2. Attach a managed permissions policy to the role: [`aws iam attach-role-policy`](#).

For example, the following `attach-role-policy` command attaches the AWS managed policy named `ReadOnlyAccess` to the IAM role named `ReadOnlyRole`:

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/ReadonlyAccess --role-name ReadOnlyRole
```

or

Create an inline permissions policy for the role: [`aws iam put-role-policy`](#)

To add an inline permissions policy, see the following example:

```
aws iam put-role-policy --role-name Test-Role --policy-name ExamplePolicy --policy-document file://AdminPolicy.json
```

3. (Optional) Add custom attributes to the role by attaching tags: [`aws iam tag-role`](#)

For more information, see [Managing tags on IAM roles \(AWS CLI or AWS API\) \(p. 406\)](#).

4. (Optional) Set the [permissions boundary \(p. 501\)](#) 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 \(p. 381\)](#). 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 \(p. 501\)](#) 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 users \(AWS CLI or AWS API\) \(p. 404\)](#).

4. (Optional) Set the [permissions boundary \(p. 501\)](#) 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 \(p. 381\)](#). 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 \(p. 198\)](#).

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 \(console\) \(p. 262\)](#).
- For SAML 2.0, see [Creating a role for SAML 2.0 federation \(console\) \(p. 268\)](#).

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 \(p. 263\)](#).
- For a SAML provider, see [Prerequisites for creating a role for SAML \(p. 268\)](#).

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 \(p. 501\)](#) 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 roles \(AWS CLI or AWS API\) \(p. 406\)](#).
4. (Optional) Set the [permissions boundary \(p. 501\)](#) 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](#).

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"}  
    }  
}
```

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 \(p. 263\)](#).
- For a SAML provider, see [Prerequisites for creating a role for SAML \(p. 268\)](#).

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 users \(AWS CLI or AWS API\) \(p. 404\)](#).
4. (Optional) Set the [permissions boundary \(p. 501\)](#) 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 (OIDC) federated identity providers instead of creating AWS Identity and Access Management 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 IdPs, see [Identity providers and federation \(p. 198\)](#).

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 with one or more services offering federated OIDC identity. 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 is 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 as follows:
 - [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 you receive the required information from the IdP, create an IdP in IAM. For more information, see [Creating OpenID Connect \(OIDC\) identity providers \(p. 210\)](#).

Important

If you are using an OIDC IdP from Google, Facebook, or Amazon Cognito, don't create a separate IAM IdP in the AWS Management Console. These OIDC identity providers are already built into AWS and are available for you to use. Skip this step and create new roles using your IdP in the following step.

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 IdPs, 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 assigns 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 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 Amazon Resource Name (ARN) of the OIDC IdP 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. The identity pool ID should match the app ID that you received when you configured the app with the IdP. This match between the IDs ensures that the request comes from your app. Create a condition element similar to one of 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"}}
```

Note

The values for the principal in the trust policy for the role are specific to an IdP. A role for web identity or OIDC can specify only one principal. Therefore, if the mobile app allows users to sign in from more than one IdP, create a separate role for each IdP that you want to support. Create separate trust policies for each IdP.

If a user uses a mobile app to sign in from Login with Amazon, the following example trust policy would apply. In the example, `amzn1.application-oa2-123456` represents the app ID that Amazon assigns 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"}}
    }]
}

```

If a user uses a mobile app to sign in from Facebook, the following example trust policy would apply. In this example, **111222333444555** represents the app ID that Facebook assigns.

```

{
    "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"}}
    }]
}

```

If a user uses a mobile app to sign in from Google, the following example trust policy would apply. In this example, **666777888999000** represents the app ID that Google assigns.

```

{
    "Version": "2012-10-17",
    "Statement": [{
        "Sid": "RoleForGoogle",
        "Effect": "Allow",
        "Principal": {"Federated": "accounts.google.com"},
        "Action": "sts:AssumeRoleWithWebIdentity",
        "Condition": {"StringEquals": {"accounts.google.com:aud": "666777888999000"}}
    }]
}

```

If a user uses a mobile app to sign in from Amazon Cognito, the following example trust policy would apply. In this example, **us-east:12345678-ffff-ffff-ffff-123456** represents the identity pool ID that Amazon Cognito assigns.

```

{
    "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 or 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 \(federation\) \(p. 260\)](#).

Important

If you use Amazon Cognito, 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 IdP for your role:
 - If you want to create a role for an individual web IdP, choose **Login with Amazon, Facebook, or Google**.

Note

You must create a separate role for each IdP that you want to support.

- If you want to create an advanced scenario role for Amazon Cognito, choose **Amazon Cognito**.

Note

You must manually create a role to use with Amazon Cognito only when you work on an advanced scenario. Otherwise, Amazon Cognito can create roles for you. For more information about Amazon Cognito, see [Identity pools \(federated identities\) external identity providers](#) in the *Amazon Cognito Developer Guide*.

5. Enter the identifier for your application. The label of the identifier changes based on the provider you choose:
 - If you want to create a role for Login with Amazon, enter the app ID into the **Application ID** box.
 - If you want to create a role for Facebook, enter the app ID into the **Application ID** box.
 - If you want to create a role for Google, enter the audience name into the **Audience** box.
 - If you want to create a role for Amazon Cognito, enter the ID of the identity pool that you have created for your Amazon Cognito applications into the **Identity Pool ID** box.
6. (Optional) Choose **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**.
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 [Creating IAM policies \(p. 582\)](#). 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 \(p. 501\)](#). This is an advanced feature.

Open the **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**.
11. For **Role name**, enter a role name. Role names must be unique within your AWS account. They are not case dependent. For example, you can't create roles named both **PRODROLE** and **prodrole**. Because other AWS resources might reference the role, you can't edit the name of the role after you create it.
12. (Optional) For **Description**, enter a description for the new role.

13. To edit the use cases and permissions for the role, choose **Edit** in the **Step 1: Select trusted entities** or **Step 2: Add permissions** sections.
14. (Optional) To add metadata to the role, attach tags as key–value pairs. For more information about using tags in IAM, see [Tagging IAM resources \(p. 399\)](#).
15. Review the role and then choose **Create role**.

Configuring a role for GitHub OIDC identity provider

If you use GitHub as an OIDC IdP, best practice is to limit the entities that can assume the role associated with the IAM IdP. When you include a condition statement in the trust policy, you can limit the role to a specific GitHub organization, repository, or branch. For information about limiting roles, see [GitHub Docs - About security hardening with OpenID Connect](#). You can use the condition key `token.actions.githubusercontent.com:sub` to limit access. We recommend that you limit the condition to a specific set of repositories or branches. If you do not include this condition, then GitHub Actions from organizations or repositories outside of your control are able to assume roles associated with the GitHub IAM IdP in your AWS account. For information about how to configure AWS to trust GitHub's OIDC as a federated identity, see [GitHub Docs - Configuring OpenID Connect in Amazon Web Services](#).

The following example trust policy limits access to the defined GitHub organization, repository, and branch.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": [
        "Federated": "arn:aws:iam::012345678910:oidc-provider/token.actions.githubusercontent.com"
      ],
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "token.actions.githubusercontent.com:aud": "sts.amazonaws.com",
          "token.actions.githubusercontent.com:sub": "repo:GitHubOrg/GitHubRepo:ref:refs/heads/GitHubBranch"
        }
      }
    }
  ]
}
```

The following example condition limits access to the defined GitHub organization and repository, but grants access to any branch within the repository.

```
"Condition": {
  "StringEquals": {
    "token.actions.githubusercontent.com:aud": "sts.amazonaws.com"
  },
  "StringLike": {
    "token.actions.githubusercontent.com:sub": "repo:GitHubOrg/GitHubRepo:"*
  }
}
```

The following example condition limits access to any repository or branch within the defined GitHub organization. We recommend this as a minimum trust policy condition for roles that use GitHub as an OIDC IdP.

```

"Condition": {
    "StringEquals": {
        "token.actions.githubusercontent.com:aud": "sts.amazonaws.com"
    },
    "StringLike": {
        "token.actions.githubusercontent.com:sub": "repo:GitHubOrg/*"
    }
}

```

For more information about the web identity federation keys available for condition checks in policies, see [Available keys for AWS web identity federation \(p. 1372\)](#).

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 \(p. 198\)](#).

Note

To improve federation resiliency, we recommend that you configure your IdP and AWS federation to support multiple SAML sign-in endpoints. For details, see the AWS Security Blog article [How to use regional SAML endpoints for failover](#).

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 \(p. 218\)](#).
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 to ensure that only your application can assume the role:

- 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 \(p. 205\)](#) 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: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 **Select a 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 AWS Management 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, enter 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 **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**.
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 [Creating IAM policies \(p. 582\)](#). 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 \(p. 501\)](#). This is an advanced feature.

Open the **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**.
12. Choose **Next: Review**.
13. For **Role name**, enter 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 **Description**, enter a description for the new role.
15. Choose **Edit** in the **Step 1: Select trusted entities** or **Step 2: Add permissions** sections to edit the use cases and permissions for the role.
16. (Optional) Add metadata to the role by attaching tags as key-value pairs. For more information about using tags in IAM, see [Tagging IAM resources \(p. 399\)](#).
17. 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 \(p. 222\)](#).

Creating a role using custom trust policies (console)

You can create a custom trust policy to delegate access and allow others to perform actions in your AWS account. For more information, see [Creating IAM policies \(p. 582\)](#).

For information about how to use roles to delegate permissions, see [Roles terms and concepts \(p. 184\)](#).

Creating an IAM role using a custom trust policy (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 high-level information about creating 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 \(p. 188\)](#).

To create a role using a custom 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 console, choose **Roles** and then choose **Create role**.
3. Choose the **Custom trust policy** role type.
4. In the **Custom trust policy** section, enter or paste the custom trust policy for the role. For more information, see [Creating IAM policies \(p. 582\)](#).
5. Resolve any security warnings, errors, or general warnings generated during [policy validation \(p. 588\)](#), and then choose **Next**.
6. Select the check box next to the custom trust policy you created.
7. (Optional) Set a [permissions boundary \(p. 501\)](#). This is an advanced feature that is available for service roles, but not service-linked roles.

Open the **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.

8. Choose **Next**.

9. 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. 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.

10. (Optional) For **Description**, enter a description for the new role.
11. Choose **Edit** in the **Step 1: Select trusted entities** or **Step 2: Add permissions** sections to edit the custom policy and permissions for the role.
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 resources \(p. 399\)](#).
13. Review the role and then choose **Create role**.

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 using the JSON editor" \(p. 583\)](#).

Topics

- [Using roles to delegate access to the resources of another AWS account resources \(p. 271\)](#)
- [Using a policy to delegate access to services \(p. 271\)](#)
- [Using a resource-based policy to delegate access to an Amazon S3 bucket in another account \(p. 272\)](#)
- [Using a resource-based policy to delegate access to an Amazon SQS queue in another account \(p. 273\)](#)
- [Cannot delegate access when the account is denied access \(p. 274\)](#)

Using roles to delegate access to the resources of another AWS account 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 \(p. 40\)](#).

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 User 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 \(p. 1306\)](#).)

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 a 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 a groups or to the user 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. When using AssumeRole* API operations, the IAM role that you assume is the resource. The user or role that calls AssumeRole* API operations is the principal.

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	User (by switching roles (p. 282))	Maximum session duration on the Role Summary page	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 CLI or AssumeRole API operation	User or role ¹	duration-seconds CLI or DurationSeconds API parameter	15m Maximum session duration setting ² 1hr
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. 233) constructed with AssumeRole	User or role	SessionDuration HTML parameter in the URL	15m 12hr 1hr
Console URL (p. 233) constructed with AssumeRoleWithSAML	Any user authenticated using SAML	SessionDuration HTML parameter in the URL	15m 12hr 1hr
Console URL (p. 233) 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 \(p. 185\)](#). 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 \(p. 374\)](#), 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 \(p. 384\)](#). 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 their user 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 \(p. 276\)](#) later in this page.

Notes

- The maximum session duration setting does not limit sessions that are assumed by AWS services.
- To allow users to assume the current role again within a role session, specify the role ARN or AWS account ARN as a principal in the role trust policy. AWS services that provide compute

resources such as Amazon EC2, Amazon ECS, Amazon EKS, and Lambda provide temporary credentials and automatically rotate these credentials. This ensures that you always have a valid set of credentials. For these services, it's not necessary to assume the current role again to obtain temporary credentials. However, if you intend to pass [session tags \(p. 417\)](#) or a [session policy \(p. 487\)](#), you need to assume the current role again. To learn how to modify a role trust policy to add the principal role ARN or AWS account ARN, see [Modifying a role trust policy \(console\) \(p. 385\)](#).

Topics

- [View the maximum session duration setting for a role \(p. 276\)](#)
- [Granting a user permissions to switch roles \(p. 277\)](#)
- [Granting a user permissions to pass a role to an AWS service \(p. 279\)](#)
- [Switching to a role \(console\) \(p. 282\)](#)
- [Switching to an IAM role \(AWS CLI\) \(p. 285\)](#)
- [Switching to an IAM role \(Tools for Windows PowerShell\) \(p. 289\)](#)
- [Switching to an IAM role \(AWS API\) \(p. 291\)](#)
- [Using an IAM role to grant permissions to applications running on Amazon EC2 instances \(p. 374\)](#)
- [Revoking IAM role temporary security credentials \(p. 382\)](#)

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 \(p. 251\)](#), they establish trust between the account that owns the role, 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 *potentially* allows 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 permission to switch to a role

1. As the administrator of the trusted account, create a new policy for the user, or edit an existing policy to add the required elements. For details, see [Creating or editing the policy \(p. 278\)](#).
2. Then, choose how you want to share the role information:
 - **Role link:** Send users a link that takes them to the **Switch Role** page with all the details already filled in.
 - **Account ID or alias:** Provide each user with the role name along with the account ID number or account alias. The user then goes to the **Switch Role** page and adds the details manually.

For details, see [Providing information to the user \(p. 278\)](#).

Note that you can switch roles only when you sign in as an IAM user, a SAML-federated role, or a web-identity federated role. 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 \(p. 191\)](#) 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 you are ultimately granting permissions to a user to accomplish a task. However, we don't recommend that you grant permissions directly to an individual user. When a user assumes a role, they are assigned the permissions associated with that role.
- 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 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. The credentials for RoleA are not used for this action.

Topics

- [Creating or editing the policy \(p. 278\)](#)
- [Providing information to the user \(p. 278\)](#)

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

Users that get the policy are allowed to switch roles on the resource listed (either through group membership or directly attached).

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: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 streamline access for your users by sending them a link that is preconfigured with the account ID and role name. You can see the role link after completing the **Create Role** wizard by selecting the **View Role** banner, or on 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 \(console\) \(p. 282\)](#) to walk them through the process. To troubleshoot common issues that you might encounter when you assume a role, see [I can't assume a role \(p. 1196\)](#).

Considerations

- If you create the role programmatically, you can create the role with a path and 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 and 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 \(p. 473\)](#) to learn who performed an action in AWS. You can use the sts:SourceIdentity condition key in the role trust policy to require users to specify an identity when they assume a role. For example, you can require that IAM users specify their own user name as their source identity. This can help you determine which user performed a specific action in AWS. For more information, see [sts:SourceIdentity \(p. 1382\)](#). You can also use [sts:RoleSessionName \(p. 1381\)](#) to require users to specify a session name when they assume a role. This can help you differentiate between role sessions when a role is used by different principals.

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 assume the role later and perform actions on your behalf. For most services, you only have to pass the role to the service once during setup, 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 Amazon 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.

Warning

- You cannot use the PassRole permission to pass a cross-account role.
- Do not try to control who can pass a role by tagging the role and then using the ResourceTag condition key in a policy with the iam:PassRole action. This approach does not have reliable results.

When you specify 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 when you create an Auto Scaling group for the first time. If you try to specify the service-linked role when you create an Auto Scaling group and you don't have the iam:PassRole permission, you receive an error. If you don't explicitly specify the role, the iam:PassRole permission is not required, and the default is to use AWSServiceRoleForAutoScaling role for all operations that are performed on that group. To learn which services support service-linked roles, see [AWS services that work with IAM \(p. 1224\)](#). 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 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 \(p. 1278\)](#). 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 \(p. 1371\)](#).

Example 1

Suppose 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 an 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 approved roles. You usually add `iam:GetRole` to `iam:PassRole` so 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 beginning 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 for **Use cases for other AWS services**, choose the **RDS** service. Choose **RDS – Enhanced Monitoring**, and then choose **Next**.
4. Choose the **AmazonRDSEnhancedMonitoringRole** permissions policy.
5. Choose **Next**.
6. For **Role name**, enter a role name that helps you identify the purpose of this role. Role names must be unique within your AWS account. When a role name is used in a policy or as part of an ARN, the role name is case sensitive. When a role name appears to customers in the console, such as during the sign-in process, the role name is case insensitive. Because various entities might reference the role, you can't edit the name of the role after it is created.
7. (Optional) For **Description**, enter a description for the new role.
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 resources \(p. 399\)](#).
9. 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 access Enhanced Monitoring needs a policy that includes a statement that allows the user to list the RDS roles and 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 6.

```
{  
  "Sid": "PolicyStatementToAllowUserToListRoles",  
  "Effect": "Allow",  
  "Action": ["iam>ListRoles"],  
  "Resource": "*"  
},  
{  
  "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, as follows.

```
"Resource": "arn:aws:iam::account-id:role/RDS-*"
```

iam:PassRole actions in AWS CloudTrail logs

PassRole is not an API call. PassRole is a permission, meaning no CloudTrail logs are generated for IAM PassRole. To review what roles are passed to which AWS services in CloudTrail, you must review the CloudTrail log that created or modified the AWS resource receiving the role. For example, a role is passed to an AWS Lambda function when it's created. The log for the CreateFunction action shows a record of role that was passed to the function.

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 \(p. 183\)](#), and [Creating IAM roles \(p. 250\)](#).

Important

The permissions of your 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, a user in IAM Identity Center, 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.

Notes:

- 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, a user in IAM Identity Center, a SAML-federated role, or a web-identity federated role.
 - You cannot switch roles in the AWS Management Console to a role that requires an [ExternalId \(p. 191\)](#) value. You can switch to such a role only by calling the [AssumeRole](#) API that supports the ExternalId parameter.
-
- 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 [User Types](#) in the [AWS Sign-In User Guide](#).
- ***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: arn:aws:iam::123456789012: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 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 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 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 \(p. 1196\)](#).

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.

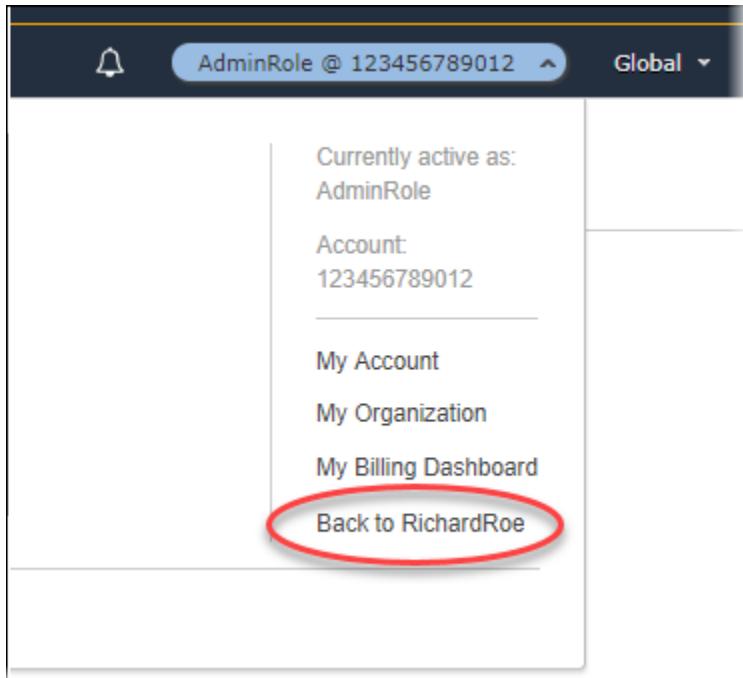
Tip

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 \(p. 183\)](#), and [Creating IAM roles \(p. 250\)](#). To learn about the different methods that you can use to assume a role, see [Using IAM roles \(p. 274\)](#).

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 \(p. 198\)](#) ([SAML \(p. 205\)](#) or [OIDC \(p. 199\)](#)) 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 cannot assume a role when you are signed in as the AWS account root user.

[Role chaining \(p. 185\)](#) – You can also use role chaining, which is using permissions from a role to access a second role.

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. If you switch roles in the console, your session duration is limited to maximum of one hour. To learn how to view the maximum value for your role, see [View the maximum session duration setting for a role \(p. 276\)](#).

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 \(p. 473\)](#) to learn who performed an action in AWS. Your administrator might require that you specify a source identity or a role session name when you assume the role. For more information, see [sts:SourceIdentity \(p. 1382\)](#) and [sts:RoleSessionName \(p. 1381\)](#).

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 `efgh` 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:GetBucketLocation",
                "s3:GetAccountPublicAccessBlock",
                "s3>ListAccessPoints",
                "s3>ListAllMyBuckets"
            ],
            "Resource": "arn:aws:s3:::*"
        },
        {
            "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 `efgh` cross-account role allows read-only Amazon S3 tasks on the `my-bucket-2` bucket within the same 222222222222 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:GetBucketLocation",
                "s3:GetAccountPublicAccessBlock",
                "s3>ListAccessPoints",
                "s3>ListAllMyBuckets"
            ],
            "Resource": "arn:aws:s3:::*"
        },
        {
            "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 differentiate between role sessions when a role is used by different principals 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-*nnnnnnnnn*. Here *nnnnnnnnn* 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 \(p. 183\)](#), and [Creating IAM roles \(p. 250\)](#).

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-STSRole` 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 is in effect at a time.

Note

For security purposes, administrators can [review AWS CloudTrail logs \(p. 473\)](#) to learn who performed an action in AWS. Your administrator might require that you specify a

source identity or a role session name when you assume the role. For more information, see [sts:SourceIdentity \(p. 1382\)](#) and [sts:RoleSessionName \(p. 1381\)](#).

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-STSRole -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/uZEXAMPLEcihzFB51TYLto9dyBgSDyEXAMPLE9/
g7QRUhZp4bqbEXAMPLEnwGPy
Oj59pFA41NKCikVgkREXAMPLEjlzxQ7y52gekeVEXAMPLEDiB9ST3UuysgsKdEXAMPLE1TVastU1A0SKFEXAMPLEiywCC/
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 *-Credential* 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)

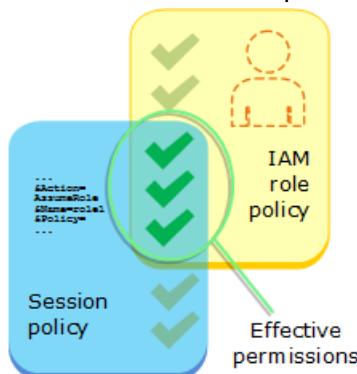
A *role* specifies a set of permissions that you can use to access AWS resources. In that sense, it is similar to an [IAM user](#). A principal (person or application) assumes a role to receive temporary permissions to carry out required tasks and interact with AWS resources. The role can be in your own account or any other AWS account. For more information about roles, their benefits, and how to create and configure them, see [IAM roles \(p. 183\)](#), and [Creating IAM roles \(p. 250\)](#). To learn about the different methods that you can use to assume a role, see [Using IAM roles \(p. 274\)](#).

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 \(p. 487\)](#). 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 \(p. 1306\)](#).



You can call `AssumeRole` when you are signed in as an IAM user, or as an [externally authenticated user \(p. 198\)](#) ([SAML \(p. 205\)](#) or [OIDC \(p. 199\)](#)) already using a role. You can also use [role chaining \(p. 185\)](#), 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 \(p. 276\)](#).

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 \(p. 473\)](#) to learn who performed an action in AWS. Your administrator might require that you specify a source identity or a role session name when you assume the role. For more information, see [sts:SourceIdentity \(p. 1382\)](#) and [sts:RoleSessionName \(p. 1381\)](#).

The following code examples show how to create a user and assume a role.

Warning

To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as [AWS IAM Identity Center \(successor to AWS Single Sign-On\)](#).

- Create a user with no permissions.
- Create a role that grants permission to list Amazon S3 buckets for the account.
- Add a policy to let the user assume the role.
- Assume the role and list S3 buckets using temporary credentials, then clean up resources.

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
global using Amazon;
global using Amazon.IdentityManagement;
global using Amazon.S3;
global using Amazon.S3.Model;
global using Amazon.SecurityToken;
global using Amazon.SecurityToken.Model;
global using IAMActions;
global using IamScenariosCommon;
global using Microsoft.Extensions.DependencyInjection;
global using Microsoft.Extensions.Hosting;
global using Microsoft.Extensions.Logging;
global using Microsoft.Extensions.Logging.Console;
global using Microsoft.Extensions.Logging.Debug;

namespace IAMActions;

public class IAMWrapper
{
    private readonly IAmazonIdentityManagementService _IAMService;

    /// <summary>
    /// Constructor for the IAMWrapper class.
    /// </summary>
    /// <param name="IAMService">An IAM client object.</param>
    public IAMWrapper(IAmazonIdentityManagementService IAMService)
    {
        _IAMService = IAMService;
    }

    /// <summary>
    /// Add an existing IAM user to an existing IAM group.
    /// </summary>
```

```

    ///> </summary>
    ///> <param name="userName">The username of the user to add.</param>
    ///> <param name="groupName">The name of the group to add the user to.</param>
    ///> <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> AddUserToGroupAsync(string userName, string groupName)
    {
        var response = await _IAMService.AddUserToGroupAsync(new
AddUserToGroupRequest
        {
            GroupName = groupName,
            UserName = userName,
        });

        return response.HttpStatusCode == HttpStatusCode.OK;
    }

    ///> <summary>
    ///> Attach an IAM policy to a role.
    ///> </summary>
    ///> <param name="policyArn">The policy to attach.</param>
    ///> <param name="roleName">The role that the policy will be attached to.</param>
    ///> <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> AttachRolePolicyAsync(string policyArn, string
roleName)
    {
        var response = await _IAMService.AttachRolePolicyAsync(new
AttachRolePolicyRequest
        {
            PolicyArn = policyArn,
            RoleName = roleName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    ///> <summary>
    ///> Create an IAM access key for a user.
    ///> </summary>
    ///> <param name="userName">The username for which to create the IAM access
    ///> key.</param>
    ///> <returns>The AccessKey.</returns>
    public async Task<AccessKey> CreateAccessKeyAsync(string userName)
    {
        var response = await _IAMService.CreateAccessKeyAsync(new
CreateAccessKeyRequest
        {
            UserName = userName,
        });

        return response.AccessKey;
    }

    ///> <summary>
    ///> Create an IAM group.
    ///> </summary>
    ///> <param name="groupName">The name to give the IAM group.</param>
    ///> <returns>The IAM group that was created.</returns>
    public async Task<Group> CreateGroupAsync(string groupName)
    {
        var response = await _IAMService.CreateGroupAsync(new CreateGroupRequest
{ GroupName = groupName });
    }
}

```

```

        return response.Group;
    }

    ///<summary>
    ///<param name="policyName">The name to give the new IAM policy.</param>
    ///<param name="policyDocument">The policy document for the new policy.</param>
    public async Task<ManagedPolicy> CreatePolicyAsync(string policyName, string
    policyDocument)
    {
        var response = await _IAMService.CreatePolicyAsync(new CreatePolicyRequest
        {
            PolicyDocument = policyDocument,
            PolicyName = policyName,
        });

        return response.Policy;
    }

    ///<summary>
    ///<param name="roleName">The name of the IAM role.</param>
    ///<param name="rolePolicyDocument">The name of the IAM policy document
    ///for the new role.</param>
    ///<returns>The Amazon Resource Name (ARN) of the role.</returns>
    public async Task<string> CreateRoleAsync(string roleName, string
    rolePolicyDocument)
    {
        var request = new CreateRoleRequest
        {
            RoleName = roleName,
            AssumeRolePolicyDocument = rolePolicyDocument,
        };

        var response = await _IAMService.CreateRoleAsync(request);
        return response.Role.Arn;
    }

    ///<summary>
    ///<param name="serviceName">The name of the AWS Service.</param>
    ///<param name="description">A description of the IAM service-linked role.</param>
    public async Task<Role> CreateServiceLinkedRoleAsync(string serviceName, string
    description)
    {
        var request = new CreateServiceLinkedRoleRequest
        {
            AWSServiceName = serviceName,
            Description = description
        };

        var response = await _IAMService.CreateServiceLinkedRoleAsync(request);
        return response.Role;
    }
}

```

```

    ///<summary>
    /// Create an IAM user.
    /// </summary>
    ///<param name="userName">The username for the new IAM user.</param>
    ///<returns>The IAM user that was created.</returns>
    public async Task<User> CreateUserAsync(string userName)
    {
        var response = await _IAMService.CreateUserAsync(new CreateUserRequest
{ UserName = userName });
        return response.User;
    }

    ///<summary>
    /// Delete an IAM user's access key.
    /// </summary>
    ///<param name="accessKeyId">The Id for the IAM access key.</param>
    ///<param name="userName">The username of the user that owns the IAM
    /// access key.</param>
    ///<returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteAccessKeyAsync(string accessKeyId, string
userName)
    {
        var response = await _IAMService.DeleteAccessKeyAsync(new
DeleteAccessKeyRequest
{
    AccessKeyId = accessKeyId,
    UserName = userName,
});
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    ///<summary>
    /// Delete an IAM group.
    /// </summary>
    ///<param name="groupName">The name of the IAM group to delete.</param>
    ///<returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteGroupAsync(string groupName)
    {
        var response = await _IAMService.DeleteGroupAsync(new DeleteGroupRequest
{ GroupName = groupName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }

    ///<summary>
    /// Delete an IAM policy associated with an IAM group.
    /// </summary>
    ///<param name="groupName">The name of the IAM group associated with the
    /// policy.</param>
    ///<param name="policyName">The name of the policy to delete.</param>
    ///<returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteGroupPolicyAsync(string groupName, string
policyName)
    {
        var request = new DeleteGroupPolicyRequest()
{
    GroupName = groupName,
    PolicyName = policyName,
};
        var response = await _IAMService.DeleteGroupPolicyAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
}

```

```

    ///<summary>
    /// Delete an IAM policy.
    ///</summary>
    ///<param name="policyArn">The Amazon Resource Name (ARN) of the policy to
    /// delete.</param>
    ///<returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeletePolicyAsync(string policyArn)
    {
        var response = await _IAMService.DeletePolicyAsync(new DeletePolicyRequest
        { PolicyArn = policyArn });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    ///<summary>
    /// Delete an IAM role.
    ///</summary>
    ///<param name="roleName">The name of the IAM role to delete.</param>
    ///<returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteRoleAsync(string roleName)
    {
        var response = await _IAMService.DeleteRoleAsync(new DeleteRoleRequest
        { RoleName = roleName });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    ///<summary>
    /// Delete an IAM role policy.
    ///</summary>
    ///<param name="roleName">The name of the IAM role.</param>
    ///<param name="policyName">The name of the IAM role policy to delete.</param>
    ///<returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteRolePolicyAsync(string roleName, string
    policyName)
    {
        var response = await _IAMService.DeleteRolePolicyAsync(new
DeleteRolePolicyRequest
        {
            PolicyName = policyName,
            RoleName = roleName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    ///<summary>
    /// Delete an IAM user.
    ///</summary>
    ///<param name="userName">The username of the IAM user to delete.</param>
    ///<returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteUserAsync(string userName)
    {
        var response = await _IAMService.DeleteUserAsync(new DeleteUserRequest
        { UserName = userName });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    ///<summary>
    /// Delete an IAM user policy.
    ///</summary>

```

```

    ///> <param name="policyName">The name of the IAM policy to delete.</param>
    ///> <param name="userName">The username of the IAM user.</param>
    ///> <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteUserPolicyAsync(string policyName, string
    userName)
    {
        var response = await _IAMService.DeleteUserPolicyAsync(new
        DeleteUserPolicyRequest { PolicyName = policyName, UserName = userName });

        return response.HttpStatusCode == System.Net HttpStatusCode.OK;
    }

    ///> <summary>
    ///> Detach an IAM policy from an IAM role.
    ///> </summary>
    ///> <param name="policyArn">The Amazon Resource Name (ARN) of the IAM policy.</
    param>
    ///> <param name="roleName">The name of the IAM role.</param>
    ///> <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DetachRolePolicyAsync(string policyArn, string
    roleName)
    {
        var response = await _IAMService.DetachRolePolicyAsync(new
        DetachRolePolicyRequest
        {
            PolicyArn = policyArn,
            RoleName = roleName,
        });

        return response.HttpStatusCode == System.Net HttpStatusCode.OK;
    }

    ///> <summary>
    ///> Gets the IAM password policy for an AWS account.
    ///> </summary>
    ///> <returns>The PasswordPolicy for the AWS account.</returns>
    public async Task<PasswordPolicy> GetAccountPasswordPolicyAsync()
    {
        var response = await _IAMService.GetAccountPasswordPolicyAsync(new
        GetAccountPasswordPolicyRequest());
        return response.PasswordPolicy;
    }

    ///> <summary>
    ///> Get information about an IAM policy.
    ///> </summary>
    ///> <param name="policyArn">The IAM policy to retrieve information for.</param>
    ///> <returns>The IAM policy.</returns>
    public async Task<ManagedPolicy> GetPolicyAsync(string policyArn)
    {

        var response = await _IAMService.GetPolicyAsync(new GetPolicyRequest
        { PolicyArn = policyArn });
        return response.Policy;
    }

    ///> <summary>
    ///> Get information about an IAM role.
    ///> </summary>
    ///> <param name="roleName">The name of the IAM role to retrieve information
    ///> for.</param>
    ///> <returns>The IAM role that was retrieved.</returns>

```

```

public async Task<Role> GetRoleAsync(string roleName)
{
    var response = await _IAMService.GetRoleAsync(new GetRoleRequest
    {
        RoleName = roleName,
    });

    return response.Role;
}

/// <summary>
/// Get information about an IAM user.
/// </summary>
/// <param name="userName">The username of the user.</param>
/// <returns>An IAM user object.</returns>
public async Task<User> GetUserAsync(string userName)
{
    var response = await _IAMService.GetUserAsync(new GetUserRequest { UserName
= userName });
    return response.User;
}

/// <summary>
/// List the IAM role policies that are attached to an IAM role.
/// </summary>
/// <param name="roleName">The IAM role to list IAM policies for.</param>
/// <returns>A list of the IAM policies attached to the IAM role.</returns>
public async Task<List<AttachedPolicyType>>
ListAttachedRolePoliciesAsync(string roleName)
{
    var attachedPolicies = new List<AttachedPolicyType>();
    var attachedRolePoliciesPaginator =
_IAMService.Paginator.ListAttachedRolePolicies(new
ListAttachedRolePoliciesRequest { RoleName = roleName });

    await foreach (var response in attachedRolePoliciesPaginator.Responses)
    {
        attachedPolicies.AddRange(response.AttachedPolicies);
    }

    return attachedPolicies;
}

/// <summary>
/// List IAM groups.
/// </summary>
/// <returns>A list of IAM groups.</returns>
public async Task<List<Group>> ListGroupsAsync()
{
    var groupsPaginator = _IAMService.Paginator.ListGroups(new
ListGroupsRequest());
    var groups = new List<Group>();

    await foreach (var response in groupsPaginator.Responses)
    {
        groups.AddRange(response.Groups);
    }

    return groups;
}

/// <summary>

```

```

    ///> List IAM policies.
    ///</summary>
    ///<returns>A list of the IAM policies.</returns>
    public async Task<List<ManagedPolicy>> ListPoliciesAsync()
    {
        var listPoliciesPaginator = _IAMService.Paginator.ListPolicies(new
ListPoliciesRequest());
        var policies = new List<ManagedPolicy>();

        await foreach (var response in listPoliciesPaginator.Responses)
        {
            policies.AddRange(response.Policies);
        }

        return policies;
    }

    ///<summary>
    ///> List IAM role policies.
    ///</summary>
    ///<param name="roleName">The IAM role for which to list IAM policies.</param>
    ///<returns>A list of IAM policy names.</returns>
    public async Task<List<string>> ListRolePoliciesAsync(string roleName)
    {
        var listRolePoliciesPaginator = _IAMService.Paginator.ListRolePolicies(new
ListRolePoliciesRequest { RoleName = roleName });
        var policyNames = new List<string>();

        await foreach (var response in listRolePoliciesPaginator.Responses)
        {
            policyNames.AddRange(response.PolicyNames);
        }

        return policyNames;
    }

    ///<summary>
    ///> List IAM roles.
    ///</summary>
    ///<returns>A list of IAM roles.</returns>
    public async Task<List<Role>> ListRolesAsync()
    {
        var listRolesPaginator = _IAMService.Paginator.ListRoles(new
ListRolesRequest());
        var roles = new List<Role>();

        await foreach (var response in listRolesPaginator.Responses)
        {
            roles.AddRange(response.Roles);
        }

        return roles;
    }

    ///<summary>
    ///> List SAML authentication providers.
    ///</summary>
    ///<returns>A list of SAML providers.</returns>
    public async Task<List<SAMLProviderListEntry>> ListSAMLProvidersAsync()
    {
        var response = await _IAMService.ListSAMLProvidersAsync(new
ListSAMLProvidersRequest());
        return response.SAMLProviderList;
    }

```

```

}

    ///<summary>
    ///<summary>
    ///</summary>
    ///<returns>A list of IAM users.</returns>
    public async Task<List<User>> ListUsersAsync()
    {
        var listUsersPaginator = _IAMService.Paginator.ListUsers(new
ListUsersRequest());
        var users = new List<User>();

        await foreach (var response in listUsersPaginator.Responses)
        {
            users.AddRange(response.Users);
        }

        return users;
    }

    ///<summary>
    ///<summary>
    ///</summary>
    ///<param name="userName">The username of the user to remove.</param>
    ///<param name="groupName">The name of the IAM group to remove the user
from.</param>
    ///<returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> RemoveUserFromGroupAsync(string userName, string
groupName)
    {
        // Remove the user from the group.
        var removeUserRequest = new RemoveUserFromGroupRequest()
        {
            UserName = userName,
            GroupName = groupName,
        };

        var response = await
_IAMService.RemoveUserFromGroupAsync(removeUserRequest);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }

    ///<summary>
    ///<summary>
    ///</summary>
    ///<param name="groupName">The name of the IAM group.</param>
    ///<param name="policyName">The name of the IAM policy.</param>
    ///<param name="policyDocument">The policy document defining the IAM policy.</
param>
    ///<returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> PutGroupPolicyAsync(string groupName, string
policyName, string policyDocument)
    {
        var request = new PutGroupPolicyRequest
        {
            GroupName = groupName,
            PolicyName = policyName,
            PolicyDocument = policyDocument
        };

        var response = await _IAMService.PutGroupPolicyAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
}

```

```

    ///<summary>
    /// Update the inline policy document embedded in a role.
    ///</summary>
    ///<param name="policyName">The name of the policy to embed.</param>
    ///<param name="roleName">The name of the role to update.</param>
    ///<param name="policyDocument">The policy document that defines the role.</param>
    ///<returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> PutRolePolicyAsync(string policyName, string roleName,
    string policyDocument)
    {
        var request = new PutRolePolicyRequest
        {
            PolicyName = policyName,
            RoleName = roleName,
            PolicyDocument = policyDocument
        };

        var response = await _IAMService.PutRolePolicyAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }

    ///<summary>
    /// Add or update an inline policy document that is embedded in an IAM user.
    ///</summary>
    ///<param name="userName">The name of the IAM user.</param>
    ///<param name="policyName">The name of the IAM policy.</param>
    ///<param name="policyDocument">The policy document defining the IAM policy.</param>
    ///<returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> PutUserPolicyAsync(string userName, string policyName,
    string policyDocument)
    {
        var request = new PutUserPolicyRequest
        {
            UserName = userName,
            PolicyName = policyName,
            PolicyDocument = policyDocument
        };

        var response = await _IAMService.PutUserPolicyAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    ///<summary>
    /// Wait for a new access key to be ready to use.
    ///</summary>
    ///<param name="accessKeyId">The Id of the access key.</param>
    ///<returns>A boolean value indicating the success of the action.</returns>
    public async Task<bool> WaitUntilAccessKeyIsReady(string accessKeyId)
    {
        var keyReady = false;

        do
        {
            try
            {
                var response = await _IAMService.GetAccessKeyLastUsedAsync(
                    new GetAccessKeyLastUsedRequest { AccessKeyId = accessKeyId });
                if (response.UserName is not null)
                {
                    keyReady = true;
                }
            }
        
```

```

        }
        catch (NoSuchEntityException)
        {
            keyReady = false;
        }
    } while (!keyReady);

    return keyReady;
}
}

using Microsoft.Extensions.Configuration;
namespace IAMBasics;
public class IAMBasics
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for the AWS service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
                        LogLevel.Information)
                .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonIdentityManagementService>()
                    .AddTransient<IAMWrapper>()
                    .AddTransient<UIWrapper>()
                )
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<IAMBasics>();

        IConfiguration configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load test settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally load local settings.
            .Build();

        // Values needed for user, role, and policies.
        string userName = configuration["UserName"];
        string s3PolicyName = configuration["S3PolicyName"];
        string roleName = configuration["RoleName"];

        var iamWrapper = host.Services.GetRequiredService<IAMWrapper>();
        var uiWrapper = host.Services.GetRequiredService<UIWrapper>();

        uiWrapper.DisplayBasicsOverview();
        uiWrapper.PressEnter();

        // First create a user. By default, the new user has
        // no permissions.
        uiWrapper.DisplayTitle("Create User");
        Console.WriteLine($"Creating a new user with user name: {userName}.");
        var user = await iamWrapper.CreateUserAsync(userName);
        var userArn = user.Arn;
    }
}

```

```

Console.WriteLine($"Successfully created user: {userName} with ARN: {userArn}.");
uiWrapper.WaitABit(15, "Now let's wait for the user to be ready for use.");

// Define a role policy document that allows the new user
// to assume the role.
string assumeRolePolicyDocument = "{" +
    "\"Version\": \"2012-10-17\", " +
    "\"Statement\": [{" +
        "\"Effect\": \"Allow\", " +
        "\"Principal\": {\" + " +
        $" \"AWS\": \"{userArn}\", \" + " +
        "}, \" + " +
        "\"Action\": \"sts:AssumeRole\" + " +
        "}]" +
    "}";

// Permissions to list all buckets.
string policyDocument = "{" +
    "\"Version\": \"2012-10-17\", " +
    "\"Statement\": [{" +
        "\"Action\": [\"s3>ListAllMyBuckets\"], \" + " +
        "\"Effect\": \"Allow\", \" + " +
        "\"Resource\": \"*\" + " +
    "}]" +
    "}";

// Create an AccessKey for the user.
uiWrapper.DisplayTitle("Create access key");
Console.WriteLine("Now let's create an access key for the new user.");
var accessKey = await iamWrapper.CreateAccessKeyAsync(userName);

var accessKeyId = accessKey.AccessKeyId;
var secretAccessKey = accessKey.SecretAccessKey;

Console.WriteLine($"We have created the access key with Access key id: {accessKeyId}.");
Console.WriteLine("Now let's wait until the IAM access key is ready to use.");
var keyReady = await iamWrapper.WaitUntilAccessKeyIsReady(accessKeyId);

// Now try listing the Amazon Simple Storage Service (Amazon S3)
// buckets. This should fail at this point because the user doesn't
// have permissions to perform this task.
uiWrapper.DisplayTitle("Try to display Amazon S3 buckets");
Console.WriteLine("Now let's try to display a list of the user's Amazon S3 buckets.");
var s3Client1 = new AmazonS3Client(accessKeyId, secretAccessKey);
var stsClient1 = new AmazonSecurityTokenServiceClient(accessKeyId, secretAccessKey);

var s3Wrapper = new S3Wrapper(s3Client1, stsClient1);
var buckets = await s3Wrapper.ListMyBucketsAsync();

Console.WriteLine(buckets is null
    ? "As expected, the call to list the buckets has returned a null list."
    : "Something went wrong. This shouldn't have worked.");

uiWrapper.PressEnter();

uiWrapper.DisplayTitle("Create IAM role");
Console.WriteLine($"Creating the role: {roleName}");

// Creating an IAM role to allow listing the S3 buckets. A role name

```

```
// is not case sensitive and must be unique to the account for which it
// is created.
var roleArn = await iamWrapper.CreateRoleAsync(roleName,
assumeRolePolicyDocument);

uiWrapper.PressEnter();

// Create a policy with permissions to list S3 buckets.
uiWrapper.DisplayTitle("Create IAM policy");
Console.WriteLine($"Creating the policy: {s3PolicyName}");
Console.WriteLine("with permissions to list the Amazon S3 buckets for the
account.");
var policy = await iamWrapper.CreatePolicyAsync(s3PolicyName,
policyDocument);

// Wait 15 seconds for the IAM policy to be available.
uiWrapper.WaitABit(15, "Waiting for the policy to be available.");

// Attach the policy to the role you created earlier.
uiWrapper.DisplayTitle("Attach new IAM policy");
Console.WriteLine("Now let's attach the policy to the role.");
await iamWrapper.AttachRolePolicyAsync(policy.Arn, roleName);

// Wait 15 seconds for the role to be updated.
Console.WriteLine();
uiWrapper.WaitABit(15, "Waiting for the policy to be attached.");

// Use the AWS Security Token Service (AWS STS) to have the user
// assume the role we created.
var stsClient2 = new AmazonSecurityTokenServiceClient(accessKeyId,
secretAccessKey);

// Wait for the new credentials to become valid.
uiWrapper.WaitABit(10, "Waiting for the credentials to be valid.");

var assumedRoleCredentials = await s3Wrapper.AssumeS3RoleAsync("temporary-
session", roleArn);

// Try again to list the buckets using the client created with
// the new user's credentials. This time, it should work.
var s3Client2 = new AmazonS3Client(assumedRoleCredentials);

s3Wrapper.UpdateClients(s3Client2, stsClient2);

buckets = await s3Wrapper.ListMyBucketsAsync();

uiWrapper.DisplayTitle("List Amazon S3 buckets");
Console.WriteLine("This time we should have buckets to list.");
if (buckets is not null)
{
    buckets.ForEach(bucket =>
    {
        Console.WriteLine($"{bucket.BucketName} created:
{bucket.CreationDate}");
    });
}

uiWrapper.PressEnter();

// Now clean up all the resources used in the example.
uiWrapper.DisplayTitle("Clean up resources");
Console.WriteLine("Thank you for watching. The IAM Basics demo is
complete.");
Console.WriteLine("Please wait while we clean up the resources we
created.");
```

```

        await iamWrapper.DetachRolePolicyAsync(policy.Arn, roleName);

        await iamWrapper.DeletePolicyAsync(policy.Arn);

        await iamWrapper.DeleteRoleAsync(roleName);

        await iamWrapper.DeleteAccessKeyAsync(accessKeyId, userName);

        await iamWrapper.DeleteUserAsync(userName);

        uiWrapper.PressEnter();

        Console.WriteLine("All done cleaning up our resources. Thank you for your
patience.");
    }
}

namespace IamScenariosCommon;

using System.Net;

/// <summary>
/// A class to perform Amazon Simple Storage Service (Amazon S3) actions for
/// the IAM Basics scenario.
/// </summary>
public class S3Wrapper
{
    private IAmazonS3 _s3Service;
    private IAmazonSecurityTokenService _stsService;

    /// <summary>
    /// Constructor for the S3Wrapper class.
    /// </summary>
    /// <param name="s3Service">An Amazon S3 client object.</param>
    /// <param name="stsService">An AWS Security Token Service (AWS STS)
    /// client object.</param>
    public S3Wrapper(IAmazonS3 s3Service, IAmazonSecurityTokenService stsService)
    {
        _s3Service = s3Service;
        _stsService = stsService;
    }

    // snippet.start:[STS.dotnetv3.AssumeS3Role]
    /// <summary>
    /// Assumes an AWS Identity and Access Management (IAM) role that allows
    /// Amazon S3 access for the current session.
    /// </summary>
    /// <param name="roleSession">A string representing the current session.</param>
    /// <param name="roleToAssume">The name of the IAM role to assume.</param>
    /// <returns>Credentials for the newly assumed IAM role.</returns>
    public async Task<Credentials> AssumeS3RoleAsync(string roleSession, string
roleToAssume)
    {
        // Create the request to use with the AssumeRoleAsync call.
        var request = new AssumeRoleRequest()
        {
            RoleSessionName = roleSession,
            RoleArn = roleToAssume,
        };

        var response = await _stsService.AssumeRoleAsync(request);

        return response.Credentials;
    }
}

```

```

// snippet.end:[STS.dotnetv3.AssumeS3Role]

/// <summary>
/// Delete an S3 bucket.
/// </summary>
/// <param name="bucketName">Name of the S3 bucket to delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteBucketAsync(string bucketName)
{
    var result = await _s3Service.DeleteBucketAsync(new DeleteBucketRequest
{ BucketName = bucketName });
    return result.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// List the buckets that are owned by the user's account.
/// </summary>
/// <returns>Async Task.</returns>
public async Task<List<S3Bucket>?> ListMyBucketsAsync()
{
    try
    {
        // Get the list of buckets accessible by the new user.
        var response = await _s3Service.ListBucketsAsync();

        return response.Buckets;
    }
    catch (AmazonS3Exception ex)
    {
        // Something else went wrong. Display the error message.
        Console.WriteLine($"Error: {ex.Message}");
        return null;
    }
}

/// <summary>
/// Create a new S3 bucket.
/// </summary>
/// <param name="bucketName">The name for the new bucket.</param>
/// <returns>A Boolean value indicating whether the action completed
/// successfully.</returns>
public async Task<bool> PutBucketAsync(string bucketName)
{
    var response = await _s3Service.PutBucketAsync(new PutBucketRequest
{ BucketName = bucketName });
    return response.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// Update the client objects with new client objects. This is available
/// because the scenario uses the methods of this class without and then
/// with the proper permissions to list S3 buckets.
/// </summary>
/// <param name="s3Service">The Amazon S3 client object.</param>
/// <param name="stsService">The AWS STS client object.</param>
public void UpdateClients(IAmazonS3 s3Service, IAmazonSecurityTokenService
stsService)
{
    _s3Service = s3Service;
    _stsService = stsService;
}
}

namespace IamScenariosCommon;

```

```

public class UIWrapper
{
    public readonly string SepBar = new(' ', Console.WindowWidth);

    ///<summary>
    /// Show information about the IAM Groups scenario.
    ///</summary>
    public void DisplayGroupsOverview()
    {
        Console.Clear();

        DisplayTitle("Welcome to the IAM Groups Demo");
        Console.WriteLine("This example application does the following:");
        Console.WriteLine("\t1. Creates an Amazon Identity and Access Management (IAM) group.");
        Console.WriteLine("\t2. Adds an IAM policy to the IAM group giving it full access to Amazon S3.");
        Console.WriteLine("\t3. Creates a new IAM user.");
        Console.WriteLine("\t4. Creates an IAM access key for the user.");
        Console.WriteLine("\t5. Adds the user to the IAM group.");
        Console.WriteLine("\t6. Lists the buckets on the account.");
        Console.WriteLine("\t7. Proves that the user has full Amazon S3 access by creating a bucket.");
        Console.WriteLine("\t8. List the buckets again to show the new bucket.");
        Console.WriteLine("\t9. Cleans up all the resources created.");
    }

    ///<summary>
    /// Show information about the IAM Basics scenario.
    ///</summary>
    public void DisplayBasicsOverview()
    {
        Console.Clear();

        DisplayTitle("Welcome to IAM Basics");
        Console.WriteLine("This example application does the following:");
        Console.WriteLine("\t1. Creates a user with no permissions.");
        Console.WriteLine("\t2. Creates a role and policy that grant s3>ListAllMyBuckets permission.");
        Console.WriteLine("\t3. Grants the user permission to assume the role.");
        Console.WriteLine("\t4. Creates an S3 client object as the user and tries to list buckets (this will fail).");
        Console.WriteLine("\t5. Gets temporary credentials by assuming the role.");
        Console.WriteLine("\t6. Creates a new S3 client object with the temporary credentials and lists the buckets (this will succeed).");
        Console.WriteLine("\t7. Deletes all the resources.");
    }

    ///<summary>
    /// Display a message and wait until the user presses enter.
    ///</summary>
    public void PressEnter()
    {
        Console.Write("\nPress <Enter> to continue. ");
        _ = Console.ReadLine();
        Console.WriteLine();
    }

    ///<summary>
    /// Pad a string with spaces to center it on the console display.
    ///</summary>
    ///<param name="strToCenter">The string to be centered.</param>
    ///<returns>The padded string.</returns>
    public string CenterString(string strToCenter)
    {

```

```
        var padAmount = (Console.WindowWidth - strToCenter.Length) / 2;
        var leftPad = new string(' ', padAmount);
        return $"{leftPad}{strToCenter}";
    }

    ///<summary>
    /// Display a line of hyphens, the centered text of the title, and another
    /// line of hyphens.
    ///</summary>
    ///<param name="strTitle">The string to be displayed.</param>
    public void DisplayTitle(string strTitle)
    {
        Console.WriteLine(SepBar);
        Console.WriteLine(CenterString(strTitle));
        Console.WriteLine(SepBar);
    }

    ///<summary>
    /// Display a countdown and wait for a number of seconds.
    ///</summary>
    ///<param name="numSeconds">The number of seconds to wait.</param>
    public void WaitABit(int numSeconds, string msg)
    {
        Console.WriteLine(msg);

        // Wait for the requested number of seconds.
        for (int i = numSeconds; i > 0; i--)
        {
            System.Threading.Thread.Sleep(1000);
            Console.Write($"{i}...");
        }

        PressEnter();
    }
}
```

- For API details, see the following topics in *AWS SDK for .NET API Reference*.
 - [AttachRolePolicy](#)
 - [CreateAccessKey](#)
 - [CreatePolicy](#)
 - [CreateRole](#)
 - [CreateUser](#)
 - [DeleteAccessKey](#)
 - [DeletePolicy](#)
 - [DeleteRole](#)
 - [DeleteUser](#)
 - [DeleteUserPolicy](#)
 - [DetachRolePolicy](#)
 - [PutUserPolicy](#)

Bash

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function iam_create_user_assume_role
#
# Scenario to create an IAM user, create an IAM role, and apply the role to the
# user.
#
#      "IAM access" permissions are needed to run this code.
#      "STS assume role" permissions are needed to run this code. (Note: It might be
# necessary to
#          create a custom policy).
#
# Returns:
#      0 - If successful.
#      1 - If an error occurred.
#####
function iam_create_user_assume_role() {
    if [ "$IAM_OPERATIONS_SOURCED" != "True" ]; then
        # shellcheck disable=SC1091
        source ./iam_operations.sh
    fi
}

echo_repeat "*" 88
echo "Welcome to the IAM create user and assume role demo."
echo
echo "This demo will create an IAM user, create an IAM role, and apply the role
to the user."
echo_repeat "*" 88
echo

echo -n "Enter a name for a new IAM user: "
get_input
user_name=$get_input_result

local user_arn
user_arn=$(iam_create_user -u "$user_name")

# shellcheck disable=SC2181
if [[ ${?} == 0 ]]; then
    echo "Created demo IAM user named $user_name"
else
    errecho "$user_arn"
    errecho "The user failed to create. This demo will exit."
    return 1
fi

local access_key_response
access_key_response=$(iam_create_user_access_key -u "$user_name")
# shellcheck disable=SC2181
if [[ ${?} != 0 ]]; then
    errecho "The access key failed to create. This demo will exit."
    clean_up "$user_name"
    return 1
fi

IFS=$'\t ' read -r -a access_key_values <<<"$access_key_response"
local key_name=${access_key_values[0]}
local key_secret=${access_key_values[1]}

echo "Created access key named $key_name"

echo "Wait 10 seconds for the user to be ready."
sleep 10
```

```

echo_repeat "*" 88
echo

local iam_role_name
iam_role_name=$(generate_random_name "test-role")
echo "Creating a role named $iam_role_name with user $user_name as the
principal."

local assume_role_policy_document="{
  \"Version\": \"2012-10-17\",
  \"Statement\": [
    {
      \"Effect\": \"Allow\",
      \"Principal\": {\"AWS\": \"$user_arn\"},
      \"Action\": \"sts:AssumeRole\"
    }
  ]
}"

local role_arn
role_arn=$(iam_create_role -n "$iam_role_name" -p "$assume_role_policy_document")

# shellcheck disable=SC2181
if [ ${?} == 0 ]; then
  echo "Created IAM role named $iam_role_name"
else
  errecho "The role failed to create. This demo will exit."
  clean_up "$user_name" "$key_name"
  return 1
fi

local policy_name
policy_name=$(generate_random_name "test-policy")
local policy_document="{
  \"Version\": \"2012-10-17\",
  \"Statement\": [
    {
      \"Effect\": \"Allow\",
      \"Action\": \"s3>ListAllMyBuckets\",
      \"Resource\": \"arn:aws:s3:::*\"}
  ]
}"

local policy_arn
policy_arn=$(iam_create_policy -n "$policy_name" -p "$policy_document")
# shellcheck disable=SC2181
if [[ ${?} == 0 ]]; then
  echo "Created IAM policy named $policy_name"
else
  errecho "The policy failed to create."
  clean_up "$user_name" "$key_name" "$iam_role_name"
  return 1
fi

if (iam_attach_role_policy -n "$iam_role_name" -p "$policy_arn"); then
  echo "Attached policy $policy_arn to role $iam_role_name"
else
  errecho "The policy failed to attach."
  clean_up "$user_name" "$key_name" "$iam_role_name" "$policy_arn"
  return 1
fi

local assume_role_policy_document="{
  \"Version\": \"2012-10-17\",
  \"Statement\": [
    {
      \"Effect\": \"Allow\",
      \"Action\": \"sts:AssumeRole\",
      \"Resource\": \"$role_arn\"}
  ]
}"

local assume_role_policy_name
assume_role_policy_name=$(generate_random_name "test-assume-role-")

```

```

# shellcheck disable=SC2181
local assume_role_policy_arn
assume_role_policy_arn=$(iam_create_policy -n "$assume_role_policy_name" -p
"$assume_role_policy_document")
# shellcheck disable=SC2181
if [ ${?} == 0 ]; then
    echo "Created IAM policy named $assume_role_policy_name for sts assume role"
else
    errecho "The policy failed to create."
    clean_up "$user_name" "$key_name" "$iam_role_name" "$policy_arn" "$policy_arn"
    return 1
fi

echo "Wait 10 seconds to give AWS time to propagate these new resources and
connections."
sleep 10
echo_repeat "*" 88
echo

echo "Try to list buckets without the new user assuming the role."
echo_repeat "*" 88
echo

# Set the environment variables for the created user.
# bashsupport disable=BP2001
export AWS_ACCESS_KEY_ID=$key_name
# bashsupport disable=BP2001
export AWS_SECRET_ACCESS_KEY=$key_secret

local buckets
buckets=$(s3_list_buckets)

# shellcheck disable=SC2181
if [ ${?} == 0 ]; then
    local bucket_count
    bucket_count=$(echo "$buckets" | wc -w | xargs)
    echo "There are $bucket_count buckets in the account. This should not have
happened."
else
    errecho "Because the role with permissions has not been assumed, listing
buckets failed."
fi

echo
echo_repeat "*" 88
echo "Now assume the role $iam_role_name and list the buckets."
echo_repeat "*" 88
echo

local credentials

credentials=$(sts_assume_role -r "$role_arn" -n "AssumeRoleDemoSession")
# shellcheck disable=SC2181
if [ ${?} == 0 ]; then
    echo "Assumed role $iam_role_name"
else
    errecho "Failed to assume role."
    export AWS_ACCESS_KEY_ID=""
    export AWS_SECRET_ACCESS_KEY=""
    clean_up "$user_name" "$key_name" "$iam_role_name" "$policy_arn" "$policy_arn"
    "$assume_role_policy_arn"
    return 1
fi

IFS=$'\t ' read -r -a credentials <<<"$credentials"

```

```

export AWS_ACCESS_KEY_ID=${credentials[0]}
export AWS_SECRET_ACCESS_KEY=${credentials[1]}
# bashsupport disable=BP2001
export AWS_SESSION_TOKEN=${credentials[2]}

buckets=$(s3_list_buckets)

# shellcheck disable=SC2181
if [ ${?} == 0 ]; then
    local bucket_count
    bucket_count=$(echo "$buckets" | wc -w | xargs)
    echo "There are $bucket_count buckets in the account. Listing buckets succeeded
because of"
    echo "the assumed role."
else
    errecho "Failed to list buckets. This should not happen."
    export AWS_ACCESS_KEY_ID=""
    export AWS_SECRET_ACCESS_KEY=""
    export AWS_SESSION_TOKEN=""
    clean_up "$user_name" "$key_name" "$iam_role_name" "$policy_arn" "$policy_arn"
"$assume_role_policy_arn"
    return 1
fi

local result=0
export AWS_ACCESS_KEY_ID=""
export AWS_SECRET_ACCESS_KEY=""

echo
echo_repeat ** 88
echo "The created resources will now be deleted."
echo_repeat ** 88
echo

clean_up "$user_name" "$key_name" "$iam_role_name" "$policy_arn" "$policy_arn"
"$assume_role_policy_arn"

# shellcheck disable=SC2181
if [[ ${?} -ne 0 ]]; then
    result=1
fi

return $result
}

#####
# function iam_user_exists
#
# This function checks to see if the specified AWS Identity and Access Management
# (IAM) user already exists.
#
# Parameters:
#     $1 - The name of the IAM user to check.
#
# Returns:
#     0 - If the user already exists.
#     1 - If the user doesn't exist.
#####
function iam_user_exists() {
    local user_name
    user_name=$1

    # Check whether the IAM user already exists.
    # We suppress all output - we're interested only in the return code.

```

```

local errors
errors=$(aws iam get-user \
--user-name "$user_name" 2>&1 >/dev/null)

local error_code=${?}

if [[ $error_code -eq 0 ]]; then
    return 0 # 0 in Bash script means true.
else
    if [[ $errors != *"error""*(NoSuchEntity)"* ]]; then
        aws_cli_error_log $error_code
        errecho "Error calling iam get-user $errors"
    fi

    return 1 # 1 in Bash script means false.
fi
}

#####
# function iam_create_user
#
# This function creates the specified IAM user, unless
# it already exists.
#
# Parameters:
#     -u user_name -- The name of the user to create.
#
# Returns:
#     The ARN of the user.
#     And:
#         0 - If successful.
#         1 - If it fails.
#####
function iam_create_user() {
    local user_name response
    local option OPTARG # Required to use getopt command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_create_user"
        echo "Creates an AWS Identity and Access Management (IAM) user. You must supply"
        echo "a username:"
        echo "    -u user_name      The name of the user. It must be unique within the"
        echo "account."
        echo ""
    }

    # Retrieve the calling parameters.
    while getopt "u:h" option; do
        case "${option}" in
            u) user_name="${OPTARG}" ;;
            h)
                usage
                return 0
                ;;
            \?)
                echo "Invalid parameter"
                usage
                return 1
                ;;
        esac
    done
    export OPTIND=1

    if [[ -z "$user_name" ]]; then
        errecho "ERROR: You must provide a username with the -u parameter."
    fi
}

```

```

        usage
        return 1
    fi

    iecho "Parameters:\n"
    iecho "    User name: $user_name"
    iecho ""

    # If the user already exists, we don't want to try to create it.
    if (iam_user_exists "$user_name"); then
        errecho "ERROR: A user with that name already exists in the account."
        return 1
    fi

    response=$(aws iam create-user --user-name "$user_name" \
        --output text \
        --query 'User.Arn')

    local error_code=${?}

    if [[ $error_code -ne 0 ]]; then
        aws_cli_error_log $error_code
        errecho "ERROR: AWS reports create-user operation failed.$response"
        return 1
    fi

    echo "$response"

    return 0
}

#####
# function iam_create_user_access_key
#
# This function creates an IAM access key for the specified user.
#
# Parameters:
#     -u user_name -- The name of the IAM user.
#     [-f file_name] -- The optional file name for the access key output.
#
# Returns:
#     [access_key_id access_key_secret]
#     And:
#         0 - If successful.
#         1 - If it fails.
#####
function iam_create_user_access_key() {
    local user_name file_name response
    local option OPTARG # Required to use getopt command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_create_user_access_key"
        echo "Creates an AWS Identity and Access Management (IAM) key pair."
        echo " -u user_name    The name of the IAM user."
        echo " [-f file_name]  Optional file name for the access key output."
        echo ""
    }

    # Retrieve the calling parameters.
    while getopt "u:f:h" option; do
        case "${option}" in
            u) user_name="${OPTARG}" ;;
            f) file_name="${OPTARG}" ;;
            h)
                usage

```

```

        return 0
    ;;
\?)
    echo "Invalid parameter"
    usage
    return 1
;;
esac
done
export OPTIND=1

if [[ -z "$user_name" ]]; then
    errecho "ERROR: You must provide a username with the -u parameter."
    usage
    return 1
fi

response=$(aws iam create-access-key \
    --user-name "$user_name" \
    --output text)

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports create-access-key operation failed.$response"
    return 1
fi

if [[ -n "$file_name" ]]; then
    echo "$response" >"$file_name"
fi

local key_id key_secret
# shellcheck disable=SC2086
key_id=$(echo $response | cut -f 2 -d ' ')
# shellcheck disable=SC2086
key_secret=$(echo $response | cut -f 4 -d ' ')

echo "$key_id $key_secret"

return 0
}

#####
# function iam_create_role
#
# This function creates an IAM role.
#
# Parameters:
#     -n role_name -- The name of the IAM role.
#     -p policy_json -- The assume role policy document.
#
# Returns:
#     The ARN of the role.
#     And:
#         0 - If successful.
#         1 - If it fails.
#####
function iam_create_role() {
    local role_name policy_document response
    local option OPTARG # Required to use getopts command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_create_user_access_key"

```

```

echo "Creates an AWS Identity and Access Management (IAM) role."
echo " -n role_name -- The name of the IAM role."
echo " -p policy_json -- The assume role policy document."
echo ""
}

# Retrieve the calling parameters.
while getopts "n:p:h" option; do
    case "${option}" in
        n) role_name="${OPTARG}" ;;
        p) policy_document="${OPTARG}" ;;
        h)
            usage
            return 0
            ;;
        \?)
            echo "Invalid parameter"
            usage
            return 1
            ;;
    esac
done
export OPTIND=1

if [[ -z "$role_name" ]]; then
    errecho "ERROR: You must provide a role name with the -n parameter."
    usage
    return 1
fi

if [[ -z "$policy_document" ]]; then
    errecho "ERROR: You must provide a policy document with the -p parameter."
    usage
    return 1
fi

response=$(aws iam create-role \
--role-name "$role_name" \
--assume-role-policy-document "$policy_document" \
--output text \
--query Role.Arn)

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports create-role operation failed.\n$response"
    return 1
fi

echo "$response"

return 0
}

#####
# function iam_create_policy
#
# This function creates an IAM policy.
#
# Parameters:
#     -n policy_name -- The name of the IAM policy.
#     -p policy_json -- The policy document.
#
# Returns:
#     0 - If successful.
#
#####

```

```

#      1 - If it fails.
#####
function iam_create_policy() {
    local policy_name policy_document response
    local option OPTARG # Required to use getopts command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_create_policy"
        echo "Creates an AWS Identity and Access Management (IAM) policy."
        echo " -n policy_name   The name of the IAM policy."
        echo " -p policy_json -- The policy document."
        echo ""
    }

    # Retrieve the calling parameters.
    while getopts "n:p:h" option; do
        case "${option}" in
            n) policy_name="${OPTARG}" ;;
            p) policy_document="${OPTARG}" ;;
            h)
                usage
                return 0
                ;;
            \?)
                echo "Invalid parameter"
                usage
                return 1
                ;;
        esac
    done
    export OPTIND=1

    if [[ -z "$policy_name" ]]; then
        errecho "ERROR: You must provide a policy name with the -n parameter."
        usage
        return 1
    fi

    if [[ -z "$policy_document" ]]; then
        errecho "ERROR: You must provide a policy document with the -p parameter."
        usage
        return 1
    fi

    response=$(aws iam create-policy \
        --policy-name "$policy_name" \
        --policy-document "$policy_document" \
        --output text \
        --query Policy.Arn)

    local error_code=${?}

    if [[ $error_code -ne 0 ]]; then
        aws_cli_error_log $error_code
        errecho "ERROR: AWS reports create-policy operation failed.\n$response"
        return 1
    fi

    echo "$response"
}

#####
# function iam_attach_role_policy
#
# This function attaches an IAM policy to a role.

```

```

#
# Parameters:
#   -n role_name -- The name of the IAM role.
#   -p policy_ARN -- The IAM policy document ARN..
#
# Returns:
#   0 - If successful.
#   1 - If it fails.
#####
function iam_attach_role_policy() {
    local role_name policy_arn response
    local option OPTARG # Required to use getopts command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_attach_role_policy"
        echo "Attaches an AWS Identity and Access Management (IAM) policy to an IAM"
        echo "role."
        echo "  -n role_name  The name of the IAM role."
        echo "  -p policy_ARN -- The IAM policy document ARN."
        echo ""
    }

    # Retrieve the calling parameters.
    while getopts "n:p:h" option; do
        case "${option}" in
            n) role_name="${OPTARG}" ;;
            p) policy_arn="${OPTARG}" ;;
            h)
                usage
                return 0
                ;;
            \?)
                echo "Invalid parameter"
                usage
                return 1
                ;;
        esac
    done
    export OPTIND=1

    if [[ -z "$role_name" ]]; then
        errecho "ERROR: You must provide a role name with the -n parameter."
        usage
        return 1
    fi

    if [[ -z "$policy_arn" ]]; then
        errecho "ERROR: You must provide a policy ARN with the -p parameter."
        usage
        return 1
    fi

    response=$(aws iam attach-role-policy \
        --role-name "$role_name" \
        --policy-arn "$policy_arn")

    local error_code=${?}

    if [[ $error_code -ne 0 ]]; then
        aws_cli_error_log $error_code
        errecho "ERROR: AWS reports attach-role-policy operation failed.\n$response"
        return 1
    fi

    echo "$response"
}

```

```

        return 0
    }

#####
# function iam_detach_role_policy
#
# This function detaches an IAM policy to a role.
#
# Parameters:
#     -n role_name -- The name of the IAM role.
#     -p policy_ARN -- The IAM policy document ARN..
#
# Returns:
#     0 - If successful.
#     1 - If it fails.
#####
function iam_detach_role_policy() {
    local role_name policy_arn response
    local option OPTARG # Required to use getopts command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_detach_role_policy"
        echo "Detaches an AWS Identity and Access Management (IAM) policy to an IAM"
        echo "role."
        echo "  -n role_name  The name of the IAM role."
        echo "  -p policy_ARN -- The IAM policy document ARN."
        echo ""
    }

    # Retrieve the calling parameters.
    while getopts "n:p:h" option; do
        case "${option}" in
            n) role_name="${OPTARG}";;
            p) policy_arn="${OPTARG}";;
            h)
                usage
                return 0
                ;;
            \?)
                echo "Invalid parameter"
                usage
                return 1
                ;;
        esac
    done
    export OPTIND=1

    if [[ -z "$role_name" ]]; then
        errecho "ERROR: You must provide a role name with the -n parameter."
        usage
        return 1
    fi

    if [[ -z "$policy_arn" ]]; then
        errecho "ERROR: You must provide a policy ARN with the -p parameter."
        usage
        return 1
    fi

    response=$(aws iam detach-role-policy \
        --role-name "$role_name" \
        --policy-arn "$policy_arn")

    local error_code=${?}

```

```

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports detach-role-policy operation failed.\n$response"
    return 1
fi

echo "$response"

return 0
}

#####
# function iam_delete_policy
#
# This function deletes an IAM policy.
#
# Parameters:
#     -n policy_arn -- The name of the IAM policy arn.
#
# Returns:
#     0 - If successful.
#     1 - If it fails.
#####
function iam_delete_policy() {
    local policy_arn response
    local optionOPTARG # Required to use getopts command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_delete_policy"
        echo "Deletes an WS Identity and Access Management (IAM) policy"
        echo "  -n policy_arn -- The name of the IAM policy arn."
        echo ""
    }

    # Retrieve the calling parameters.
    while getopts "n:h" option; do
        case "${option}" in
            n)
                policy_arn="${OPTARG}" ;;
            h)
                usage
                return 0
                ;;
            \?)
                echo "Invalid parameter"
                usage
                return 1
                ;;
        esac
    done
    export OPTIND=1

    if [[ -z "$policy_arn" ]]; then
        errecho "ERROR: You must provide a policy arn with the -n parameter."
        usage
        return 1
    fi

    iecho "Parameters:\n"
    iecho "  Policy arn: $policy_arn"
    iecho ""

    response=$(aws iam delete-policy \
        --policy-arn "$policy_arn")
}

```

```

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-policy operation failed.\n$response"
    return 1
fi

iecho "delete-policy response:$response"
iecho

return 0
}

#####
# function iam_delete_role
#
# This function deletes an IAM role.
#
# Parameters:
#     -n role_name -- The name of the IAM role.
#
# Returns:
#     0 - If successful.
#     1 - If it fails.
#####
function iam_delete_role() {
    local role_name response
    local optionOPTARG # Required to use getopts command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_delete_role"
        echo "Deletes an AWS Identity and Access Management (IAM) role"
        echo " -n role_name -- The name of the IAM role."
        echo ""
    }

    # Retrieve the calling parameters.
    while getopts "n:h" option; do
        case "${option}" in
            n)
                role_name="${OPTARG}"
                ;;
            h)
                usage
                return 0
                ;;
            \?)
                echo "Invalid parameter"
                usage
                return 1
                ;;
        esac
    done
    export OPTIND=1

    echo "role_name:$role_name"
    if [[ -z "$role_name" ]]; then
        errecho "ERROR: You must provide a role name with the -n parameter."
        usage
        return 1
    fi

    iecho "Parameters:\n"
    iecho "    Role name: $role_name"
    iecho ""
}

```

```

response=$(aws iam delete-role \
--role-name "$role_name")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-role operation failed.\n$response"
    return 1
fi

iecho "delete-role response:$response"
iecho

return 0
}

#####
# function iam_delete_access_key
#
# This function deletes an IAM access key for the specified IAM user.
#
# Parameters:
#     -u user_name -- The name of the user.
#     -k access_key -- The access key to delete.
#
# Returns:
#     0 - If successful.
#     1 - If it fails.
#####
function iam_delete_access_key() {
    local user_name access_key response
    local option OPTARG # Required to use getopts command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_delete_access_key"
        echo "Deletes an AWS Identity and Access Management (IAM) access key for the"
        echo "specified IAM user"
        echo "  -u user_name      The name of the user."
        echo "  -k access_key     The access key to delete."
        echo ""
    }

    # Retrieve the calling parameters.
    while getopts "u:k:h" option; do
        case "${option}" in
            u) user_name="${OPTARG}";;
            k) access_key="${OPTARG}";;
            h)
                usage
                return 0
                ;;
            \?)
                echo "Invalid parameter"
                usage
                return 1
                ;;
        esac
    done
    export OPTIND=1

    if [[ -z "$user_name" ]]; then
        errecho "ERROR: You must provide a username with the -u parameter."
        usage
        return 1
    fi
}

```

```

fi

if [[ -z "$access_key" ]]; then
    errecho "ERROR: You must provide an access key with the -k parameter."
    usage
    return 1
fi

iecho "Parameters:\n"
iecho "    Username: $user_name"
iecho "    Access key: $access_key"
iecho ""

response=$(aws iam delete-access-key \
--user-name "$user_name" \
--access-key-id "$access_key")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-access-key operation failed.\n$response"
    return 1
fi

iecho "delete-access-key response:$response"
iecho

return 0
}

#####
# function iam_delete_user
#
# This function deletes the specified IAM user.
#
# Parameters:
#     -u user_name -- The name of the user to create.
#
# Returns:
#     0 - If successful.
#     1 - If it fails.
#####
function iam_delete_user() {
    local user_name response
    local option OPTARG # Required to use getopt command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_delete_user"
        echo "Deletes an AWS Identity and Access Management (IAM) user. You must supply a username:"
        echo "    -u user_name      The name of the user."
        echo ""
    }

    # Retrieve the calling parameters.
    while getopt "u:h" option; do
        case "${option}" in
            u)
                user_name="${OPTARG}" ;;
            h)
                usage
                return 0
                ;;
            \?)
                echo "Invalid parameter"
                ;;
        esac
    done
}

```

```
        usage
        return 1
        ;;
    esac
done
export OPTIND=1

if [[ -z "$user_name" ]]; then
    errecho "ERROR: You must provide a username with the -u parameter."
    usage
    return 1
fi

iecho "Parameters:\n"
iecho "    User name: $user_name"
iecho ""

# If the user does not exist, we don't want to try to delete it.
if (! iam_user_exists "$user_name"); then
    errecho "ERROR: A user with that name does not exist in the account."
    return 1
fi

response=$(aws iam delete-user \
--user-name "$user_name")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-user operation failed.$response"
    return 1
fi

iecho "delete-user response:$response"
iecho

return 0
}
```

- For API details, see the following topics in *AWS CLI Command Reference*.
 - [AttachRolePolicy](#)
 - [CreateAccessKey](#)
 - [CreatePolicy](#)
 - [CreateRole](#)
 - [CreateUser](#)
 - [DeleteAccessKey](#)
 - [DeletePolicy](#)
 - [DeleteRole](#)
 - [DeleteUser](#)
 - [DeleteUserPolicy](#)
 - [DetachRolePolicy](#)
 - [PutUserPolicy](#)

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
namespace AwsDoc {
    namespace IAM {

        ///! Cleanup by deleting created entities.
        /*!
         \sa DeleteCreatedEntities
         \param client: IAM client.
         \param role: IAM role.
         \param user: IAM user.
         \param policy: IAM policy.
        */
        static bool DeleteCreatedEntities(const Aws::IAM::IAMClient &client,
                                         const Aws::IAM::Model::Role &role,
                                         const Aws::IAM::Model::User &user,
                                         const Aws::IAM::Model::Policy &policy);
    }

    static const int LIST_BUCKETS_WAIT_SEC = 20;

    static const char ALLOCATION_TAG[] = "example_code";
}

///! Scenario to create an IAM user, create an IAM role, and apply the role to the
// user.
// "IAM access" permissions are needed to run this code.
// "STS assume role" permissions are needed to run this code. (Note: It might be
// necessary to
//   create a custom policy).
/*!
 \sa iamCreateUserAssumeRoleScenario
 \param clientConfig: Aws client configuration.
 \return bool: Successful completion.
*/
bool AwsDoc::IAM::iamCreateUserAssumeRoleScenario(
    const Aws::Client::ClientConfiguration &clientConfig) {

    Aws::IAM::IAMClient client(clientConfig);
    Aws::IAM::Model::User user;
    Aws::IAM::Model::Role role;
    Aws::IAM::Model::Policy policy;

    // 1. Create a user.
    {
        Aws::IAM::Model::CreateUserRequest request;
        Aws::String uuid = Aws::Utils::UUID::RandomUUID();
        Aws::String userName = "iam-demo-user-" +
            Aws::Utils::StringUtils::ToLower(uuid.c_str());
        request.SetUserName(userName);

        Aws::IAM::Model::CreateUserOutcome outcome = client.CreateUser(request);
        if (!outcome.IsSuccess()) {
            std::cout << "Error creating IAM user " << userName << ":" <<
                outcome.GetError().GetMessage() << std::endl;
            return false;
        }
        else {
```

```

        std::cout << "Successfully created IAM user " << userName << std::endl;
    }

    user = outcome.GetResult().GetUser();
}

// 2. Create a role.
{
    // Get the IAM user for the current client in order to access its ARN.
    Aws::String iamUserArn;
    {
        Aws::IAM::Model:: GetUserRequest request;
        Aws::IAM::Model:: GetUserOutcome outcome = client.GetUser(request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Error getting Iam user. " <<
                outcome.GetError().GetMessage() << std::endl;

            DeleteCreatedEntities(client, role, user, policy);
            return false;
        }
        else {
            std::cout << "Successfully retrieved Iam user "
                << outcome.GetResult().GetUser().GetUserName()
                << std::endl;
        }
    }

    iamUserArn = outcome.GetResult().GetUser().GetArn();
}

Aws::IAM::Model::CreateRoleRequest request;

Aws::String uuid = Aws::Utils::UUID::RandomUUID();
Aws::String roleName = "iam-demo-role-"
    Aws::Utils::StringUtils::ToLower(uuid.c_str());
request.SetRoleName(roleName);

// Build policy document for role.
Aws::Utils::Document jsonStatement;
jsonStatement.WithString("Effect", "Allow");

Aws::Utils::Document jsonPrincipal;
jsonPrincipal.WithString("AWS", iamUserArn);
jsonStatement.WithObject("Principal", jsonPrincipal);
jsonStatement.WithString("Action", "sts:AssumeRole");
jsonStatement.WithObject("Condition", Aws::Utils::Document());

Aws::Utils::Document policyDocument;
policyDocument.WithString("Version", "2012-10-17");

Aws::Utils::Array< Aws::Utils::Document> statements(1);
statements[0] = jsonStatement;
policyDocument.WithArray("Statement", statements);

std::cout << "Setting policy for role\n"
    << policyDocument.View().WriteCompact() << std::endl;

// Set role policy document as JSON string.
request.SetAssumeRolePolicyDocument(policyDocument.View().WriteCompact());

Aws::IAM::Model::CreateRoleOutcome outcome = client.CreateRole(request);
if (!outcome.IsSuccess()) {
    std::cerr << "Error creating role. " <<
        outcome.GetError().GetMessage() << std::endl;

    DeleteCreatedEntities(client, role, user, policy);
    return false;
}

```

```

        }
    else {
        std::cout << "Successfully created a role with name " << roleName
        << std::endl;
    }

    role = outcome.GetResult().GetRole();
}

// 3. Create an IAM policy.
{
    Aws::IAM::Model::CreatePolicyRequest request;
    Aws::String uuid = Aws::Utils::UUID::RandomUUID();
    Aws::String policyName = "iam-demo-policy-" +
        Aws::Utils::StringUtils::ToLower(uuid.c_str());
    request.SetPolicyName(policyName);

    // Build IAM policy document.
    Aws::Utils::Document jsonStatement;
    jsonStatement.WithString("Effect", "Allow");
    jsonStatement.WithString("Action", "s3>ListAllMyBuckets");
    jsonStatement.WithString("Resource", "arn:aws:s3:::*");

    Aws::Utils::Document policyDocument;
    policyDocument.WithString("Version", "2012-10-17");

    Aws::Utils::Array<Aws::Utils::Document> statements(1);
    statements[0] = jsonStatement;
    policyDocument.WithArray("Statement", statements);

    std::cout << "Creating a policy.\n" <<
    policyDocument.View().WriteCompact()
    << std::endl;

    // Set IAM policy document as JSON string.
    request.SetPolicyDocument(policyDocument.View().WriteCompact());

    Aws::IAM::Model::CreatePolicyOutcome outcome =
    client.CreatePolicy(request);
    if (!outcome.IsSuccess()) {
        std::cerr << "Error creating policy. " <<
        outcome.GetError().GetMessage() << std::endl;

        DeleteCreatedEntities(client, role, user, policy);
        return false;
    }
    else {
        std::cout << "Successfully created a policy with name, " << policyName
        << ".\n" << std::endl;
    }

    policy = outcome.GetResult().GetPolicy();
}

// 4. Assume the new role using the AWS Security Token Service (STS).
Aws::STS::Model::Credentials credentials;
{
    Aws::STS::STSCClient stsClient(clientConfig);

    Aws::STS::Model::AssumeRoleRequest request;
    request.SetRoleArn(role.GetArn());
    Aws::String uuid = Aws::Utils::UUID::RandomUUID();
    Aws::String roleSessionName = "iam-demo-role-session-" +
        Aws::Utils::StringUtils::ToLower(uuid.c_str());
}

```

```

request.SetRoleSessionName(roleSessionName);

Aws::STS::Model::AssumeRoleOutcome assumeRoleOutcome;

// Repeatedly call AssumeRole, because there is often a delay
// before the role is available to be assumed.
// Repeat at most 20 times when access is denied.
int count = 0;
while (true) {
    assumeRoleOutcome = stsClient.AssumeRole(request);
    if (!assumeRoleOutcome.IsSuccess()) {
        if (count > 20 ||
            assumeRoleOutcome.GetError().GetErrorCode() !=
            Aws::STS::STSErrors::ACCESS_DENIED) {
            std::cerr << "Error assuming role after 20 tries. " <<
            assumeRoleOutcome.GetError().GetMessage() <<
            std::endl;
            DeleteCreatedEntities(client, role, user, policy);
            return false;
        }
        std::this_thread::sleep_for(std::chrono::seconds(1));
    }
    else {
        std::cout << "Successfully assumed the role after " << count
        << " seconds." << std::endl;
        break;
    }
    count++;
}

credentials = assumeRoleOutcome.GetResult().GetCredentials();
}

// 5. List objects in the bucket (This should fail).
{
    Aws::S3::S3Client s3Client(
        Aws::Auth::AWSCredentials(credentials.GetAccessKeyId(),
                                  credentials.GetSecretAccessKey(),
                                  credentials.GetSessionToken()),
        Aws::MakeShared<Aws::S3::S3EndpointProvider>(ALLOCATION_TAG),
        clientConfig);
    Aws::S3::Model::ListBucketsOutcome listBucketsOutcome =
s3Client.ListBuckets();
    if (!listBucketsOutcome.IsSuccess()) {
        if (listBucketsOutcome.GetError().GetErrorCode() !=
            Aws::S3::S3Errors::ACCESS_DENIED) {
            std::cerr << "Could not lists buckets. " <<
            listBucketsOutcome.GetError().GetMessage() << std::endl;
        }
        else {
            std::cout
                << "Access to list buckets denied because privileges have
not been applied."
                << std::endl;
        }
    }
    else {
        std::cerr
            << "Successfully retrieved bucket lists when this should not
happen."
            << std::endl;
    }
}

```

```

// 6. Attach the policy to the role.
{
    Aws::IAM::Model::AttachRolePolicyRequest request;
    request.SetRoleName(role.GetRoleName());
    request.WithPolicyArn(policy.GetArn());

    Aws::IAM::Model::AttachRolePolicyOutcome outcome = client.AttachRolePolicy(
        request);
    if (!outcome.IsSuccess()) {
        std::cerr << "Error creating policy. " <<
            outcome.GetError().GetMessage() << std::endl;

        DeleteCreatedEntities(client, role, user, policy);
        return false;
    }
    else {
        std::cout << "Successfully attached the policy with name, "
            << policy.GetPolicyName() <<
            ", to the role, " << role.GetRoleName() << "." << std::endl;
    }
}

int count = 0;
// 7. List objects in the bucket (this should succeed).
// Repeatedly call ListBuckets, because there is often a delay
// before the policy with ListBucket permissions has been applied to the role.
// Repeat at most LIST_BUCKETS_WAIT_SEC times when access is denied.
while (true) {
    Aws::S3::S3Client s3Client(
        Aws::Auth::AWSCredentials(credentials.GetAccessKeyId(),
            credentials.GetSecretAccessKey(),
            credentials.GetSessionToken()),
        Aws::MakeShared<Aws::S3::S3EndpointProvider>(ALLOCATION_TAG),
        clientConfig);
    Aws::S3::Model::ListBucketsOutcome listBucketsOutcome =
    s3Client.ListBuckets();
    if (!listBucketsOutcome.IsSuccess()) {
        if ((count > LIST_BUCKETS_WAIT_SEC) ||
            listBucketsOutcome.GetError().GetErrorType() !=
            Aws::S3::S3Errors::ACCESS_DENIED) {
            std::cerr << "Could not lists buckets after " <<
                LIST_BUCKETS_WAIT_SEC << " seconds. " <<
                listBucketsOutcome.GetError().GetMessage() << std::endl;
            DeleteCreatedEntities(client, role, user, policy);
            return false;
        }
        std::this_thread::sleep_for(std::chrono::seconds(1));
    }
    else {

        std::cout << "Successfully retrieved bucket lists after " << count
            << " seconds." << std::endl;
        break;
    }
    count++;
}

// 8. Delete all the created resources.
return DeleteCreatedEntities(client, role, user, policy);
}

bool AwsDoc::IAM::DeleteCreatedEntities(const Aws::IAM::IAMClient &client,
    const Aws::IAM::Model::Role &role,
    const Aws::IAM::Model::User &user,
    const Aws::IAM::Model::Policy &policy) {

```

```

bool result = true;
if (policy.ArnsHasBeenSet()) {
    // Detach the policy from the role.
    {
        Aws::IAM::Model::DetachRolePolicyRequest request;
        request.SetPolicyArn(policy.GetArns());
        request.SetRoleName(role.GetRoleName());

        Aws::IAM::Model::DetachRolePolicyOutcome outcome =
client.DetachRolePolicy(
            request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Error Detaching policy from roles. " <<
                outcome.GetError().GetMessage() << std::endl;
            result = false;
        }
        else {
            std::cout << "Successfully detached the policy with arn "
                << policy.GetArns()
                << " from role " << role.GetRoleName() << "." <<
std::endl;
        }
    }

    // Delete the policy.
    {
        Aws::IAM::Model::DeletePolicyRequest request;
        request.WithPolicyArn(policy.GetArns());

        Aws::IAM::Model::DeletePolicyOutcome outcome =
client.DeletePolicy(request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Error deleting policy. " <<
                outcome.GetError().GetMessage() << std::endl;
            result = false;
        }
        else {
            std::cout << "Successfully deleted the policy with arn "
                << policy.GetArns() << std::endl;
        }
    }

    if (role.RoleIdHasBeenSet()) {
        // Delete the role.
        Aws::IAM::Model::DeleteRoleRequest request;
        request.SetRoleName(role.GetRoleName());

        Aws::IAM::Model::DeleteRoleOutcome outcome = client.DeleteRole(request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Error deleting role. " <<
                outcome.GetError().GetMessage() << std::endl;
            result = false;
        }
        else {
            std::cout << "Successfully deleted the role with name "
                << role.GetRoleName() << std::endl;
        }
    }

    if (user.ArnsHasBeenSet()) {
        // Delete the user.
        Aws::IAM::Model::DeleteUserRequest request;
        request.WithUserName(user.GetUserName());
    }
}

```

```
Aws::IAM::Model::DeleteUserOutcome outcome = client.DeleteUser(request);
if (!outcome.IsSuccess()) {
    std::cerr << "Error deleting user. " <<
        outcome.GetError().GetMessage() << std::endl;
    result = false;
}
else {
    std::cout << "Successfully deleted the user with name "
        << user.GetUserName() << std::endl;
}
}

return result;
}
```

- For API details, see the following topics in *AWS SDK for C++ API Reference*.
 - [AttachRolePolicy](#)
 - [CreateAccessKey](#)
 - [CreatePolicy](#)
 - [CreateRole](#)
 - [CreateUser](#)
 - [DeleteAccessKey](#)
 - [DeletePolicy](#)
 - [DeleteRole](#)
 - [DeleteUser](#)
 - [DeleteUserPolicy](#)
 - [DetachRolePolicy](#)
 - [PutUserPolicy](#)

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Run an interactive scenario at a command prompt.

```
// AssumeRoleScenario shows you how to use the AWS Identity and Access Management
// (IAM)
// service to perform the following actions:
//
// 1. Create a user who has no permissions.
// 2. Create a role that grants permission to list Amazon Simple Storage Service
//     (Amazon S3) buckets for the account.
// 3. Add a policy to let the user assume the role.
// 4. Try and fail to list buckets without permissions.
// 5. Assume the role and list S3 buckets using temporary credentials.
// 6. Delete the policy, role, and user.
type AssumeRoleScenario struct {
    sdkConfig aws.Config
    accountWrapper actions.AccountWrapper
    policyWrapper actions.PolicyWrapper
    roleWrapper actions.RoleWrapper
    userWrapper actions.UserWrapper
```

```

questioner demotools.IQuestioner
helper IScenarioHelper
isTestRun bool
}

// NewAssumeRoleScenario constructs an AssumeRoleScenario instance from a
// configuration.
// It uses the specified config to get an IAM client and create wrappers for the
// actions
// used in the scenario.
func NewAssumeRoleScenario(sdkConfig aws.Config, questioner demotools.IQuestioner,
    helper IScenarioHelper) AssumeRoleScenario {
    iamClient := iam.NewFromConfig(sdkConfig)
    return AssumeRoleScenario{
        sdkConfig:   sdkConfig,
        accountWrapper: actions.AccountWrapper{IamClient: iamClient},
        policyWrapper: actions.PolicyWrapper{IamClient: iamClient},
        roleWrapper:   actions.RoleWrapper{IamClient: iamClient},
        userWrapper:   actions.UserWrapper{IamClient: iamClient},
        questioner:    questioner,
        helper:       helper,
    }
}

// addTestOptions appends the API options specified in the original configuration
// to
// another configuration. This is used to attach the middleware stubber to clients
// that are constructed during the scenario, which is needed for unit testing.
func (scenario AssumeRoleScenario) addTestOptions(scenarioConfig *aws.Config) {
    if scenario.isTestRun {
        scenarioConfig.APIOptions = append(scenarioConfig.APIOptions,
            scenario.sdkConfig.APIOptions...)
    }
}

// Run runs the interactive scenario.
func (scenario AssumeRoleScenario) Run() {
    defer func() {
        if r := recover(); r != nil {
            log.Printf("Something went wrong with the demo.\n")
            log.Println(r)
        }
    }()
}

log.Println(strings.Repeat("-", 88))
log.Println("Welcome to the AWS Identity and Access Management (IAM) assume role
demo.")
log.Println(strings.Repeat("-", 88))

user := scenario.CreateUser()
accessKey := scenario.CreateAccessKey(user)
role := scenario.CreateRoleAndPolicies(user)
noPermsConfig := scenario.ListBucketsWithoutPermissions(accessKey)
scenario.ListBucketsWithAssumedRole(noPermsConfig, role)
scenario.Cleanup(user, role)

log.Println(strings.Repeat("-", 88))
log.Println("Thanks for watching!")
log.Println(strings.Repeat("-", 88))
}

// CreateUser creates a new IAM user. This user has no permissions.
func (scenario AssumeRoleScenario) CreateUser() *types.User {
    log.Println("Let's create an example user with no permissions.")
    userName := scenario.questioner.Ask("Enter a name for the example user:",
        demotools.NotEmpty{})
}

```

```

user, err := scenario.userWrapper.GetUser(userName)
if err != nil {
    panic(err)
}
if user == nil {
    user, err = scenario.userWrapper.CreateUser(userName)
    if err != nil {
        panic(err)
    }
    log.Printf("Created user %v.\n", *user.UserName)
} else {
    log.Printf("User %v already exists.\n", *user.UserName)
}
log.Println(strings.Repeat("-", 88))
return user
}

// CreateAccessKey creates an access key for the user.
func (scenario AssumeRoleScenario) CreateAccessKey(user *types.User)
*types.AccessKey {
accessKey, err := scenario.userWrapper.CreateAccessKeyPair(*user.UserName)
if err != nil {
    panic(err)
}
log.Printf("Created access key %v for your user.", *accessKey.AccessKeyId)
log.Println("Waiting a few seconds for your user to be ready...")
scenario.helper.Pause(10)
log.Println(strings.Repeat("-", 88))
return accessKey
}

// CreateRoleAndPolicies creates a policy that grants permission to list S3 buckets
for
// the current account and attaches the policy to a newly created role. It also
// adds an
// inline policy to the specified user that grants the user permission to assume
// the role.
func (scenario AssumeRoleScenario) CreateRoleAndPolicies(user *types.User)
*types.Role {
log.Println("Let's create a role and policy that grant permission to list S3
buckets.")
scenario.questioner.Ask("Press Enter when you're ready.")
listBucketsRole, err := scenario.roleWrapper.CreateRole(scenario.helper.GetName(),
*user.Arn)
if err != nil {panic(err)}
log.Printf("Created role %v.\n", *listBucketsRole.RoleName)
listBucketsPolicy, err := scenario.policyWrapper.CreatePolicy(
    scenario.helper.GetName(), []string{"s3>ListAllMyBuckets"}, "arn:aws:s3:::*")
if err != nil {panic(err)}
log.Printf("Created policy %v.\n", *listBucketsPolicy.PolicyName)
err = scenario.roleWrapper.AttachRolePolicy(*listBucketsPolicy.Arn,
*listBucketsRole.RoleName)
if err != nil {panic(err)}
log.Println("Attached policy %v to role %v.\n", *listBucketsPolicy.PolicyName,
*listBucketsRole.RoleName)
err = scenario.userWrapper.CreateUserPolicy(*user.UserName,
scenario.helper.GetName(),
[]string{"sts:AssumeRole"}, *listBucketsRole.Arn)
if err != nil {panic(err)}
log.Printf("Created an inline policy for user %v that lets the user assume the
role.\n",
*user.UserName)
log.Println("Let's give AWS a few seconds to propagate these new resources and
connections...")
scenario.helper.Pause(10)
log.Println(strings.Repeat("-", 88))

```

```

        return listBucketsRole
    }

    // ListBucketsWithoutPermissions creates an Amazon S3 client from the user's access
    // key
    // credentials and tries to list buckets for the account. Because the user does not
    // have
    // permission to perform this action, the action fails.
    func (scenario AssumeRoleScenario) ListBucketsWithoutPermissions(accessKey
        *types.AccessKey) *aws.Config {
        log.Println("Let's try to list buckets without permissions. This should return an
        AccessDenied error.")
        scenario.questioner.Ask("Press Enter when you're ready.")
        noPermsConfig, err := config.LoadDefaultConfig(context.TODO(),
            config.WithCredentialsProvider(credentials.NewStaticCredentialsProvider(
                *accessKey.AccessKeyId, *accessKey.SecretAccessKey, "")),
        )
        if err != nil {panic(err)}

        // Add test options if this is a test run. This is needed only for testing
        purposes.
        scenario.addTestOptions(&noPermsConfig)

        s3Client := s3.NewFromConfig(noPermsConfig)
        _, err = s3Client.ListBuckets(context.TODO(), &s3.ListBucketsInput{})
        if err != nil {
            // The SDK for Go does not model the AccessDenied error, so check ErrorCode
            directly.
            var ae smithy.APIError
            if errors.As(err, &ae) {
                switch ae.ErrorCode() {
                case "AccessDenied":
                    log.Println("Got AccessDenied error, which is the expected result because\n" +
                        "the ListBuckets call was made without permissions.")
                default:
                    log.Println("Expected AccessDenied, got something else.")
                    panic(err)
                }
            }
        } else {
            log.Println("Expected AccessDenied error when calling ListBuckets without
            permissions,\n" +
                "but the call succeeded. Continuing the example anyway...")
        }
        log.Println(strings.Repeat("-", 88))
        return &noPermsConfig
    }

    // ListBucketsWithAssumedRole performs the following actions:
    //
    // 1. Creates an AWS Security Token Service (AWS STS) client from the config
    // created from
    //   the user's access key credentials.
    // 2. Gets temporary credentials by assuming the role that grants permission to
    // list the
    //   buckets.
    // 3. Creates an Amazon S3 client from the temporary credentials.
    // 4. Lists buckets for the account. Because the temporary credentials are
    // generated by
    //   assuming the role that grants permission, the action succeeds.
    func (scenario AssumeRoleScenario) ListBucketsWithAssumedRole(noPermsConfig
        *aws.Config, role *types.Role) {
        log.Println("Let's assume the role that grants permission to list buckets and try
        again.")
        scenario.questioner.Ask("Press Enter when you're ready.")
        stsClient := sts.NewFromConfig(*noPermsConfig)

```

```

tempCredentials, err := stsClient.AssumeRole(context.TODO(), &sts.AssumeRoleInput{
    RoleArn:           role.Arn,
    RoleSessionName:  aws.String("AssumeRoleExampleSession"),
    DurationSeconds:  aws.Int32(900),
})
if err != nil {
    log.Printf("Couldn't assume role %v.\n", *role.RoleName)
    panic(err)
}
log.Printf("Assumed role %v, got temporary credentials.\n", *role.RoleName)
assumeRoleConfig, err := config.LoadDefaultConfig(context.TODO(),
    config.WithCredentialsProvider(credentials.NewStaticCredentialsProvider(
        *tempCredentials.Credentials.AccessKeyId,
        *tempCredentials.Credentials.SecretAccessKey,
        *tempCredentials.Credentials.SessionToken),
    ),
)
if err != nil {panic(err)}

// Add test options if this is a test run. This is needed only for testing
purposes.
scenario.addTestOptions(&assumeRoleConfig)

s3Client := s3.NewFromConfig(assumeRoleConfig)
result, err := s3Client.ListBuckets(context.TODO(), &s3.ListBucketsInput{})
if err != nil {
    log.Println("Couldn't list buckets with assumed role credentials.")
    panic(err)
}
log.Println("Successfully called ListBuckets with assumed role credentials, \n" +
    "here are some of them:")
for i := 0; i < len(result.Buckets) && i < 5; i++ {
    log.Printf("\t%v\n", *result.Buckets[i].Name)
}
log.Println(strings.Repeat("-", 88))
}

// Cleanup deletes all resources created for the scenario.
func (scenario AssumeRoleScenario) Cleanup(user *types.User, role *types.Role) {
    if scenario.questioner.AskBool(
        "Do you want to delete the resources created for this example? (y/n)", "y",
    ) {
        policies, err := scenario.roleWrapper.ListAttachedRolePolicies(*role.RoleName)
        if err != nil {panic(err)}
        for _, policy := range policies {
            err = scenario.roleWrapper.DetachRolePolicy(*role.RoleName, *policy.PolicyArn)
            if err != nil {panic(err)}
            err = scenario.policyWrapper.DeletePolicy(*policy.PolicyArn)
            if err != nil {panic(err)}
            log.Printf("Detached policy %v from role %v and deleted the policy.\n",
                *policy.PolicyName, *role.RoleName)
        }
        err = scenario.roleWrapper.DeleteRole(*role.RoleName)
        if err != nil {panic(err)}
        log.Printf("Deleted role %v.\n", *role.RoleName)

        userPols, err := scenario.userWrapper.ListUserPolicies(*user.UserName)
        if err != nil {panic(err)}
        for _, userPol := range userPols {
            err = scenario.userWrapper.DeleteUserPolicy(*user.UserName, userPol)
            if err != nil {panic(err)}
            log.Printf("Deleted policy %v from user %v.\n", userPol, *user.UserName)
        }
        keys, err := scenario.userWrapper.ListAccessKeys(*user.UserName)
        if err != nil {panic(err)}
        for _, key := range keys {

```

```
    err = scenario.userWrapper.DeleteAccessKey(*user.UserName, *key.AccessKeyId)
    if err != nil {panic(err)}
    log.Printf("Deleted access key %v from user %v.\n", *key.AccessKeyId,
    *user.UserName)
}
err = scenario.userWrapper.DeleteUser(*user.UserName)
if err != nil {panic(err)}
log.Printf("Deleted user %v.\n", *user.UserName)
log.Println(strings.Repeat("-", 88))
}
```

Define a struct that wraps account actions.

```
// AccountWrapper encapsulates AWS Identity and Access Management (IAM) account
// actions
// used in the examples.
// It contains an IAM service client that is used to perform account actions.
type AccountWrapper struct {
    IamClient *iam.Client
}

// GetAccountPasswordPolicy gets the account password policy for the current
// account.
// If no policy has been set, a NoSuchEntityException is error is returned.
func (wrapper AccountWrapper) GetAccountPasswordPolicy() (*types.PasswordPolicy,
error) {
var pwPolicy *types.PasswordPolicy
result, err := wrapper.IamClient.GetAccountPasswordPolicy(context.TODO(),
&iam.GetAccountPasswordPolicyInput{})
if err != nil {
    log.Printf("Couldn't get account password policy. Here's why: %v\n", err)
} else {
    pwPolicy = result.PasswordPolicy
}
return pwPolicy, err
}

// ListSAMLProviders gets the SAML providers for the account.
func (wrapper AccountWrapper) ListSAMLProviders() ([]types.SAMLProviderListEntry,
error) {
var providers []types.SAMLProviderListEntry
result, err := wrapper.IamClient.ListSAMLProviders(context.TODO(),
&iam.ListSAMLProvidersInput{})
if err != nil {
    log.Printf("Couldn't list SAML providers. Here's why: %v\n", err)
} else {
    providers = result.SAMLProviderList
}
return providers, err
}
```

Define a struct that wraps policy actions.

```
// PolicyDocument defines a policy document as a Go struct that can be serialized
// to JSON.
type PolicyDocument struct {
    Version string
    Statement []PolicyStatement
}

// PolicyStatement defines a statement in a policy document.
type PolicyStatement struct {
    Effect string
    Action []string
    Principal map[string]string `json:"",omitempty"`
    Resource *string `json:"",omitempty"`
}

// PolicyWrapper encapsulates AWS Identity and Access Management (IAM) policy
// actions
// used in the examples.
// It contains an IAM service client that is used to perform policy actions.
type PolicyWrapper struct {
    IamClient *iam.Client
}

// ListPolicies gets up to maxPolicies policies.
func (wrapper PolicyWrapper) ListPolicies(maxPolicies int32) ([]types.Policy,
    error) {
    var policies []types.Policy
    result, err := wrapper.IamClient.ListPolicies(context.TODO(),
        &iام.ListPoliciesInput{
            MaxItems: aws.Int32(maxPolicies),
        })
    if err != nil {
        log.Printf("Couldn't list policies. Here's why: %v\n", err)
    } else {
        policies = result.Policies
    }
    return policies, err
}

// CreatePolicy creates a policy that grants a list of actions to the specified
// resource.
// PolicyDocument shows how to work with a policy document as a data structure and
// serialize it to JSON by using Go's JSON marshaler.
func (wrapper PolicyWrapper) CreatePolicy(policyName string, actions []string,
    resourceArn string) (*types.Policy, error) {
    var policy *types.Policy
    policyDoc := PolicyDocument{
        Version: "2012-10-17",
        Statement: []PolicyStatement{
            {
                Effect: "Allow",
                Action: actions,
                Resource: aws.String(resourceArn),
            },
        },
    }
    policyBytes, err := json.Marshal(policyDoc)
    if err != nil {
        log.Printf("Couldn't create policy document for %v. Here's why: %v\n",
            resourceArn, err)
    }
}
```

```

        return nil, err
    }
    result, err := wrapper.IamClient.CreatePolicy(context.TODO(),
&iam.CreatePolicyInput{
    PolicyDocument: aws.String(string(policyBytes)),
    PolicyName:     aws.String(policyName),
})
if err != nil {
    log.Printf("Couldn't create policy %v. Here's why: %v\n", policyName, err)
} else {
    policy = result.Policy
}
return policy, err
}

// GetPolicy gets data about a policy.
func (wrapper PolicyWrapper) GetPolicy(policyArn string) (*types.Policy, error) {
var policy *types.Policy
result, err := wrapper.IamClient.GetPolicy(context.TODO(), &iam.GetPolicyInput{
    PolicyArn: aws.String(policyArn),
})
if err != nil {
    log.Printf("Couldn't get policy %v. Here's why: %v\n", policyArn, err)
} else {
    policy = result.Policy
}
return policy, err
}

// DeletePolicy deletes a policy.
func (wrapper PolicyWrapper) DeletePolicy(policyArn string) error {
_, err := wrapper.IamClient.DeletePolicy(context.TODO(), &iam.DeletePolicyInput{
    PolicyArn: aws.String(policyArn),
})
if err != nil {
    log.Printf("Couldn't delete policy %v. Here's why: %v\n", policyArn, err)
}
return err
}

```

Define a struct that wraps role actions.

```

// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
// It contains an IAM service client that is used to perform role actions.
type RoleWrapper struct {
    IamClient *iam.Client
}

// ListRoles gets up to maxRoles roles.
func (wrapper RoleWrapper) ListRoles(maxRoles int32) ([]types.Role, error) {
var roles []types.Role
result, err := wrapper.IamClient.ListRoles(context.TODO(),
    &iam.ListRolesInput{MaxItems: aws.Int32(maxRoles)},
)
if err != nil {

```

```

        log.Printf("Couldn't list roles. Here's why: %v\n", err)
    } else {
        roles = result.Roles
    }
    return roles, err
}

// CreateRole creates a role that trusts a specified user. The trusted user can
// assume
// the role to acquire its permissions.
// PolicyDocument shows how to work with a policy document as a data structure and
// serialize it to JSON by using Go's JSON marshaler.
func (wrapper RoleWrapper) CreateRole(roleName string, trustedUserArn string) (*types.Role, error) {
    var role *types.Role
    trustPolicy := PolicyDocument{
        Version: "2012-10-17",
        Statement: []PolicyStatement{
            Effect: "Allow",
            Principal: map[string]string{"AWS": trustedUserArn},
            Action: []string{"sts:AssumeRole"},
        },
    }
    policyBytes, err := json.Marshal(trustPolicy)
    if err != nil {
        log.Printf("Couldn't create trust policy for %v. Here's why: %v\n",
        trustedUserArn, err)
        return nil, err
    }
    result, err := wrapper.IamClient.CreateRole(context.TODO(), &iam.CreateRoleInput{
        AssumeRolePolicyDocument: aws.String(string(policyBytes)),
        RoleName:                 aws.String(roleName),
    })
    if err != nil {
        log.Printf("Couldn't create role %v. Here's why: %v\n", roleName, err)
    } else {
        role = result.Role
    }
    return role, err
}

// GetRole gets data about a role.
func (wrapper RoleWrapper) GetRole(roleName string) (*types.Role, error) {
    var role *types.Role
    result, err := wrapper.IamClient.GetRole(context.TODO(),
        &iam.GetRoleInput{RoleName: aws.String(roleName)})
    if err != nil {
        log.Printf("Couldn't get role %v. Here's why: %v\n", roleName, err)
    } else {
        role = result.Role
    }
    return role, err
}

// CreateServiceLinkedRole creates a service-linked role that is owned by the
// specified service.
func (wrapper RoleWrapper) CreateServiceLinkedRole(serviceName string, description
    string) (*types.Role, error) {
    var role *types.Role

```

```
result, err := wrapper.IamClient.CreateServiceLinkedRole(context.TODO(),
&iam.CreateServiceLinkedRoleInput{
    AWSServiceName: aws.String(serviceName),
    Description:   aws.String(description),
})
if err != nil {
    log.Printf("Couldn't create service-linked role %v. Here's why: %v\n",
    serviceName, err)
} else {
    role = result.Role
}
return role, err
}

// DeleteServiceLinkedRole deletes a service-linked role.
func (wrapper RoleWrapper) DeleteServiceLinkedRole(roleName string) error {
_, err := wrapper.IamClient.DeleteServiceLinkedRole(context.TODO(),
&iam.DeleteServiceLinkedRoleInput{
    RoleName: aws.String(roleName)},
)
if err != nil {
    log.Printf("Couldn't delete service-linked role %v. Here's why: %v\n", roleName,
err)
}
return err
}

// AttachRolePolicy attaches a policy to a role.
func (wrapper RoleWrapper) AttachRolePolicy(policyArn string, roleName string)
error {
_, err := wrapper.IamClient.AttachRolePolicy(context.TODO(),
&iam.AttachRolePolicyInput{
    PolicyArn: aws.String(policyArn),
    RoleName:  aws.String(roleName),
})
if err != nil {
    log.Printf("Couldn't attach policy %v to role %v. Here's why: %v\n", policyArn,
roleName, err)
}
return err
}

// ListAttachedRolePolicies lists the policies that are attached to the specified
role.
func (wrapper RoleWrapper) ListAttachedRolePolicies(roleName string)
([]types.AttachedPolicy, error) {
var policies []types.AttachedPolicy
result, err := wrapper.IamClient.ListAttachedRolePolicies(context.TODO(),
&iam.ListAttachedRolePoliciesInput{
    RoleName: aws.String(roleName),
})
if err != nil {
    log.Printf("Couldn't list attached policies for role %v. Here's why: %v\n",
roleName, err)
} else {
    policies = result.AttachedPolicies
}
return policies, err
}
```

```
// DetachRolePolicy detaches a policy from a role.
func (wrapper RoleWrapper) DetachRolePolicy(roleName string, policyArn string) error {
    _, err := wrapper.IamClient.DetachRolePolicy(context.TODO(),
    &iam.DetachRolePolicyInput{
        PolicyArn: aws.String(policyArn),
        RoleName:  aws.String(roleName),
    })
    if err != nil {
        log.Printf("Couldn't detach policy from role %v. Here's why: %v\n", roleName, err)
    }
    return err
}

// ListRolePolicies lists the inline policies for a role.
func (wrapper RoleWrapper) ListRolePolicies(roleName string) ([]string, error) {
    var policies []string
    result, err := wrapper.IamClient.ListRolePolicies(context.TODO(),
    &iam.ListRolePoliciesInput{
        RoleName: aws.String(roleName),
    })
    if err != nil {
        log.Printf("Couldn't list policies for role %v. Here's why: %v\n", roleName, err)
    } else {
        policies = result.PolicyNames
    }
    return policies, err
}

// DeleteRole deletes a role. All attached policies must be detached before a
// role can be deleted.
func (wrapper RoleWrapper) DeleteRole(roleName string) error {
    _, err := wrapper.IamClient.DeleteRole(context.TODO(), &iam.DeleteRoleInput{
        RoleName: aws.String(roleName),
    })
    if err != nil {
        log.Printf("Couldn't delete role %v. Here's why: %v\n", roleName, err)
    }
    return err
}
```

Define a struct that wraps user actions.

```
// UserWrapper encapsulates user actions used in the examples.
// It contains an IAM service client that is used to perform user actions.
type UserWrapper struct {
    IamClient *iam.Client
}

// ListUsers gets up to maxUsers number of users.
func (wrapper UserWrapper) ListUsers(maxUsers int32) ([]types.User, error) {
    var users []types.User
```

```
result, err := wrapper.IamClient.ListUsers(context.TODO(), &iam.ListUsersInput{
    MaxItems: aws.Int32(maxUsers),
})
if err != nil {
    log.Printf("Couldn't list users. Here's why: %v\n", err)
} else {
    users = result.Users
}
return users, err
}

// GetUser gets data about a user.
func (wrapper UserWrapper) GetUser(userName string) (*types.User, error) {
    var user *types.User
    result, err := wrapper.IamClient.GetUser(context.TODO(), &iam.GetUserInput{
        UserName: aws.String(userName),
    })
    if err != nil {
        var apiError smithy.APIError
        if errors.As(err, &apiError) {
            switch apiError.(type) {
            case *types.NoSuchEntityException:
                log.Printf("User %v does not exist.\n", userName)
                err = nil
            default:
                log.Printf("Couldn't get user %v. Here's why: %v\n", userName, err)
            }
        }
    } else {
        user = result.User
    }
    return user, err
}

// CreateUser creates a new user with the specified name.
func (wrapper UserWrapper) CreateUser(userName string) (*types.User, error) {
    var user *types.User
    result, err := wrapper.IamClient.CreateUser(context.TODO(), &iam.CreateUserInput{
        UserName: aws.String(userName),
    })
    if err != nil {
        log.Printf("Couldn't create user %v. Here's why: %v\n", userName, err)
    } else {
        user = result.User
    }
    return user, err
}

// CreateUserPolicy adds an inline policy to a user. This example creates a policy
// that
// grants a list of actions on a specified role.
// PolicyDocument shows how to work with a policy document as a data structure and
// serialize it to JSON by using Go's JSON marshaler.
func (wrapper UserWrapper) CreateUserPolicy(userName string, policyName string,
    actions []string,
    roleArn string) error {
    policyDoc := PolicyDocument{
        Version: "2012-10-17",
        Statement: []PolicyStatement{
            Effect: "Allow",

```

```

        Action: actions,
        Resource: aws.String(roleArn),
    },
}
policyBytes, err := json.Marshal(policyDoc)
if err != nil {
    log.Printf("Couldn't create policy document for %v. Here's why: %v\n", roleArn,
err)
    return err
}
-, err = wrapper.IamClient.PutUserPolicy(context.TODO(), &iam.PutUserPolicyInput{
    PolicyDocument: aws.String(string(policyBytes)),
    PolicyName:     aws.String(policyName),
    UserName:       aws.String(userName),
})
if err != nil {
    log.Printf("Couldn't create policy for user %v. Here's why: %v\n", userName, err)
}
return err
}

// ListUserPolicies lists the inline policies for the specified user.
func (wrapper UserWrapper) ListUserPolicies(userName string) ([]string, error) {
var policies []string
result, err := wrapper.IamClient.ListUserPolicies(context.TODO(),
&iam.ListUserPoliciesInput{
    UserName: aws.String(userName),
})
if err != nil {
    log.Printf("Couldn't list policies for user %v. Here's why: %v\n", userName, err)
} else {
    policies = result.PolicyNames
}
return policies, err
}

// DeleteUserPolicy deletes an inline policy from a user.
func (wrapper UserWrapper) DeleteUserPolicy(userName string, policyName string) error {
error {
-, err := wrapper.IamClient.DeleteUserPolicy(context.TODO(),
&iam.DeleteUserPolicyInput{
    PolicyName: aws.String(policyName),
    UserName:   aws.String(userName),
})
if err != nil {
    log.Printf("Couldn't delete policy from user %v. Here's why: %v\n", userName,
err)
}
return err
}

// DeleteUser deletes a user.
func (wrapper UserWrapper) DeleteUser(userName string) error {
-, err := wrapper.IamClient.DeleteUser(context.TODO(), &iam.DeleteUserInput{
    UserName: aws.String(userName),
})
if err != nil {
    log.Printf("Couldn't delete user %v. Here's why: %v\n", userName, err)
}
return err
}

```

```

}

// CreateAccessKeyPair creates an access key for a user. The returned access key
// contains
// the ID and secret credentials needed to use the key.
func (wrapper UserWrapper) CreateAccessKeyPair(userName string) (*types.AccessKey,
    error) {
    var key *types.AccessKey
    result, err := wrapper.IamClient.CreateAccessKey(context.TODO(),
    &iam.CreateAccessKeyInput{
        UserName: aws.String(userName)})
    if err != nil {
        log.Printf("Couldn't create access key pair for user %v. Here's why: %v\n",
        userName, err)
    } else {
        key = result.AccessKey
    }
    return key, err
}

// DeleteAccessKey deletes an access key from a user.
func (wrapper UserWrapper) DeleteAccessKey(userName string, keyId string) error {
    _, err := wrapper.IamClient.DeleteAccessKey(context.TODO(),
    &iam.DeleteAccessKeyInput{
        AccessKeyId: aws.String(keyId),
        UserName:    aws.String(userName),
    })
    if err != nil {
        log.Printf("Couldn't delete access key %v. Here's why: %v\n", keyId, err)
    }
    return err
}

// ListAccessKeys lists the access keys for the specified user.
func (wrapper UserWrapper) ListAccessKeys(userName string)
    ([]types.AccessKeyMetadata, error) {
    var keys []types.AccessKeyMetadata
    result, err := wrapper.IamClient.ListAccessKeys(context.TODO(),
    &iam.ListAccessKeysInput{
        UserName: aws.String(userName),
    })
    if err != nil {
        log.Printf("Couldn't list access keys for user %v. Here's why: %v\n", userName,
        err)
    } else {
        keys = result.AccessKeyMetadata
    }
    return keys, err
}

```

- For API details, see the following topics in *AWS SDK for Go API Reference*.
 - [AttachRolePolicy](#)
 - [CreateAccessKey](#)
 - [CreatePolicy](#)
 - [CreateRole](#)

- [CreateUser](#)
- [DeleteAccessKey](#)
- [DeletePolicy](#)
- [DeleteRole](#)
- [DeleteUser](#)
- [DeleteUserPolicy](#)
- [DetachRolePolicy](#)
- [PutUserPolicy](#)

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Create functions that wrap IAM user actions.

```
/*
 To run this Java V2 code example, set up your development environment, including
 your credentials.

 For information, see this documentation topic:

 https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html

 This example performs these operations:

 1. Creates a user that has no permissions.
 2. Creates a role and policy that grants Amazon S3 permissions.
 3. Creates a role.
 4. Grants the user permissions.
 5. Gets temporary credentials by assuming the role. Creates an Amazon S3 Service
 client object with the temporary credentials.
 6. Deletes the resources.
 */

public class IAMScenario {
    public static final String DASHES = new String(new char[80]).replace("\0",
    "-");
    public static final String PolicyDocument =
        "{" +
            "  \"Version\": \"2012-10-17\", " +
            "  \"Statement\": [" +
                "    {" +
                    "      \"Effect\": \"Allow\", " +
                    "      \"Action\": [" +
                        "        \"s3:*\" " +
                    "      ], " +
                    "      \"Resource\": \"*\" " +
                "    }" +
            "  ]" +
        "}";
    public static String userArn;
    public static void main(String[] args) throws Exception {

        final String usage = "\n" +
            "Usage:\n" +
```

```

\n\n" +
    "      <username> <policyName> <roleName> <roleSessionName> <bucketName>
"Where:\n" +
    "      username - The name of the IAM user to create. \n\n" +
    "      policyName - The name of the policy to create. \n\n" +
    "      roleName - The name of the role to create. \n\n" +
    "      roleSessionName - The name of the session required for the
assumeRole operation. \n\n" +
    "      bucketName - The name of the Amazon S3 bucket from which objects
are read. \n\n";
if (args.length != 5) {
    System.out.println(usage);
    System.exit(1);
}

String userName = args[0];
String policyName = args[1];
String roleName = args[2];
String roleSessionName = args[3];
String bucketName = args[4];

Region region = Region.AWS_GLOBAL;
IamClient iam = IamClient.builder()
    .region(region)
    .credentialsProvider(ProfileCredentialsProvider.create())
    .build();

System.out.println(DASHES);
System.out.println("Welcome to the AWS IAM example scenario.");
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println(" 1. Create the IAM user.");
User createUser = createIAMUser(iam, userName);

System.out.println(DASHES);
userArn = createUser.arn();

AccessKey myKey = createIAMAccessKey(iam, userName);
String accessKey = myKey.accessKeyId();
String secretKey = myKey.secretAccessKey();
String assumeRolePolicyDocument = "{" +
    "\\"Version\\": \\"2012-10-17\\", " +
    "\\"Statement\\": [{" +
        "\\"Effect\\": \\"Allow\\", " +
        "\\"Principal\\": {" +
            " \\"AWS\\": \" + userArn + "\" +
        "}, " +
        "\\"Action\\": \\"sts:AssumeRole\\\" +
    "}]" +
"}";

System.out.println(assumeRolePolicyDocument);
System.out.println(userName + " was successfully created.");
System.out.println(DASHES);
System.out.println("2. Creates a policy.");
String polArn = createIAMPolicy(iam, policyName);
System.out.println("The policy " + polArn + " was successfully created.");
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("3. Creates a role.");
TimeUnit.SECONDS.sleep(30);
String roleArn = createIAMRole(iam, roleName, assumeRolePolicyDocument);
System.out.println(roleArn + " was successfully created.");

```

```

        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("4. Grants the user permissions.");
        attachIAMRolePolicy(iam, roleName, polArn);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("*** Wait for 30 secs so the resource is available");
        TimeUnit.SECONDS.sleep(30);
        System.out.println("5. Gets temporary credentials by assuming the role.");
        System.out.println("Perform an Amazon S3 Service operation using the
temporary credentials.");
        assumeRole(roleArn, roleSessionName, bucketName, accessKey, secretKey);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("6 Getting ready to delete the AWS resources");
        deleteKey(iam, userName, accessKey );
        deleteRole(iam, roleName, polArn);
        deleteIAMUser(iam, userName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("This IAM Scenario has successfully completed");
        System.out.println(DASHES);
    }

    public static AccessKey createIAMAccessKey(IamClient iam, String user) {
        try {
            CreateAccessKeyRequest request = CreateAccessKeyRequest.builder()
                .userName(user)
                .build();

            CreateAccessKeyResponse response = iam.createAccessKey(request);
            return response.accessKey();

        } catch (IamException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return null;
    }

    public static User createIAMUser(IamClient iam, String username) {
        try {
            // Create an IamWaiter object
            IamWaiter iamWaiter = iam.waiter();
            CreateUserRequest request = CreateUserRequest.builder()
                .userName(username)
                .build();

            // Wait until the user is created.
            CreateUserResponse response = iam.createUser(request);
            GetUserRequest userRequest = GetUserRequest.builder()
                .userName(response.user().userName())
                .build();

            WaiterResponse< GetUserResponse> waitUntilUserExists =
iamWaiter.waitUntilUserExists(userRequest);

            waitUntilUserExists.matched().response().ifPresent(System.out::println);
            return response.user();

        } catch (IamException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
        }
    }
}

```

```

        System.exit(1);
    }
    return null;
}

public static String createIAMRole(IamClient iam, String rolename, String json)
{
    try {
        CreateRoleRequest request = CreateRoleRequest.builder()
            .roleName(rolename)
            .assumeRolePolicyDocument(json)
            .description("Created using the AWS SDK for Java")
            .build();

        CreateRoleResponse response = iam.createRole(request);
        System.out.println("The ARN of the role is " + response.role().arn());
        return response.role().arn();
    } catch (IamException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
    return "";
}

public static String createIAMPolicy(IamClient iam, String policyName) {
    try {
        // Create an IamWaiter object.
        IamWaiter iamWaiter = iam.waiter();
        CreatePolicyRequest request = CreatePolicyRequest.builder()
            .policyName(policyName)
            .policyDocument(PolicyDocument).build();

        CreatePolicyResponse response = iam.createPolicy(request);
        GetPolicyRequest polRequest = GetPolicyRequest.builder()
            .policyArn(response.policy().arn())
            .build();

        WaiterResponse<GetPolicyResponse> waitUntilPolicyExists =
            iamWaiter.waitUntilPolicyExists(polRequest);

        waitUntilPolicyExists.matched().response().ifPresent(System.out::println);
        return response.policy().arn();
    } catch (IamException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
    return "";
}

public static void attachIAMRolePolicy(IamClient iam, String roleName, String
policyArn) {
    try {
        ListAttachedRolePoliciesRequest request =
ListAttachedRolePoliciesRequest.builder()
            .roleName(roleName)
            .build();

        ListAttachedRolePoliciesResponse response =
iam.listAttachedRolePolicies(request);
        List<AttachedPolicy> attachedPolicies = response.attachedPolicies();
        String polArn;
        for (AttachedPolicy policy : attachedPolicies) {
            polArn = policy.policyArn();
    }
}
}

```

```

        if (polArn.compareTo(policyArn) == 0) {
            System.out.println(roleName + " policy is already attached to
this role.");
            return;
        }
    }

AttachRolePolicyRequest attachRequest =
AttachRolePolicyRequest.builder()
    .roleName(roleName)
    .policyArn(policyArn)
    .build();

iam.attachRolePolicy(attachRequest);
System.out.println("Successfully attached policy " + policyArn + " to
role " + roleName);

} catch (IamException e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}
}

// Invoke an Amazon S3 operation using the Assumed Role.
public static void assumeRole(String roleArn, String roleSessionName, String
bucketName, String keyVal, String keySecret) {

    // Use the creds of the new IAM user that was created in this code example.
    AwsBasicCredentials credentials = AwsBasicCredentials.create(keyVal,
keySecret);
    StsClient stsClient = StsClient.builder()
        .region(Region.US_EAST_1)
        .credentialsProvider(StaticCredentialsProvider.create(credentials))
        .build();

    try {
        AssumeRoleRequest roleRequest = AssumeRoleRequest.builder()
            .roleArn(roleArn)
            .roleSessionName(roleSessionName)
            .build();

        AssumeRoleResponse roleResponse = stsClient.assumeRole(roleRequest);
        Credentials myCreds = roleResponse.credentials();
        String key = myCreds.accessKeyId();
        String secKey = myCreds.secretAccessKey();
        String secToken = myCreds.sessionToken();

        // List all objects in an Amazon S3 bucket using the temp creds
retrieved by invoking assumeRole.
        Region region = Region.US_EAST_1;
        S3Client s3 = S3Client.builder()

.credentialsProvider(StaticCredentialsProvider.create(AwsSessionCredentials.create(key,
secKey, secToken)))
        .region(region)
        .build();

        System.out.println("Created a S3Client using temp credentials.");
        System.out.println("Listing objects in " + bucketName);
        ListObjectsRequest listObjects = ListObjectsRequest.builder()
            .bucket(bucketName)
            .build();

        ListObjectsResponse res = s3.listObjects(listObjects);
        List<S3Object> objects = res.contents();
        for (S3Object myValue : objects) {

```

```

        System.out.println("The name of the key is " + myValue.key());
        System.out.println("The owner is " + myValue.owner());
    }

} catch (StsException e) {
    System.err.println(e.getMessage());
    System.exit(1);
}
}

public static void deleteRole(IamClient iam, String roleName, String polArn) {

    try {
        // First the policy needs to be detached.
        DetachRolePolicyRequest rolePolicyRequest =
DetachRolePolicyRequest.builder()
    .policyArn(polArn)
    .roleName(roleName)
    .build();

        iam.detachRolePolicy(rolePolicyRequest);

        // Delete the policy.
        DeletePolicyRequest request = DeletePolicyRequest.builder()
            .policyArn(polArn)
            .build();

        iam.deletePolicy(request);
        System.out.println("*** Successfully deleted " + polArn);

        // Delete the role.
        DeleteRoleRequest roleRequest = DeleteRoleRequest.builder()
            .roleName(roleName)
            .build();

        iam.deleteRole(roleRequest);
        System.out.println("*** Successfully deleted " + roleName);

    } catch (IamException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}

public static void deleteKey(IamClient iam ,String username, String accessKey )
{
    try {
        DeleteAccessKeyRequest request = DeleteAccessKeyRequest.builder()
            .accessKeyId(accessKey)
            .userName(username)
            .build();

        iam.deleteAccessKey(request);
        System.out.println("Successfully deleted access key " + accessKey +
            " from user " + username);

    } catch (IamException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}

public static void deleteIAMUser(IamClient iam, String userName) {
    try {
        DeleteUserRequest request = DeleteUserRequest.builder()
            .userName(userName)
            .build();
    }
}

```

```
        iam.deleteUser(request);
        System.out.println("**** Successfully deleted " + userName);

    } catch (IamException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

- For API details, see the following topics in *AWS SDK for Java 2.x API Reference*.

- [AttachRolePolicy](#)
- [CreateAccessKey](#)
- [CreatePolicy](#)
- [CreateRole](#)
- [CreateUser](#)
- [DeleteAccessKey](#)
- [DeletePolicy](#)
- [DeleteRole](#)
- [DeleteUser](#)
- [DeleteUserPolicy](#)
- [DetachRolePolicy](#)
- [PutUserPolicy](#)

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Create an IAM user and a role that grants permission to list Amazon S3 buckets. The user has rights only to assume the role. After assuming the role, use temporary credentials to list buckets for the account.

```
import {
  CreateUserCommand,
  CreateAccessKeyCommand,
  CreatePolicyCommand,
  CreateRoleCommand,
  AttachRolePolicyCommand,
  DeleteAccessKeyCommand,
  DeleteUserCommand,
  DeleteRoleCommand,
  DeletePolicyCommand,
  DetachRolePolicyCommand,
  IAMClient,
} from "@aws-sdk/client-iam";
import { ListBucketsCommand, S3Client } from "@aws-sdk/client-s3";
import { AssumeRoleCommand, STSClient } from "@aws-sdk/client-sts";
import { retry } from "libs/utils/util-timers.js";

// Set the parameters.
const iamClient = new IAMClient({});
const(userName = "test_name";
```

```

const policyName = "test_policy";
const roleName = "test_role";

export const main = async () => {
    // Create a user. The user has no permissions by default.
    const { User } = await iamClient.send(
        new CreateUserCommand({ UserName: userName })
    );

    if (!User) {
        throw new Error("User not created");
    }

    // Create an access key. This key is used to authenticate the new user to
    // Amazon Simple Storage Service (Amazon S3) and AWS Security Token Service (AWS
    // STS).
    // It's not best practice to use access keys. For more information, see https://aws.amazon.com/iam/resources/best-practices/.
    const createAccessKeyResponse = await iamClient.send(
        new CreateAccessKeyCommand({ UserName: userName })
    );

    if (
        !createAccessKeyResponse.AccessKey?.AccessKeyId ||
        !createAccessKeyResponse.AccessKey?.SecretAccessKey
    ) {
        throw new Error("Access key not created");
    }

    const {
        AccessKey: { AccessKeyId, SecretAccessKey },
    } = createAccessKeyResponse;

    let s3Client = new S3Client({
        credentials: {
            accessKeyId: AccessKeyId,
            secretAccessKey: SecretAccessKey,
        },
    });

    // Retry the list buckets operation until it succeeds. InvalidAccessKeyId is
    // thrown while the user and access keys are still stabilizing.
    await retry({ intervalInMs: 1000, maxRetries: 300 }, async () => {
        try {
            return await listBuckets(s3Client);
        } catch (err) {
            if (err instanceof Error && err.name === "InvalidAccessKeyId") {
                throw err;
            }
        }
    });
}

// Retry the create role operation until it succeeds. A MalformedPolicyDocument
// error
// is thrown while the user and access keys are still stabilizing.
const { Role } = await retry(
    {
        intervalInMs: 2000,
        maxRetries: 60,
    },
    () =>
        iamClient.send(
            new CreateRoleCommand({
                AssumeRolePolicyDocument: JSON.stringify({
                    Version: "2012-10-17",
                    Statement: [

```

```

    {
      Effect: "Allow",
      Principal: {
        // Allow the previously created user to assume this role.
        AWS: User.Arn,
      },
      Action: "sts:AssumeRole",
    },
  ],
}),
RoleName: roleName,
})
)
);

if (!Role) {
  throw new Error("Role not created");
}

// Create a policy that allows the user to list S3 buckets.
const { Policy: listBucketPolicy } = await iamClient.send(
  new CreatePolicyCommand({
    PolicyDocument: JSON.stringify({
      Version: "2012-10-17",
      Statement: [
        {
          Effect: "Allow",
          Action: ["s3>ListAllMyBuckets"],
          Resource: "*",
        },
      ],
    }),
    PolicyName: policyName,
  })
);

if (!listBucketPolicy) {
  throw new Error("Policy not created");
}

// Attach the policy granting the 's3>ListAllMyBuckets' action to the role.
await iamClient.send(
  new AttachRolePolicyCommand({
    PolicyArn: listBucketPolicy.Arn,
    RoleName: Role.RoleName,
  })
);

// Assume the role.
const stsClient = new STSClient({
  credentials: {
    accessKeyId: AccessKeyId,
    secretAccessKey: SecretAccessKey,
  },
});

// Retry the assume role operation until it succeeds.
const { Credentials } = await retry(
  { intervalInMs: 2000, maxRetries: 60 },
  () =>
    stsClient.send(
      new AssumeRoleCommand({
        RoleArn: Role.Arn,
        RoleSessionName: `iamBasicScenarioSession-${Math.floor(
          Math.random() * 1000000
        )}`,
      })
    )
);

```

```

        DurationSeconds: 900,
    })
)
);

if (!Credentials?.AccessKeyId || !Credentials?.SecretAccessKey) {
    throw new Error("Credentials not created");
}

s3Client = new S3Client({
    credentials: {
        accessKeyId: Credentials.AccessKeyId,
        secretAccessKey: Credentials.SecretAccessKey,
        sessionToken: Credentials.SessionToken,
    },
});

// List the S3 buckets again.
// Retry the list buckets operation until it succeeds. AccessDenied might
// be thrown while the role policy is still stabilizing.
await retry({ intervalInMs: 2000, maxRetries: 60 }, () =>
    listBuckets(s3Client)
);

// Clean up.
await iamClient.send(
    new DetachRolePolicyCommand({
        PolicyArn: listBucketPolicy.Arn,
        RoleName: Role.RoleName,
    })
);

await iamClient.send(
    new DeletePolicyCommand({
        PolicyArn: listBucketPolicy.Arn,
    })
);

await iamClient.send(
    new DeleteRoleCommand({
        RoleName: Role.RoleName,
    })
);

await iamClient.send(
    new DeleteAccessKeyCommand({
        UserName: userName,
        AccessKeyId,
    })
);

await iamClient.send(
    new DeleteUserCommand({
        UserName: userName,
    })
);
);

/***
 *
 * @param {S3Client} s3Client
 */
const listBuckets = async (s3Client) => {
    const { Buckets } = await s3Client.send(new ListBucketsCommand([]));

    if (!Buckets) {

```

```
        throw new Error("Buckets not listed");
    }

    console.log(Buckets.map((bucket) => bucket.Name).join("\n"));
};
```

- For API details, see the following topics in *AWS SDK for JavaScript API Reference*.
 - [AttachRolePolicy](#)
 - [CreateAccessKey](#)
 - [CreatePolicy](#)
 - [CreateRole](#)
 - [CreateUser](#)
 - [DeleteAccessKey](#)
 - [DeletePolicy](#)
 - [DeleteRole](#)
 - [DeleteUser](#)
 - [DeleteUserPolicy](#)
 - [DetachRolePolicy](#)
 - [PutUserPolicy](#)

Kotlin

SDK for Kotlin

Note

This is prerelease documentation for a feature in preview release. It is subject to change.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Create functions that wrap IAM user actions.

```
suspend fun main(args: Array<String>) {

    val usage = """
    Usage:
        <username> <policyName> <roleName> <roleSessionName> <fileLocation>
    <bucketName>

    Where:
        username - The name of the IAM user to create.
        policyName - The name of the policy to create.
        roleName - The name of the role to create.
        roleSessionName - The name of the session required for the assumeRole
operation.
        fileLocation - The file location to the JSON required to create the role
(see Readme).
        bucketName - The name of the Amazon S3 bucket from which objects are read.
    """

    if (args.size != 6) {
        println(usage)
        exitProcess(1)
    }
}
```

```

    val userName = args[0]
    val policyName = args[1]
    val roleName = args[2]
    val roleSessionName = args[3]
    val fileLocation = args[4]
    val bucketName = args[5]

    createUser(userName)
    println("$userName was successfully created.")

    val polArn = createPolicy(policyName)
    println("The policy $polArn was successfully created.")

    val roleArn = createRole(roleName, fileLocation)
    println("$roleArn was successfully created.")
    attachRolePolicy(roleName, polArn)

    println("**** Wait for 1 MIN so the resource is available.")
    delay(60000)
    assumeGivenRole(roleArn, roleSessionName, bucketName)

    println("**** Getting ready to delete the AWS resources.")
    deleteRole(roleName, polArn)
    deleteUser(userName)
    println("This IAM Scenario has successfully completed.")
}

suspend fun createUser(usernameVal: String?): String? {
    val request = CreateUserRequest {
        userName = usernameVal
    }

    IamClient { region = "AWS_GLOBAL" }.use { iamClient ->
        val response = iamClient.createUser(request)
        return response.user?.userName
    }
}

suspend fun createPolicy(policyNameVal: String?): String {
    val policyDocumentValue: String = "{" +
        " \"Version\": \"2012-10-17\", " +
        " \"Statement\": [ " +
        "     { " +
        "         \"Effect\": \"Allow\", " +
        "         \"Action\": [ " +
        "             \"s3:*\" " +
        "         ], " +
        "         \"Resource\": \"*\" " +
        "     } " +
        " ] " +
    "}"

    val request = CreatePolicyRequest {
        policyName = policyNameVal
        policyDocument = policyDocumentValue
    }

    IamClient { region = "AWS_GLOBAL" }.use { iamClient ->
        val response = iamClient.createPolicy(request)
        return response.policy?.arn.toString()
    }
}

```

```

suspend fun createRole(rolenameVal: String?, fileLocation: String?): String? {
    val jsonObject = fileLocation?.let { readJsonSimpleDemo(it) } as JSONObject
    val request = CreateRoleRequest {
        roleName = rolenameVal
        assumeRolePolicyDocument = jsonObject.toJSONString()
        description = "Created using the AWS SDK for Kotlin"
    }
    IamClient { region = "AWS_GLOBAL" }.use { iamClient ->
        val response = iamClient.createRole(request)
        return response.role?.arn
    }
}

suspend fun attachRolePolicy(roleNameVal: String, policyArnVal: String) {
    val request = ListAttachedRolePoliciesRequest {
        roleName = roleNameVal
    }
    IamClient { region = "AWS_GLOBAL" }.use { iamClient ->
        val response = iamClient.listAttachedRolePolicies(request)
        val attachedPolicies = response.attachedPolicies
        // Ensure that the policy is not attached to this role.
        val checkStatus: Int
        if (attachedPolicies != null) {
            checkStatus = checkMyList(attachedPolicies, policyArnVal)
            if (checkStatus == -1)
                return
        }
        val policyRequest = AttachRolePolicyRequest {
            roleName = roleNameVal
            policyArn = policyArnVal
        }
        iamClient.attachRolePolicy(policyRequest)
        println("Successfully attached policy $policyArnVal to role $roleNameVal")
    }
}

fun checkMyList(attachedPolicies: List<AttachedPolicy>, policyArnVal: String): Int {
    for (policy in attachedPolicies) {
        val polArn = policy.policyArn.toString()
        if (polArn.compareTo(policyArnVal) == 0) {
            println("The policy is already attached to this role.")
            return -1
        }
    }
    return 0
}

suspend fun assumeGivenRole(roleArnVal: String?, roleSessionNameVal: String?,
                           bucketName: String?) {
    val stsClient = StsClient {
        region = "us-east-1"
    }
    val roleRequest = AssumeRoleRequest {
        roleArn = roleArnVal
    }
}

```

```

        roleSessionName = roleSessionNameVal
    }

    val roleResponse = stsClient.assumeRole(roleRequest)
    val myCreds = roleResponse.credentials
    val key = myCreds?.accessKeyId
    val secKey = myCreds?.secretAccessKey
    val secToken = myCreds?.sessionToken

    val staticCredentials = StaticCredentialsProvider {
        accessKeyId = key
        secretAccessKey = secKey
        sessionToken = secToken
    }

    // List all objects in an Amazon S3 bucket using the temp creds.
    val s3 = S3Client {
        credentialsProvider = staticCredentials
        region = "us-east-1"
    }

    println("Created a S3Client using temp credentials.")
    println("Listing objects in $bucketName")

    val listObjects = ListObjectsRequest {
        bucket = bucketName
    }

    val response = s3.listObjects(listObjects)
    response.contents?.forEach { myObject ->
        println("The name of the key is ${myObject.key}")
        println("The owner is ${myObject.owner}")
    }
}

suspend fun deleteRole(roleNameVal: String, polArn: String) {

    val iam = IamClient { region = "AWS_GLOBAL" }

    // First the policy needs to be detached.
    val rolePolicyRequest = DetachRolePolicyRequest {
        policyArn = polArn
        roleName = roleNameVal
    }

    iam.detachRolePolicy(rolePolicyRequest)

    // Delete the policy.
    val request = DeletePolicyRequest {
        policyArn = polArn
    }

    iam.deletePolicy(request)
    println("*** Successfully deleted $polArn")

    // Delete the role.
    val roleRequest = DeleteRoleRequest {
        roleName = roleNameVal
    }

    iam.deleteRole(roleRequest)
    println("*** Successfully deleted $roleNameVal")
}

suspend fun deleteUser(userNameVal: String) {
    val iam = IamClient { region = "AWS_GLOBAL" }
}

```

```
    val request = DeleteUserRequest {
        userName = userNameVal
    }

    iam.deleteUser(request)
    println("**** Successfully deleted $userNameVal")
}

@Throws(java.lang.Exception::class)
fun readJsonSimpleDemo(filename: String): Any? {
    val reader = FileReader(filename)
    val jsonParser = JSONParser()
    return jsonParser.parse(reader)
}
```

- For API details, see the following topics in *AWS SDK for Kotlin API reference*.
 - [AttachRolePolicy](#)
 - [CreateAccessKey](#)
 - [CreatePolicy](#)
 - [CreateRole](#)
 - [CreateUser](#)
 - [DeleteAccessKey](#)
 - [DeletePolicy](#)
 - [DeleteRole](#)
 - [DeleteUser](#)
 - [DeleteUserPolicy](#)
 - [DetachRolePolicy](#)
 - [PutUserPolicy](#)

PHP

SDK for PHP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
namespace Iam\Basics;

require 'vendor/autoload.php';

use Aws\Credentials\Credentials;
use Aws\S3\Exception\S3Exception;
use Aws\S3\S3Client;
use Aws\Sts\StsClient;
use Iam\IAMService;

echo("\n");
echo("-----\n");
print("Welcome to the IAM getting started demo using PHP!\n");
echo("-----\n");

$uuid = uniqid();
$service = new IAMService();

$user = $service->createUser("iam_demo_user_$uuid");
echo "Created user with the arn: {$user['Arn']} \n";
```

```

$key = $service->createAccessKey($user['UserName']);
$assumeRolePolicyDocument = "{
    \"Version\": \"2012-10-17\",
    \"Statement\": [
        {
            \"Effect\": \"Allow\",
            \"Principal\": \"$AWS\",
            \"Action\": \"sts:AssumeRole\"
        }
    ]
}";
$assumeRoleRole = $service->createRole("iam_demo_role_$uuid",
    $assumeRolePolicyDocument);
echo "Created role: {$assumeRoleRole['RoleName']}\n";

$listAllBucketsPolicyDocument = "{
    \"Version\": \"2012-10-17\",
    \"Statement\": [
        {
            \"Effect\": \"Allow\",
            \"Action\": \"s3>ListAllMyBuckets\",
            \"Resource\": \"arn:aws:s3:::*\"
        }
];
$listAllBucketsPolicy = $service->createPolicy("iam_demo_policy_$uuid",
    $listAllBucketsPolicyDocument);
echo "Created policy: {$listAllBucketsPolicy['PolicyName']}\n";

(service->attachRolePolicy($assumeRoleRole['RoleName'],
    $listAllBucketsPolicy['Arn']);

$inlinePolicyDocument = "{
    \"Version\": \"2012-10-17\",
    \"Statement\": [
        {
            \"Effect\": \"Allow\",
            \"Action\": \"sts:AssumeRole\",
            \"Resource\": \"{$assumeRoleRole['Arn']}\"}
    ]
}";
$inlinePolicy = $service->createUserPolicy("iam_demo_inline_policy_$uuid",
    $inlinePolicyDocument, $user['UserName']);
//First, fail to list the buckets with the user
$credentials = new Credentials($key['AccessKeyId'], $key['SecretAccessKey']);
$s3Client = new S3Client(['region' => 'us-west-2', 'version' => 'latest',
    'credentials' => $credentials]);
try {
    $s3Client->listBuckets([
    ]);
    echo "this should not run";
} catch (S3Exception $exception) {
    echo "successfully failed!\n";
}

$stsClient = new StsClient(['region' => 'us-west-2', 'version' => 'latest',
    'credentials' => $credentials]);
sleep(10);
$assumedRole = $stsClient->assumeRole([
    'RoleArn' => $assumeRoleRole['Arn'],
    'RoleSessionName' => "DemoAssumeRoleSession_$uuid",
]);
$assumedCredentials = [
    'key' => $assumedRole['Credentials']['AccessKeyId'],
    'secret' => $assumedRole['Credentials']['SecretAccessKey'],
    'token' => $assumedRole['Credentials']['SessionToken'],
];
$s3Client = new S3Client(['region' => 'us-west-2', 'version' => 'latest',
    'credentials' => $assumedCredentials]);
try {
    $s3Client->listBuckets([]);
    echo "this should now run!\n";
}

```

```
    } catch (S3Exception $exception) {
        echo "this should now not fail\n";
    }

    $service->detachRolePolicy($assumeRoleRole['RoleName'],
        $listAllBucketsPolicy['Arn']);
    $deletePolicy = $service->deletePolicy($listAllBucketsPolicy['Arn']);
    echo "Delete policy: {$listAllBucketsPolicy['PolicyName']}\\n";
    $deletedRole = $service->deleteRole($assumeRoleRole['Arn']);
    echo "Deleted role: {$assumeRoleRole['RoleName']}\\n";
    $deletedKey = $service->deleteAccessKey($key['AccessKeyId'], $user['UserName']);
    $deletedUser = $service->deleteUser($user['UserName']);
    echo "Delete user: {$user['UserName']}\\n";
```

- For API details, see the following topics in *AWS SDK for PHP API Reference*.
 - [AttachRolePolicy](#)
 - [CreateAccessKey](#)
 - [CreatePolicy](#)
 - [CreateRole](#)
 - [CreateUser](#)
 - [DeleteAccessKey](#)
 - [DeletePolicy](#)
 - [DeleteRole](#)
 - [DeleteUser](#)
 - [DeleteUserPolicy](#)
 - [DetachRolePolicy](#)
 - [PutUserPolicy](#)

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Create an IAM user and a role that grants permission to list Amazon S3 buckets. The user has rights only to assume the role. After assuming the role, use temporary credentials to list buckets for the account.

```
import json
import sys
import time
from uuid import uuid4

import boto3
from botocore.exceptions import ClientError


def progress_bar(seconds):
    """Shows a simple progress bar in the command window."""
    for _ in range(seconds):
        time.sleep(1)
        print('.', end='')
        sys.stdout.flush()
    print()
```

```

def setup(iam_resource):
    """
    Creates a new user with no permissions.
    Creates an access key pair for the user.
    Creates a role with a policy that lets the user assume the role.
    Creates a policy that allows listing Amazon S3 buckets.
    Attaches the policy to the role.
    Creates an inline policy for the user that lets the user assume the role.

    :param iam_resource: A Boto3 AWS Identity and Access Management (IAM) resource
                         that has permissions to create users, roles, and policies
                         in the account.
    :return: The newly created user, user key, and role.
    """
    try:
        user = iam_resource.create_user(UserName=f'demo-user-{uuid4()}')
        print(f"Created user {user.name}.")
    except ClientError as error:
        print(f"Couldn't create a user for the demo. Here's why: "
              f"{error.response['Error']['Message']}")
        raise

    try:
        user_key = user.create_access_key_pair()
        print(f"Created access key pair for user.")
    except ClientError as error:
        print(f"Couldn't create access keys for user {user.name}. Here's why: "
              f"{error.response['Error']['Message']}")
        raise

    print(f"Wait for user to be ready.", end='')
    progress_bar(10)

    try:
        role = iam_resource.create_role(
            RoleName=f'demo-role-{uuid4()}',
            AssumeRolePolicyDocument=json.dumps({
                'Version': '2012-10-17',
                'Statement': [{
                    'Effect': 'Allow',
                    'Principal': {'AWS': user.arn},
                    'Action': 'sts:AssumeRole']}})
        print(f"Created role {role.name}.")
    except ClientError as error:
        print(f"Couldn't create a role for the demo. Here's why: "
              f"{error.response['Error']['Message']}")
        raise

    try:
        policy = iam_resource.create_policy(
            PolicyName=f'demo-policy-{uuid4()}',
            PolicyDocument=json.dumps({
                'Version': '2012-10-17',
                'Statement': [{
                    'Effect': 'Allow',
                    'Action': 's3>ListAllMyBuckets',
                    'Resource': 'arn:aws:s3:::*']}))
        role.attach_policy(PolicyArn=policy.arn)
        print(f"Created policy {policy.policy_name} and attached it to the role.")
    except ClientError as error:
        print(f"Couldn't create a policy and attach it to role {role.name}. Here's
why: "
              f"{error.response['Error']['Message']}")
        raise

```

```

try:
    user.create_policy(
        PolicyName=f'demo-user-policy-{uuid4()}',
        PolicyDocument=json.dumps({
            'Version': '2012-10-17',
            'Statement': [{
                'Effect': 'Allow',
                'Action': 'sts:AssumeRole',
                'Resource': role.arn}]}))
    print(f"Created an inline policy for {user.name} that lets the user assume
"
          f"the role.")
except ClientError as error:
    print(f"Couldn't create an inline policy for user {user.name}. Here's why:
"
          f"{error.response['Error']['Message']}")
    raise

print("Give AWS time to propagate these new resources and connections.",
end='')
progress_bar(10)

return user, user_key, role

```

```

def show_access_denied_without_role(user_key):
    """
    Shows that listing buckets without first assuming the role is not allowed.

    :param user_key: The key of the user created during setup. This user does not
                     have permission to list buckets in the account.
    """
    print(f"Try to list buckets without first assuming the role.")
    s3_denied_resource = boto3.resource(
        's3', aws_access_key_id=user_key.id, aws_secret_access_key=user_key.secret)
    try:
        for bucket in s3_denied_resource.buckets.all():
            print(bucket.name)
        raise RuntimeError("Expected to get AccessDenied error when listing
buckets!")
    except ClientError as error:
        if error.response['Error']['Code'] == 'AccessDenied':
            print("Attempt to list buckets with no permissions: AccessDenied.")
        else:
            raise

```

```

def list_buckets_from_assumed_role(user_key, assume_role_arn, session_name):
    """
    Assumes a role that grants permission to list the Amazon S3 buckets in the
    account.
    Uses the temporary credentials from the role to list the buckets that are owned
    by the assumed role's account.

    :param user_key: The access key of a user that has permission to assume the
                    role.
    :param assume_role_arn: The Amazon Resource Name (ARN) of the role that
                           grants access to list the other account's buckets.
    :param session_name: The name of the STS session.
    """
    sts_client = boto3.client(
        'sts', aws_access_key_id=user_key.id,
        aws_secret_access_key=user_key.secret)
    try:
        response = sts_clientassume_role(

```

```

        RoleArn=assume_role_arn, RoleSessionName=session_name)
        temp_credentials = response['Credentials']
        print(f"Assumed role {assume_role_arn} and got temporary credentials.")
    except ClientError as error:
        print(f"Couldn't assume role {assume_role_arn}. Here's why: "
              f"{error.response['Error']['Message']}")
        raise

    # Create an S3 resource that can access the account with the temporary
    # credentials.
    s3_resource = boto3.resource(
        's3',
        aws_access_key_id=temp_credentials['AccessKeyId'],
        aws_secret_access_key=temp_credentials['SecretAccessKey'],
        aws_session_token=temp_credentials['SessionToken'])
    print(f"Listing buckets for the assumed role's account:")
    try:
        for bucket in s3_resource.buckets.all():
            print(bucket.name)
    except ClientError as error:
        print(f"Couldn't list buckets for the account. Here's why: "
              f"{error.response['Error']['Message']}")
        raise

def teardown(user, role):
    """
    Removes all resources created during setup.

    :param user: The demo user.
    :param role: The demo role.
    """
    try:
        for attached in role.attached_policies.all():
            policy_name = attached.policy_name
            role.detach_policy(PolicyArn=attached.arn)
            attached.delete()
            print(f"Detached and deleted {policy_name}.")
        role.delete()
        print(f"Deleted {role.name}.")
    except ClientError as error:
        print("Couldn't detach policy, delete policy, or delete role. Here's why: "
              f"{error.response['Error']['Message']}")
        raise

    try:
        for user_pol in user.policies.all():
            user_pol.delete()
            print("Deleted inline user policy.")
        for key in user.access_keys.all():
            key.delete()
            print("Deleted user's access key.")
        user.delete()
        print(f"Deleted {user.name}.")
    except ClientError as error:
        print("Couldn't delete user policy or delete user. Here's why: "
              f"{error.response['Error']['Message']}")
        raise

def usage_demo():
    """Drives the demonstration."""
    print('*'*88)
    print(f"Welcome to the IAM create user and assume role demo.")
    print('*'*88)
    iam_resource = boto3.resource('iam')
    user = None

```

```
role = None
try:
    user, user_key, role = setup(iam_resource)
    print(f"Created {user.name} and {role.name}.")
    show_access_denied_without_role(user_key)
    list_buckets_from_assumed_role(user_key, role.arn, 'AssumeRoleDemoSession')
except Exception:
    print("Something went wrong!")
finally:
    if user is not None and role is not None:
        teardown(user, role)
    print("Thanks for watching!")

if __name__ == '__main__':
    usage_demo()
```

- For API details, see the following topics in *AWS SDK for Python (Boto3) API Reference*.
 - [AttachRolePolicy](#)
 - [CreateAccessKey](#)
 - [CreatePolicy](#)
 - [CreateRole](#)
 - [CreateUser](#)
 - [DeleteAccessKey](#)
 - [DeletePolicy](#)
 - [DeleteRole](#)
 - [DeleteUser](#)
 - [DeleteUserPolicy](#)
 - [DetachRolePolicy](#)
 - [PutUserPolicy](#)

Ruby

SDK for Ruby

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
require "aws-sdk-iam"
require "aws-sdk-s3"

# Wraps the scenario actions.
class ScenarioCreateUserAssumeRole
  attr_reader :iam_resource

  # @param iam_resource [Aws::IAM::Resource] An AWS IAM resource.
  def initialize(iam_resource)
    @iam_resource = iam_resource
  end

  # Waits for the specified number of seconds.
  #
  # @param duration [Integer] The number of seconds to wait.
  def wait(duration)
    puts("Give AWS time to propagate resources...")
    sleep(duration)
  end
end
```

```

    end

    # Creates a user.
    #
    # @param user_name [String] The name to give the user.
    # @return [Aws::IAM::User] The newly created user.
    def create_user(user_name)
        user = @iam_resource.create_user(user_name: user_name)
        puts("Created demo user named #{user.name}.")
    rescue Aws::Errors::ServiceError => e
        puts("Tried and failed to create demo user.")
        puts("\t#{e.code}: #{e.message}")
        puts("\nCan't continue the demo without a user!")
        raise
    else
        user
    end

    # Creates an access key for a user.
    #
    # @param user [Aws::IAM::User] The user that owns the key.
    # @return [Aws::IAM::AccessKeyPair] The newly created access key.
    def create_access_key_pair(user)
        user_key = user.create_access_key_pair
        puts("Created access key pair for user.")
    rescue Aws::Errors::ServiceError => e
        puts("Couldn't create access keys for user #{user.name}.")
        puts("\t#{e.code}: #{e.message}")
        raise
    else
        user_key
    end

    # Creates a role that can be assumed by a user.
    #
    # @param role_name [String] The name to give the role.
    # @param user [Aws::IAM::User] The user who is granted permission to assume the
    # role.
    # @return [Aws::IAM::Role] The newly created role.
    def create_role(role_name, user)
        role = @iam_resource.create_role(
            role_name: role_name,
            assume_role_policy_document: {
                Version: "2012-10-17",
                Statement: [
                    {
                        Effect: "Allow",
                        Principal: {'AWS': user.arn},
                        Action: "sts:AssumeRole"
                    }
                ].to_json
        )
        puts("Created role #{role.name}.")
    rescue Aws::Errors::ServiceError => e
        puts("Couldn't create a role for the demo. Here's why: ")
        puts("\t#{e.code}: #{e.message}")
        raise
    else
        role
    end

    # Creates a policy that grants permission to list S3 buckets in the account, and
    # then attaches the policy to a role.
    #
    # @param policy_name [String] The name to give the policy.
    # @param role [Aws::IAM::Role] The role that the policy is attached to.
    # @return [Aws::IAM::Policy] The newly created policy.
    def create_and_attach_role_policy(policy_name, role)

```

```

policy = @iam_resource.create_policy(
    policy_name: policy_name,
    policy_document: [
        Version: "2012-10-17",
        Statement: [
            Effect: "Allow",
            Action: "s3>ListAllMyBuckets",
            Resource: "arn:aws:s3:::*"
        ]
    ].to_json)
role.attach_policy(policy_arn: policy.arn)
puts("Created policy #{policy.policy_name} and attached it to role
#{role.name}.")
rescue Aws::Errors::ServiceError => e
    puts("Couldn't create a policy and attach it to role #{role.name}. Here's why:
")
    puts("\t#{e.code}: #{e.message}")
    raise
else
    policy
end

# Creates an inline policy for a user that lets the user assume a role.
#
# @param policy_name [String] The name to give the policy.
# @param user [Aws::IAM::User] The user that owns the policy.
# @param role [Aws::IAM::Role] The role that can be assumed.
# @return [Aws::IAM::UserPolicy] The newly created policy.
def create_user_policy(policy_name, user, role)
    policy = user.create_policy(
        policy_name: policy_name,
        policy_document: [
            Version: "2012-10-17",
            Statement: [
                Effect: "Allow",
                Action: "sts:AssumeRole",
                Resource: role.arn
            ]
        ].to_json)
    puts("Created an inline policy for #{user.name} that lets the user assume role
#{role.name}.")
    rescue Aws::Errors::ServiceError => e
        puts("Couldn't create an inline policy for user #{user.name}. Here's why: ")
        puts("\t#{e.code}: #{e.message}")
        raise
    else
        policy
    end

    # Creates an Amazon S3 resource with specified credentials. This is separated
    into a
    # factory function so that it can be mocked for unit testing.
    #
    # @param credentials [Aws::Credentials] The credentials used by the Amazon S3
    resource.
def create_s3_resource(credentials)
    Aws::S3::Resource.new(client: Aws::S3::Client.new(credentials: credentials))
end

    # Lists the S3 buckets for the account, using the specified Amazon S3 resource.
    # Because the resource uses credentials with limited access, it may not be able
    to
    # list the S3 buckets.
    #
    # @param s3_resource [Aws::S3::Resource] An Amazon S3 resource.
def list_buckets(s3_resource)

```

```

count = 10
s3_resource.buckets.each do |bucket|
  puts "\t#{bucket.name}"
  count -= 1
  break if count.zero?
end
rescue Aws::Errors::ServiceError => e
  if e.code == "AccessDenied"
    puts("Attempt to list buckets with no permissions: AccessDenied.")
  else
    puts("Couldn't list buckets for the account. Here's why: ")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
end

# Creates an AWS Security Token Service (AWS STS) client with specified
credentials.
# This is separated into a factory function so that it can be mocked for unit
testing.
#
# @param key_id [String] The ID of the access key used by the STS client.
# @param key_secret [String] The secret part of the access key used by the STS
client.
def create_sts_client(key_id, key_secret)
  Aws::STS::Client.new(access_key_id: key_id, secret_access_key: key_secret)
end

# Gets temporary credentials that can be used to assume a role.
#
# @param role_arn [String] The ARN of the role that is assumed when these
credentials
# are used.
# @param sts_client [AWS::STS::Client] An AWS STS client.
# @return [Aws::AssumeRoleCredentials] The credentials that can be used to assume
the role.
def assume_role(role_arn, sts_client)
  credentials = Aws::AssumeRoleCredentials.new(
    client: sts_client,
    role_arn: role_arn,
    role_session_name: "create-use-assume-role-scenario"
  )
  puts("Assumed role '#{role_arn}', got temporary credentials.")
  credentials
end

# Deletes a role. If the role has policies attached, they are detached and
# deleted before the role is deleted.
#
# @param role [Aws::IAM::Role] The role to delete.
def delete_role(role)
  role.attached_policies.each do |policy|
    name = policy.policy_name
    policy.detach_role(role_name: role.name)
    policy.delete
    puts("Deleted policy #{name}.")
  end
  name = role.name
  role.delete
  puts("Deleted role #{name}.")
rescue Aws::Errors::ServiceError => e
  puts("Couldn't detach policies and delete role #{role.name}. Here's why:")
  puts("\t#{e.code}: #{e.message}")
  raise
end

```

```

# Deletes a user. If the user has inline policies or access keys, they are
deleted
# before the user is deleted.
#
# @param user [Aws::IAM::User] The user to delete.
def delete_user(user)
  user.policies.each do |policy|
    name = policy.name
    policy.delete
    puts("Deleted user policy #{name}.")
  end
  user.access_keys.each do |key|
    key.delete
    puts("Deleted access key for user #{user.name}.")
  end
  name = user.name
  user.delete
  puts("Deleted user #{name}.")
rescue Aws::Errors::ServiceError => e
  puts("Couldn't detach policies and delete user #{user.name}. Here's why:")
  puts("\t#{e.code}: #{e.message}")
end
end

# Runs the IAM create a user and assume a role scenario.
def run_scenario(scenario)
  puts("-" * 88)
  puts("Welcome to the IAM create a user and assume a role demo!")
  puts("-" * 88)

  user = scenario.create_user("doc-example-user-#{Random.uuid}")
  user_key = scenario.create_access_key_pair(user)
  scenario.wait(10)
  role = scenario.create_role("doc-example-role-#{Random.uuid}", user)
  scenario.create_and_attach_role_policy("doc-example-role-policy-#{Random.uuid}", role)
  scenario.create_user_policy("doc-example-user-policy-#{Random.uuid}", user, role)
  scenario.wait(10)
  puts("Try to list buckets with credentials for a user who has no permissions.")
  puts("Expect AccessDenied from this call.")
  scenario.list_buckets(
    scenario.create_s3_resource(Aws::Credentials.new(user_key.id,
user_key.secret)))
  puts("Now, assume the role that grants permission.")
  temp_credentials = scenario.assume_role(
    role.arn, scenario.create_sts_client(user_key.id, user_key.secret))
  puts("Here are your buckets:")
  scenario.list_buckets(scenario.create_s3_resource(temp_credentials))
  puts("Deleting role '#{role.name}' and attached policies.")
  scenario.delete_role(role)
  puts("Deleting user '#{user.name}', policies, and keys.")
  scenario.delete_user(user)

  puts("Thanks for watching!")
  puts("-" * 88)
rescue Aws::Errors::ServiceError => e
  puts("Something went wrong with the demo.")
  puts("\t#{e.code}: #{e.message}")
end

run_scenario(ScenarioCreateUserAssumeRole.new(Aws::IAM::Resource.new)) if
$PROGRAM_NAME == __FILE__

```

- For API details, see the following topics in *AWS SDK for Ruby API Reference*.

- [AttachRolePolicy](#)
- [CreateAccessKey](#)
- [CreatePolicy](#)
- [CreateRole](#)
- [CreateUser](#)
- [DeleteAccessKey](#)
- [DeletePolicy](#)
- [DeleteRole](#)
- [DeleteUser](#)
- [DeleteUserPolicy](#)
- [DetachRolePolicy](#)
- [PutUserPolicy](#)

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
use aws_config::meta::region::RegionProviderChain;
use aws_sdk_iam::Error as iamError;
use aws_sdk_iam::{config::Credentials as iamCredentials, config::Region, Client as iamClient};
use aws_sdk_s3::Client as s3Client;
use aws_sdk_sts::Client as stsClient;
use std::borrow::Borrow;
use tokio::time::{sleep, Duration};
use uuid::Uuid;

#[tokio::main]
async fn main() -> Result<(), iamError> {
    let (client, uuid, list_all_buckets_policy_document, inline_policy_document) =
        initialize_variables().await;

    if let Err(e) = run_iam_operations(
        client,
        uuid,
        list_all_buckets_policy_document,
        inline_policy_document,
    )
    .await
    {
        println!("{}:{}", e);
    };
    Ok(())
}

async fn initialize_variables() -> (iamClient, String, String, String) {
    let region_provider = RegionProviderChain::first_try(Region::new("us-west-2"));
}
```

```

let shared_config =
aws_config::from_env().region(region_provider).load().await;
let client = iamClient::new(&shared_config);
let uuid = Uuid::new_v4().to_string();

let list_all_buckets_policy_document = "{"
    \\"Version\\": \\"2012-10-17\",
    \\"Statement\\": [{{
        \\"Effect\\": \\"Allow\",
        \\"Action\\": \\"s3>ListAllMyBuckets\",
        \\"Resource\\": \\"arn:aws:s3:::*\"]}}
}
.to_string();
let inline_policy_document = "{"
    \\"Version\\": \\"2012-10-17\",
    \\"Statement\\": [{{
        \\"Effect\\": \\"Allow\",
        \\"Action\\": \\"sts:AssumeRole\",
        \\"Resource\\": \"{}\"}}]
}
.to_string();

(
    client,
    uuid,
    list_all_buckets_policy_document,
    inline_policy_document,
)
}

async fn run_iam_operations(
    client: iamClient,
    uuid: String,
    list_all_buckets_policy_document: String,
    inline_policy_document: String,
) -> Result<(), iamError> {
    let user = iam_service::create_user(&client, &format!("{}{}", "iam_demo_user_", uuid)).await?;
    println!(
        "Created the user with the name: {}",
        user.user_name.as_ref().unwrap()
    );
    let key = iam_service::create_access_key(&client,
user.user_name.as_ref().unwrap()).await?;

    let assume_role_policy_document = "{"
        \\"Version\\": \\"2012-10-17\",
        \\"Statement\\": [{{
            \\"Effect\\": \\"Allow\",
            \\"Principal\\": \\"AWS\": \"{}\",
            \\"Action\\": \\"sts:AssumeRole\"}}]
    }
    .to_string()
    .replace("{}", user.arn.as_ref().unwrap());

    let assume_role_role = iam_service::create_role(
        &client,
        &format!("{}{}", "iam_demo_role_", uuid),
        &assume_role_policy_document,
    )
    .await?;
    println!(
        "Created the role with the ARN: {}",
        assume_role_role.arn.as_ref().unwrap()
    );
}

```

```

let list_all_buckets_policy = iam_service::create_policy(
    &client,
    &format!("{}{}", "iam_demo_policy_", uuid),
    &list_all_buckets_policy_document,
)
.await?;
println!(
    "Created policy: {}",
    list_all_buckets_policy.policy_name.as_ref().unwrap()
);

let attach_role_policy_result =
    iam_service::attach_role_policy(&client, &assume_role_role,
&list_all_buckets_policy)
    .await?;
println!(
    "Attached the policy to the role: {}",
    attach_role_policy_result
);

let inline_policy_name = format!("{}{}", "iam_demo_inline_policy_", uuid);
let inline_policy_document =
    inline_policy_document.replace("{}",
assume_role_role.arn.as_ref().unwrap());
    iam_service::create_user_policy(&client, &user, &inline_policy_name,
&inline_policy_document)
    .await?;
println!("Created inline policy.");

//First, fail to list the buckets with the user.
let creds = iamCredentials::from_keys(
    key.access_key_id.as_ref().unwrap(),
    key.secret_access_key.as_ref().unwrap(),
    None,
);
let fail_config = aws_config::from_env()
    .credentials_provider(creds.clone())
    .load()
    .await;
println!("Fail config: {}", fail_config);
let fail_client: s3Client = s3Client::new(&fail_config);
match fail_client.list_buckets().send().await {
    Ok(e) => {
        println!("This should not run. {}", e);
    }
    Err(e) => {
        println!("Successfully failed with error: {}", e)
    }
}

let sts_config = aws_config::from_env()
    .credentials_provider(creds.clone())
    .load()
    .await;
let sts_client: stsClient = stsClient::new(&sts_config);
sleep(Duration::from_secs(10)).await;
let assumed_role = sts_client
    .assume_role()
    .role_arn(assume_role_role.arn.as_ref().unwrap())
    .role_session_name(&format!("{}{}", "iam_demo_assumerole_session_", uuid))
    .send()
    .await;
println!("Assumed role: {}", assumed_role);
sleep(Duration::from_secs(10)).await;

```

```

let assumed_credentials = iamCredentials::from_keys(
    assumed_role
        .as_ref()
        .unwrap()
        .credentials
        .as_ref()
        .unwrap()
        .access_key_id
        .as_ref()
        .unwrap(),
    assumed_role
        .as_ref()
        .unwrap()
        .credentials
        .as_ref()
        .unwrap()
        .secret_access_key
        .as_ref()
        .unwrap(),
    assumed_role
        .as_ref()
        .unwrap()
        .credentials
        .as_ref()
        .unwrap()
        .session_token
        .borrow()
        .clone(),
);

let succeed_config = aws_config::from_env()
    .credentials_provider(assumed_credentials)
    .load()
    .await;
println!("succeed config: {:?}", succeed_config);
let succeed_client: s3Client = s3Client::new(&succeed_config);
sleep(Duration::from_secs(10)).await;
match succeed_client.list_buckets().send().await {
    Ok(_) => {
        println!("This should now run successfully.")
    }
    Err(e) => {
        println!("This should not run. {:?}", e);
        panic!()
    }
}
//Clean up.
iam_service::detach_role_policy(
    &client,
    assume_role.role_name.as_ref().unwrap(),
    list_all_buckets_policy.arn.as_ref().unwrap(),
)
.await?;
iam_service::delete_policy(&client, list_all_buckets_policy).await?;
iam_service::delete_role(&client, &assume_role.role).await?;
println!(
    "Deleted role {}",
    assume_role.role_name.as_ref().unwrap()
);
iam_service::delete_access_key(&client, &user, &key).await?;
println!("Deleted key for {}", key.user_name.as_ref().unwrap());
iam_service::delete_user_policy(&client, &user, &inline_policy_name).await?;
println!("Deleted inline user policy: {}", inline_policy_name);
iam_service::delete_user(&client, &user).await?;
println!("Deleted user {}", user.user_name.as_ref().unwrap());

```

```
    Ok(())
}
```

- For API details, see the following topics in *AWS SDK for Rust API reference*.
 - [AttachRolePolicy](#)
 - [CreateAccessKey](#)
 - [CreatePolicy](#)
 - [CreateRole](#)
 - [CreateUser](#)
 - [DeleteAccessKey](#)
 - [DeletePolicy](#)
 - [DeleteRole](#)
 - [DeleteUser](#)
 - [DeleteUserPolicy](#)
 - [DetachRolePolicy](#)
 - [PutUserPolicy](#)

Using an IAM role to grant permissions to applications running on Amazon EC2 instances

Applications that run on an Amazon EC2 instance must include AWS credentials in the AWS API requests. You could have your developers store AWS credentials directly within the Amazon 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 Amazon 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 Amazon EC2 instance. When you use a role, you don't have to distribute long-term credentials (such as sign-in credentials or access keys) to an Amazon EC2 instance. Instead, the role supplies temporary permissions that applications can use when they make calls to other AWS resources. When you launch an Amazon 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 Amazon EC2 instances requires a bit of extra configuration. An application running on an Amazon EC2 instance is abstracted from AWS by the virtualized operating system. Because of this extra separation, you need an additional step to assign an AWS role and its associated permissions to an Amazon EC2 instance and make them available to its applications. This extra step is the creation of an [*instance profile*](#) 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 Amazon 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 propagates automatically to all the instances.

Note

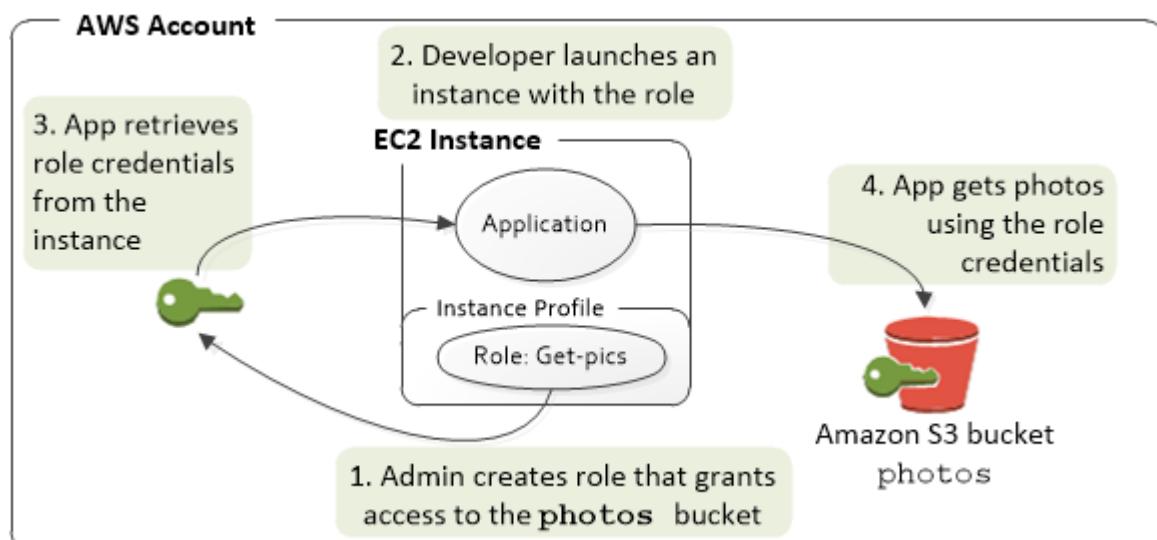
Although a role is usually assigned to an Amazon EC2 instance when you launch it, a role can also be attached to an Amazon EC2 instance currently running. To learn how to attach a role to a running instance, see [IAM Roles for Amazon Amazon EC2](#).

Topics

- [How do roles for Amazon EC2 instances work? \(p. 375\)](#)
- [Permissions required for using roles with Amazon EC2 \(p. 376\)](#)
- [How do I get started? \(p. 380\)](#)
- [Related information \(p. 380\)](#)
- [Using instance profiles \(p. 381\)](#)

How do roles for Amazon EC2 instances work?

In the following figure, a developer runs an application on an Amazon 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 Amazon 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 Amazon 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 Amazon 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 Amazon EC2 instance and assigns the Get-pics role to that instance.

Note

If you use the IAM console, the instance profile is managed for you and is mostly transparent to you. However, if you use the AWS CLI or API to create and manage the role and Amazon 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 \(p. 426\)](#) that represent the role and are valid for a limited period of time.

With some [AWS SDKs](#), the developer can use a provider that manages the temporary security credentials transparently. (The documentation for individual AWS SDKs describes the features supported by that SDK for managing credentials.)

Alternatively, the application can get the temporary credentials directly from the instance metadata of the Amazon EC2 instance. Credentials and related values are available from the `iam/security-credentials/role-name` category (in this case, `iam/security-credentials/Get-pics`) of the metadata. If the application gets the credentials from the instance metadata, it can cache the credentials.

4. Using the retrieved 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 available on the instance automatically rotate 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. It is possible to use the AWS SDK to manage credentials so the application does not need to include additional logic to refresh the credentials. For example, instantiating clients with Instance Profile Credential Providers. 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 returned in the `iam/security-credentials/role-name` category.

Permissions required for using roles with Amazon EC2

To launch an instance with a role, the developer must have permission to launch Amazon EC2 instances and permission to pass IAM roles.

The following sample policy allows users to use the AWS Management Console to launch an instance with a role. The policy includes wildcards (*) to allow a user to pass any role and to perform the listed Amazon EC2 actions. The `ListInstanceProfiles` action allows users to view all of the roles 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": [  
        {  
            "Sid": "IamPassRole",  
            "Effect": "Allow",  
            "Action": "iam:PassRole",  
            "Resource": "*",  
            "Condition": {  
                "StringEquals": {  
                    "iam:PassedToService": "ec2.amazonaws.com"  
                }  
            }  
        },  
        {  
            "Sid": "ListEc2AndListInstanceProfiles",  
            "Effect": "Allow",  
            "Action": [  
                "iam>ListInstanceProfiles",  
                "ec2:Describe*",  
                "ec2:Search*",  
            ]  
        }  
    ]  
}
```

```

        "ec2:Get*"
    ],
    "Resource": "*"
}
]
```

Restricting which roles can be passed to Amazon EC2 instances (using PassRole)

You can use the PassRole permission to restrict which role a user can pass to an Amazon EC2 instance when the user 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 Amazon EC2 instances and to work with Amazon S3 buckets, but the role she passes to an Amazon EC2 instance has permissions to work with IAM and 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 Amazon 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 Amazon EC2 instances. In the Resource element of the policy, you list the role or roles that the user is allowed to pass to Amazon 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 Amazon EC2 instance with a specific role

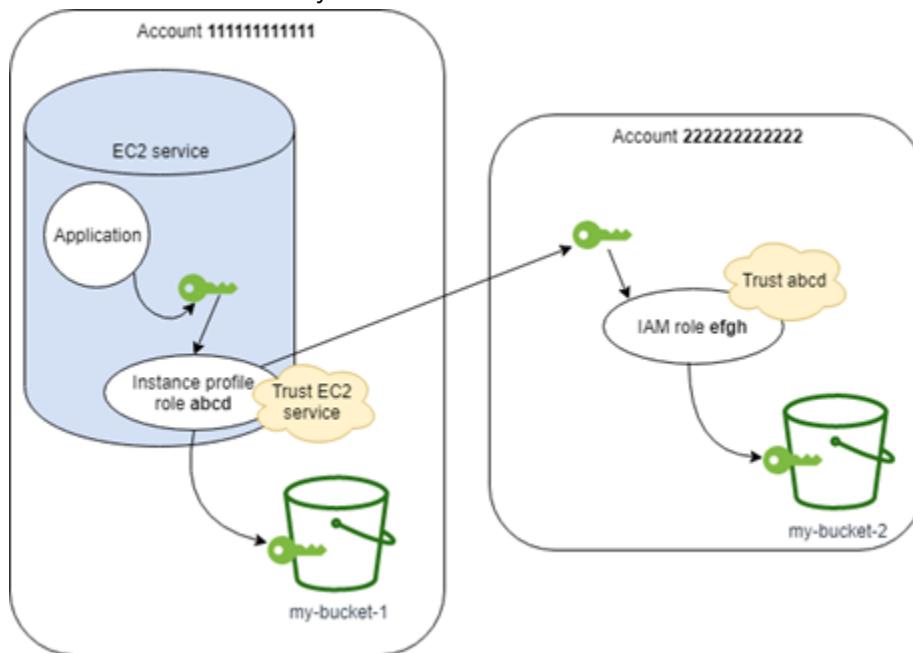
The following sample policy allows users to use the Amazon EC2 API to launch an instance with a role. The Resource element specifies the Amazon Resource Name (ARN) of a role. By specifying the ARN, the policy grants the user the permission to pass only the Get-pics role. If the user tries to specify 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: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 Amazon 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 Amazon 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 Amazon 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:GetBucketLocation",
                "s3:GetAccountPublicAccessBlock",
                "s3>ListAccessPoints",
                "s3>ListAllMyBuckets"
            ],
            "Resource": "arn:aws:s3:::*"
        },
        {
            "Sid": "AllowListAndReadS3ActionOnMyBucket",
            "Effect": "Allow",
            "Action": [
                "s3:Get*",
                "s3:List*"
            ],
            "Resource": "arn:aws:s3:::my-bucket-1"
        }
    ]
}
```

```

        "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:GetBucketLocation",
                "s3:GetAccountPublicAccessBlock",
                "s3>ListAccessPoints",
                "s3>ListAllMyBuckets"
            ],
            "Resource": "arn:aws:s3:::/*"
        },
        {
            "Sid": "AllowListAndReadS3ActionOnMyBucket",
            "Effect": "Allow",
            "Action": [
                "s3:Get*",
                "s3>List*"
            ],
            "Resource": [
                "arn:aws:s3:::my-bucket-2/*",

```

```
        "arn:aws:s3:::my-bucket-2"
    ]
}
}
```

The efg role must trust the abcd instance profile role to assume it. To do this, the efg role must have the following trust policy:

Account 222222222222 efg Role Trust Policy

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "efgTrustPolicy",
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Principal": {"AWS": "arn:aws:iam::111111111111:role/abcd"}
        }
    ]
}
```

How do I get started?

To understand how roles work with Amazon EC2 instances, you need to use the IAM console to create a role, launch an Amazon EC2 instance that uses that role, and then examine the running instance. You can 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 Amazon 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 Amazon 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 Amazon EC2 instances, see the following information:

- For more information about [using IAM roles with Amazon EC2 instances](#), go to the *Amazon EC2 User Guide for Linux Instances*.
- To create a role, see [Creating IAM roles \(p. 250\)](#)
- For more information about using temporary security credentials, see [Temporary security credentials in IAM \(p. 426\)](#).
- 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 \(p. 381\)](#).
- 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.

You can use the AWS Management Console to delete IAM roles and instance profiles for Amazon EC2 if the role and the instance profile have the same name. To learn more about deleting instance profiles, see [Deleting roles or instance profiles \(p. 396\)](#).

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.

You can attach tags to your IAM resources, including instance profiles, to identify, organize, and control access to them. You can tag instance profiles only when you use the AWS CLI or AWS API.

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](#)
- Tag an instance profile: [aws iam tag-instance-profile](#)
- List tags for an instance profile: [aws iam list-instance-profile-tags](#)
- Untag an instance profile: [aws iam untag-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](#)
- Tag an instance profile: [TagInstanceProfile](#)
- List tags on an instance profile: [ListInstanceProfileTags](#)
- Untag an instance profile: [UntagInstanceProfile](#)
- 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 permit 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 re-authenticate and request new credentials.

Note

You cannot revoke the session for a [service-linked role \(p. 185\)](#).

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.

For more information on denying access, see [Disabling permissions for temporary security credentials \(p. 456\)](#).

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, 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. Select the **I acknowledge that I am revoking all active sessions for this role** check box and 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 choose **Revoke active sessions**. Any user who assumes the role *after* you choose **Revoke active sessions** is **not** affected.

When you apply a new policy to a user or a resource, it can 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 \(p. 1172\)](#).

Note

Don't worry about remembering to delete the policy. Any user who assumes the role *after* you revoke sessions is not affected by the policy. If you choose to **Revoke Sessions** again later, then the date and 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. 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
```

Revoking session permissions before a specified time

You can also revoke session permissions programmatically by specifying a value for the [aws:TokenIssueTime \(p. 1365\)](#) key in the Condition element of a policy.

This policy denies all permissions 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 does not affect requests that are signed with the long-term credentials of the IAM user.

This 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.

```
{  
    "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. 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
```

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 \(p. 384\)](#)
- [Deleting roles or instance profiles \(p. 396\)](#)

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 \(p. 385\)](#)
- [Generate a policy based on access information \(p. 385\)](#)
- [Modifying a role \(console\) \(p. 385\)](#)
- [Modifying a role \(AWS CLI\) \(p. 389\)](#)
- [Modifying a role \(AWS API\) \(p. 392\)](#)

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 \(p. 616\)](#).

Generate a policy based on access information

You might sometimes grant permissions to an IAM entity (user or role) beyond what they require. To help you refine the permissions that you grant, you can generate an IAM policy that is based on the access activity for an entity. IAM Access Analyzer reviews your AWS CloudTrail logs and generates a policy template that contains the permissions that have been used by the entity in your specified date range. You can use the template to create a managed policy with fine-grained permissions and then attach it to the IAM entity. That way, you grant only the permissions that the user or role needs to interact with AWS resources for your specific use case. To learn more, see [Generate policies based on access activity \(p. 589\)](#).

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 roles \(console\) \(p. 406\)](#).

Topics

- [Modifying a role trust policy \(console\) \(p. 385\)](#)
- [Modifying a role permissions policy \(console\) \(p. 387\)](#)
- [Modifying a role description \(console\) \(p. 387\)](#)
- [Modifying a role maximum session duration \(console\) \(p. 388\)](#)
- [Modifying a role permissions boundary \(console\) \(p. 388\)](#)

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 \(p. 185\)](#).

Notes

- 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 \(p. 501\)](#). If a permissions boundary is set for the user, then it must allow the sts:AssumeRole action.
- To allow users to assume the current role again within a role session, specify the role ARN or AWS account ARN as a principal in the role trust policy. AWS services that provide compute resources such as Amazon EC2, Amazon ECS, Amazon EKS, and Lambda provide temporary credentials and automatically rotate these credentials. This ensures that you always have a valid set of credentials. For these services, it's not necessary to assume the current role again to obtain temporary credentials. However, if you intend to pass [session tags \(p. 417\)](#) or a [session policy \(p. 487\)](#), you need to assume the current role again.

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 policy**.
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::44445556666: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 \(p. 277\)](#).

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 policy** to save your changes.

For more information about policy structure and syntax, see [Policies and permissions in IAM \(p. 485\)](#) and the [IAM JSON policy elements reference \(p. 1260\)](#).

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 \(p. 277\)](#).

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 **User 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 \(p. 494\)](#).

- 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 \(p. 185\)](#) 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 \(p. 1224\)](#) 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 \(p. 494\)](#).

- To attach an existing managed policy to the role, choose **Add permissions** and then choose **Attach policies**.
- To edit an existing inline policy, expand the policy and choose **Edit**.
- To embed a new inline policy, choose **Add permissions** and then choose **Create 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. In the **Summary** section, choose **Edit**.
5. Enter a new description in the box and choose **Save changes**.

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. In the **Summary** section, choose **Edit**.
5. For **Maximum session duration**, choose a value. Alternatively, choose **Custom duration** and enter a value (in seconds).
6. Choose **Save changes**.

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 \(p. 382\)](#).

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 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 \(AWS CLI\) \(p. 285\)](#).
- To specify a session duration using the AWS API, use the `DurationSeconds` parameter. To learn more, see [Switching to an IAM role \(AWS API\) \(p. 291\)](#).

Modifying a role permissions boundary (console)

To change the maximum permissions allowed for a role, modify the role's [permissions boundary \(p. 501\)](#).

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 with the [permissions boundary \(p. 501\)](#) that 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 roles \(AWS CLI or AWS API\) \(p. 406\)](#).

Topics

- [Modifying a role trust policy \(AWS CLI\) \(p. 389\)](#)
- [Modifying a role permissions policy \(AWS CLI\) \(p. 390\)](#)
- [Modifying a role description \(AWS CLI\) \(p. 391\)](#)
- [Modifying a role maximum session duration \(AWS CLI\) \(p. 391\)](#)
- [Modifying a role permissions boundary \(AWS CLI\) \(p. 392\)](#)

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 \(p. 185\)](#).

Notes

- 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 \(p. 501\)](#). If a permissions boundary is set for the user, then it must allow the `sts:AssumeRole` action.
- To allow users to assume the current role again within a role session, specify the role ARN or AWS account ARN as a principal in the role trust policy. AWS services that provide compute resources such as Amazon EC2, Amazon ECS, Amazon EKS, and Lambda provide temporary credentials and automatically rotate these credentials. This ensures that you always have a valid set of credentials. For these services, it's not necessary to assume the current role again to obtain temporary credentials. However, if you intend to pass [session tags \(p. 417\)](#) or a [session policy \(p. 487\)](#), you need to assume the current role again. To learn how to modify a role trust policy to add the principal role ARN or AWS account ARN, see [Modifying a role trust policy \(console\) \(p. 385\)](#).

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 \(p. 277\)](#).

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 \(p. 277\)](#).

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 \(p. 185\)](#) 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 \(p. 1224\)](#) 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 \(p. 382\)](#).

Modifying a role permissions boundary (AWS CLI)

To change the maximum permissions allowed for a role, modify the role's [permissions boundary \(p. 501\)](#).

To change the managed policy used to set the permissions boundary for a role (AWS CLI)

1. (Optional) To view the current [permissions boundary \(p. 501\)](#) 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 roles \(AWS CLI or AWS API\) \(p. 406\)](#).

Topics

- [Modifying a role trust policy \(AWS API\) \(p. 392\)](#)
- [Modifying a role permissions policy \(AWS API\) \(p. 394\)](#)
- [Modifying a role description \(AWS API\) \(p. 394\)](#)
- [Modifying a role maximum session duration \(AWS API\) \(p. 395\)](#)
- [Modifying a role permissions boundary \(AWS API\) \(p. 395\)](#)

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 \(p. 185\)](#).

Notes

- 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 \(p. 501\)](#). If a permissions boundary is set for the user, then it must allow the `sts:AssumeRole` action.
- To allow users to assume the current role again within a role session, specify the role ARN or AWS account ARN as a principal in the role trust policy. AWS services that provide compute resources such as Amazon EC2, Amazon ECS, Amazon EKS, and Lambda provide temporary credentials and automatically rotate these credentials. This ensures that you always have a valid set of credentials. For these services, it's not necessary to assume the current role again

to obtain temporary credentials. However, if you intend to pass [session tags \(p. 417\)](#) or a [session policy \(p. 487\)](#), you need to assume the current role again. To learn how to modify a role trust policy to add the principal role ARN or AWS account ARN, see [Modifying a role trust policy \(console\) \(p. 385\)](#).

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 \(p. 277\)](#).

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 \(p. 277\)](#).

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](#)

The output of this operation 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, 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 \(p. 185\)](#) 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 \(p. 1224\)](#) 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 \(p. 382\)](#).

Modifying a role permissions boundary (AWS API)

To change the maximum permissions allowed for a role, modify the role's [permissions boundary \(p. 501\)](#).

To change the managed policy used to set the permissions boundary for a role (AWS API)

1. (Optional) To view the current [permissions boundary \(p. 501\)](#) 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 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 \(p. 382\)](#).

Topics

- [View role access \(p. 396\)](#)
- [Deleting a service-linked role \(p. 397\)](#)
- [Deleting an IAM role \(console\) \(p. 397\)](#)
- [Deleting an IAM role \(AWS CLI\) \(p. 398\)](#)
- [Deleting an IAM role \(AWS API\) \(p. 399\)](#)
- [Related information \(p. 399\)](#)

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. You should view this information because you don't want to remove access from someone using the role.

The date of the role last activity might not match the last date reported in the **Access Advisor** tab. The [Access Advisor \(p. 622\)](#) tab reports activity only for services allowed by the role permissions policies. The date of the role last activity includes the last attempt to access any service in AWS.

Note

The tracking period for a role 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 \(p. 621\)](#).

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 with the 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 **Summary** page also includes **Last activity**, which displays the last used date for the role. 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 \(p. 185\)](#), 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 **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 \(p. 248\)](#).

Deleting an IAM role (console)

When you use the AWS Management Console to delete a role, IAM automatically detaches managed policies associated with the role. It also automatically deletes any inline policies associated with the role, and 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**.
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 if the role is currently active. If you want to proceed, enter the name of the role in the text input field and choose **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. The instance profile is deleted 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 inline policies associated with the role. You must also detach managed policies associated with the role. 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 associated with.
 - 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 inline policies that are in the role, enter the following command:

```
aws iam list-role-policies --role-name role-name
```
 - b. To delete each inline policy from the role, enter the following command for each policy:

```
aws iam delete-role-policy --role-name role-name --policy-name policy-name
```
 - c. To list all managed policies that are attached to the role, enter the following command:

```
aws iam list-attached-role-policies --role-name role-name
```
 - d. To detach each managed policy from the role, enter the following command for each policy:

```
aws iam detach-role-policy --role-name role-name --policy-arn policy-arn
```
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 inline policies associated with the role. You must also detach managed policies associated with the role. 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 associated with, call [ListInstanceProfilesForRole](#).

To remove the role from an instance profile, 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 inline policies for a role, call [ListRolePolicies](#).

To delete inline policies that are associated with the role, call [DeleteRolePolicy](#). You must pass the role name and inline policy name.

3. To list all managed policies that are attached to a role, call [ListAttachedRolePolicies](#).

To detach managed policies that are attached to the role, call [DetachRolePolicy](#). You must pass the role name and managed policy ARN.

4. Call [DeleteRole](#) to delete the role.

Related information

For general information about instance profiles, see [Using instance profiles \(p. 381\)](#).

For general information about service-linked roles, see [Using service-linked roles \(p. 242\)](#).

Tagging IAM resources

A *tag* is a custom attribute label that you can assign to an AWS resource. Each tag has two parts:

- A *tag key* (for example, CostCenter, Environment, Project, or Purpose).
- An optional field known as a *tag value* (for example, 111122223333, Production, or a team name). Omitting the tag value is the same as using an empty string.

Together these are known as key-value pairs. For limits on the number of tags you can have on IAM resources, see [IAM and AWS STS quotas \(p. 1220\)](#).

Note

For details about case sensitivity for tag keys and tag key values, see [Case sensitivity](#).

Tags help you identify and organize your AWS resources. Many AWS services support tagging, so you can assign the same tag to resources from different services to indicate that the resources are related. For example, you can assign the same tag to an IAM role that you assign to an Amazon S3 bucket. For more information about tagging strategies, see the [Tagging AWS resources User Guide](#).

In addition to identifying, organizing, and tracking your IAM resources with tags, you can use tags in IAM policies to help control who can view and interact with your resources. To learn more about using tags to control access, see [Controlling access to and for IAM users and roles using tags \(p. 521\)](#).

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 \(p. 417\)](#).

Choose an AWS tag naming convention

When you begin attaching tags to your IAM resources, 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.

Note

If your account is a member of AWS Organizations, see [Tag policies](#) in the Organizations user guide to learn more about using tags in Organizations.

Best practices for tag naming

These are some best practices and naming conventions for tags.

Names for AWS tags are case sensitive so ensure that they are used consistently. For example, the tags `CostCenter` and `costcenter` are different, so one might be configured as a cost allocation tag for financial analysis and reporting and the other one might not be. Similarly, the `Name` tag appears in the AWS Console for many resources, but the `name` tag does not.

A number of tags are predefined by AWS or created automatically by various AWS services. Many AWS-defined tags names use all lowercase, with hyphens separating words in the name, and prefixes to identify the source service for the tag. For example:

- `aws:ec2spot:fleet-request-id` identifies the Amazon EC2 Spot Instance Request that launched the instance.
- `aws:cloudformation:stack-name` identifies the AWS CloudFormation stack that created the resource.
- `elasticbeanstalk:environment-name` identifies the application that created the resource.

Consider naming your tags using all lowercase, with hyphens separating words, and a prefix identifying the organization name or abbreviated name. For example, for a fictitious company named `AnyCompany`, you might define tags such as:

- `anycompany:cost-center` to identify the internal Cost Center code
- `anycompany:environment-type` to identify whether the environment is development, test, or production
- `anycompany:application-id` to identify the application the resource was created for

The prefix ensures that tags are clearly identified as having been defined by your organization and not by AWS or a third-party tool that you may be using. Using all lowercase with hyphens for separators avoids confusion about how to capitalize a tag name. For example, `anycompany:project-id` is simpler to remember than `ANYCOMPANY:ProjectID`, `anycompany:projectID`, or `Anycompany:ProjectId`.

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 resources, AWS STS assume-role sessions, and AWS STS federated user sessions:

Character requirements – Tag keys and values can include any combination of letters, numbers, spaces, and _ . : / = + - @ symbols.

Case sensitivity – Case sensitivity for tag keys differs depending on the type of IAM resource that is tagged. Tag key values for IAM users and roles 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=finance** tag and you add the **department=hr** tag, it replaces the first tag. A second tag is not added.

For other IAM resource types, tag key values are case sensitive. That means you can have separate **Costcenter** and **costcenter** tag keys. For example, if you have tagged a customer managed policy with the **Costcenter = 1234** tag and you add the **costcenter = 5678** tag, the policy will have both the **Costcenter** and **costcenter** tag keys.

As a best practice, we recommend that you avoid using similar tags with inconsistent case treatment. We recommend that you decide on a strategy for capitalizing tags, and consistently implement that strategy across all resource types. To learn more about best practices for tagging, see [Tagging AWS Resources](#) in the AWS General Reference.

The following lists show the differences in case sensitivity for tag keys that are attached to IAM resources.

Tag key values are **not** case sensitive:

- IAM roles
- IAM users

Tag key values are case sensitive:

- Customer managed policies
- Instance profiles
- OpenID Connect identity providers
- SAML identity providers
- Server certificates
- Virtual MFA devices

Additionally, the following rules apply:

- 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 tags \(p. 521\)](#).

Applying and editing tags

Observe the following conventions when attaching tags to IAM resources:

- You can tag most IAM resources, but not groups, assumed roles, access reports, or hardware-based MFA devices.
- You cannot use Tag Editor to tag IAM resources. Tag Editor does not support IAM tags. For information about using Tag Editor with other services, see [Working with Tag Editor](#) in the *AWS Resource Groups User Guide*.
- To tag an IAM resource, you must have specific permissions. To tag or untag resources, you must also have permission to list tags. For more information, see the list of topics for each IAM resource at the end of this page.
- The number and size of IAM resources in an AWS account are limited. For more information, see [IAM and AWS STS quotas \(p. 1220\)](#).
- You can apply the same tag to multiple IAM resources. 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 \(p. 1224\)](#).
- IAM entities (users or roles) 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 entity (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.

Topics

- [Tagging IAM users \(p. 402\)](#)
- [Tagging IAM roles \(p. 405\)](#)
- [Tagging customer managed policies \(p. 407\)](#)
- [Tagging IAM identity providers \(p. 409\)](#)
- [Tagging instance profiles for Amazon EC2 roles \(p. 412\)](#)
- [Tagging server certificates \(p. 414\)](#)
- [Tagging virtual MFA devices \(p. 416\)](#)
- [Passing session tags in AWS STS \(p. 417\)](#)

Tagging IAM users

You can use IAM tag key-value pairs to add custom attributes to an IAM user. 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 a user's access to resources or to control what

tags can be attached to a user. To learn more about using tags to control access, see [Controlling access to and for IAM users and roles using tags \(p. 521\)](#).

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 \(p. 417\)](#).

Permissions required for tagging IAM users

You must configure permissions to allow an IAM user to tag other users. You can specify one or all of the following IAM tag actions in an IAM policy:

- `iam>ListUserTags`
- `iam:TagUser`
- `iam:UntagUser`

To allow an IAM user to add, list, or remove a tag for a specific user

Add the following statement to the permissions policy for the IAM user that needs to manage tags. Use your account number and replace `<username>` with the name of the user whose tags need to be managed. To learn how to create a policy using this example JSON policy document, see [the section called “Creating policies using the JSON editor” \(p. 583\)](#).

```
{  
    "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 using the JSON editor” \(p. 583\)](#).

```
{  
    "Effect": "Allow",  
    "Action": [  
        "iam>ListUserTags",  
        "iam:TagUser",  
        "iam:UntagUser"  
    ],  
    "Resource": "arn:aws:iam::user/${aws:username}"  
}
```

To allow an IAM user to add a tag to a specific user

Add the following statement to the permissions policy for the IAM user that needs to add, but not remove, tags for a specific user.

Note

The `iam:TagUser` action requires that you also include the `iam>ListUserTags` action.

To use this policy, replace `<username>` with the name of the user whose tags need to be managed. To learn how to create a policy using this example JSON policy document, see [the section called “Creating policies using the JSON editor” \(p. 583\)](#).

```
{  
    "Effect": "Allow",  
    "Action": [  
        "iam>ListUserTags",  
        "iam:TagUser"  
    ],  
    "Resource": "arn:aws:iam::<account-number>:user/<username>"  
}
```

Alternatively, you can use an AWS managed policy such as [IAMFullAccess](#) to provide full access to IAM.

Managing tags on IAM users (console)

You can manage tags for IAM users from the AWS Management Console.

To manage tags on 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 of the console, choose **Users** and then choose the name of the user that you want to edit.
3. Choose the **Tags** tab and then complete one of the following actions:
 - Choose **Add new tag** if the user does not yet have tags.
 - Choose **Manage 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 users (AWS CLI or AWS API)

You can list, attach, or remove tags for IAM users. You can use the AWS CLI or the AWS API to manage tags for IAM users.

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 user (AWS CLI or AWS API)

- AWS CLI: [aws iam tag-user](#)
- AWS API: [TagUser](#)

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 \(p. 1298\)](#).

Tagging IAM roles

You can use IAM tag key-value pairs to add custom attributes to an IAM role. For example, to add location information to a role, 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 a role's access to resources or to control what tags can be attached to a role. To learn more about using tags to control access, see [Controlling access to and for IAM users and roles using tags \(p. 521\)](#).

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 \(p. 417\)](#).

Permissions required for tagging IAM roles

You must configure permissions to allow an IAM role to tag other entities (users or roles). You can specify one or all of the following IAM tag actions in an IAM policy:

- iam>ListRoleTags
- iam>TagRole
- iam>UntagRole
- iam>ListUserTags
- iam>TagUser
- iam>UntagUser

To allow an IAM role to add, list, or remove a tag for a specific user

Add the following statement to the permissions policy for the IAM role that needs to manage tags. Use your account number and replace **<username>** with the name of the user whose tags need to be managed. To learn how to create a policy using this example JSON policy document, see [the section called "Creating policies using the JSON editor" \(p. 583\)](#).

```
{  
    "Effect": "Allow",  
    "Action": [  
        "iam>ListUserTags",  
        "iam>TagUser",  
        "iam>UntagUser"  
    ],  
    "Resource": "arn:aws:iam::<account-number>:user/<username>"  
}
```

To allow an IAM role to add a tag to a specific user

Add the following statement to the permissions policy for the IAM role that needs to add, but not remove, tags for a specific user.

Note

The **iam:TagRole** action requires that you also include the **iam>ListRoleTags** action.

To use this policy, replace **<username>** with the name of the user whose tags need to be managed. To learn how to create a policy using this example JSON policy document, see [the section called "Creating policies using the JSON editor" \(p. 583\)](#).

```
{  
    "Effect": "Allow",  
    "Action": [  
        "iam>TagUser",  
        "iam>UntagUser"  
    ],  
    "Resource": "arn:aws:iam::<account-number>:user/<username>"  
}
```

```
        "iam>ListUserTags",
        "iam>TagUser"
],
"Resource": "arn:aws:iam:<account-number>:user/<username>"}
```

To allow an IAM role to add, list, or remove a tag for a specific role

Add the following statement to the permissions policy for the IAM role that needs to manage tags. Replace `<rolename>` with the name of the role whose tags need to be managed. To learn how to create a policy using this example JSON policy document, see [the section called “Creating policies using the JSON editor” \(p. 583\)](#).

```
{
    "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 roles (console)

You can manage tags for IAM roles from the AWS Management Console.

To manage tags on 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** and then choose the name of the role that you want to edit.
3. Choose the **Tags** tab and then complete one of the following actions:
 - Choose **Add new tag** if the role does not yet have tags.
 - Choose **Manage 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 roles (AWS CLI or AWS API)

You can list, attach, or remove tags for IAM roles. You can use the AWS CLI or the AWS API to manage tags for IAM 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 attach tags to an IAM role (AWS CLI or AWS API)

- AWS CLI: [aws iam tag-role](#)
- AWS API: [TagRole](#)

To remove tags from an IAM role (AWS CLI or AWS API)

- AWS CLI: [aws iam untag-role](#)
- AWS API: [UntagRole](#)

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 \(p. 1298\)](#).

Tagging customer managed policies

You can use IAM tag key-value pairs to add custom attributes to your customer managed policies. For example, to tag a policy with department information, you can add the tag key **Department** and the tag value **eng**. Or, you might want to tag policies to indicate that they are for a specific environment, such as **Environment = lab**. You can use tags to control access to resources or to control what tags can be attached to a resource. To learn more about using tags to control access, see [Controlling access to and for IAM users and roles using tags \(p. 521\)](#).

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 \(p. 417\)](#).

Permissions required for tagging customer managed policies

You must configure permissions to allow an IAM entity (users or roles) to tag customer managed policies. You can specify one or all of the following IAM tag actions in an IAM policy:

- `iam>ListPolicyTags`
- `iam:TagPolicy`
- `iam:UntagPolicy`

To allow an IAM entity (user or role) to add, list, or remove a tag for a customer managed policy

Add the following statement to the permissions policy for the IAM entity that needs to manage tags. Use your account number and replace `<policyname>` with the name of the policy whose tags need to be managed. To learn how to create a policy using this example JSON policy document, see [the section called "Creating policies using the JSON editor" \(p. 583\)](#).

```
{  
    "Effect": "Allow",  
    "Action": [  
        "iam>ListPolicyTags",  
        "iam:TagPolicy",  
        "iam:UntagPolicy"  
    ],  
    "Resource": "arn:aws:iam::<account-number>:policy/<policyname>"  
}
```

To allow an IAM entity (user or role) to add a tag to a specific customer managed policy

Add the following statement to the permissions policy for the IAM entity that needs to add, but not remove, tags for a specific policy.

Note

The `iam:TagPolicy` action requires that you also include the `iam>ListPolicyTags` action.

To use this policy, replace `<policynname>` with the name of the policy whose tags need to be managed. To learn how to create a policy using this example JSON policy document, see [the section called "Creating policies using the JSON editor" \(p. 583\)](#).

```
{  
    "Effect": "Allow",  
    "Action": [  
        "iam>ListPolicyTags",  
        "iam:TagPolicy"  
    ],  
    "Resource": "arn:aws:iam::<account-number>:policy/<policynname>"  
}
```

Alternatively, you can use an AWS managed policy such as [IAMFullAccess](#) to provide full access to IAM.

Managing tags on IAM customer managed policies (console)

You can manage tags for IAM customer managed policies from the AWS Management Console.

To manage tags on customer managed 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 of the console, choose **Policies** and then choose the name of the customer managed policy that you want to edit.
3. Choose the **Tags** tab and then choose **Manage tags**.
4. Add or remove tags to complete the set of tags. Then choose **Save changes**.

Managing tags on IAM customer managed policies (AWS CLI or AWS API)

You can list, attach, or remove tags for IAM customer managed policies. You can use the AWS CLI or the AWS API to manage tags for IAM customer managed policies.

To list the tags currently attached to an IAM customer managed policy (AWS CLI or AWS API)

- AWS CLI: [aws iam list-policy-tags](#)
- AWS API: [ListPolicyTags](#)

To attach tags to an IAM customer managed policy(AWS CLI or AWS API)

- AWS CLI: [aws iam tag-policy](#)
- AWS API: [TagPolicy](#)

To remove tags from an IAM customer managed policy (AWS CLI or AWS API)

- AWS CLI: [aws iam untag-policy](#)
- AWS API: [UntagPolicy](#)

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 \(p. 1298\)](#).

Tagging IAM identity providers

You can use IAM tag key-value pairs to add custom attributes to IAM identity providers (IdPs).

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 \(p. 417\)](#).

To learn about tagging IdPs in IAM, see the following sections:

Topics

- [Tagging OpenID Connect \(OIDC\) identity providers \(p. 409\)](#)
- [Tagging IAM SAML identity providers \(p. 411\)](#)

Tagging OpenID Connect (OIDC) identity providers

You can use IAM tag key-values to add custom attributes to IAM OpenID Connect (OIDC) identity providers. For example, to identify an OIDC identity provider, you can add the tag key **google** and the tag value **oidc**. You can use tags to control access to resources or to control what tags can be attached to an object. To learn more about using tags to control access, see [Controlling access to and for IAM users and roles using tags \(p. 521\)](#).

Permissions required for tagging IAM OIDC identity providers

You must configure permissions to allow an IAM entity (user or role) to tag IAM OIDC identity providers. You can specify one or all of the following IAM tag actions in an IAM policy:

- `iam>ListOpenIDConnectProviderTags`
- `iam:TagOpenIDConnectProvider`
- `iam:UntagOpenIDConnectProvider`

To allow an IAM entity (user or role) to add, list, or remove a tag for an IAM OIDC identity provider

Add the following statement to the permissions policy for the IAM entity that needs to manage tags. Use your account number and replace `<OIDCProviderName>` with the name of the OIDC provider whose tags need to be managed. To learn how to create a policy using this example JSON policy document, see [the section called "Creating policies using the JSON editor" \(p. 583\)](#).

```
{  
    "Effect": "Allow",  
    "Action": [  
        "iam>ListOpenIDConnectProviderTags",  
        "iam:TagOpenIDConnectProvider",  
        "iam:UntagOpenIDConnectProvider"  
    ],  
    "Resource": "arn:aws:iam::<account-number>:oidc-provider/<OIDCProviderName>"  
}
```

To allow an IAM entity (user or role) to add a tag to a specific IAM OIDC identity provider

Add the following statement to the permissions policy for the IAM entity that needs to add, but not remove, tags for a specific identity provider.

Note

The `iam:TagOpenIDConnectProvider` action requires that you also include the `iam>ListOpenIDConnectProviderTags` action.

To use this policy, replace `<OIDCProviderName>` with the name of the OIDC provider whose tags need to be managed. To learn how to create a policy using this example JSON policy document, see [the section called "Creating policies using the JSON editor" \(p. 583\)](#).

```
{  
    "Effect": "Allow",  
    "Action": [  
        "iam>ListOpenIDConnectProviderTags",  
        "iam:TagOpenIDConnectProvider"  
    ],  
    "Resource": "arn:aws:iam::<account-number>:oidc-provider/<OIDCProviderName>"  
}
```

Alternatively, you can use an AWS managed policy such as [IAMFullAccess](#) to provide full access to IAM.

Managing tags on IAM OIDC identity providers (console)

You can manage tags for IAM OIDC identity providers from the AWS Management Console.

Note

You can manage tags using the new Identity providers console experience only.

To manage tags on OIDC identity providers (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 **Identity providers** and then choose the name of the identity provider that you want to edit.
3. In the **Tags** section, choose **Manage tags** and then complete one of the following actions:
 - Choose **Add tag** if the OIDC identity provider does not yet have tags or to add a new tag.
 - Edit existing tag keys and values.
 - Choose **Remove tag** to remove a tag.
4. Then choose **Save changes**.

Managing tags on IAM OIDC identity providers (AWS CLI or AWS API)

You can list, attach, or remove tags for IAM OIDC identity providers. You can use the AWS CLI or the AWS API to manage tags for IAM OIDC identity providers.

To list the tags currently attached to an IAM OIDC identity provider (AWS CLI or AWS API)

- AWS CLI: [aws iam list-open-id-connect-provider-tags](#)
- AWS API: [ListOpenIDConnectProviderTags](#)

To attach tags to an IAM OIDC identity provider (AWS CLI or AWS API)

- AWS CLI: [aws iam tag-open-id-connect-provider](#)
- AWS API: [TagOpenIDConnectProvider](#)

To remove tags from an IAM OIDC identity provider (AWS CLI or AWS API)

- AWS CLI: [aws iam untag-open-id-connect-provider](#)
- AWS API: [UntagOpenIDConnectProvider](#)

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 \(p. 1298\)](#).

Tagging IAM SAML identity providers

You can use IAM tag key-value pairs to add custom attributes to SAML identity providers. For example, to identify a provider, you can add the tag key **okta** and the tag value **saml**. You can use tags to control access to resources or to control what tags can be attached to an object. To learn more about using tags to control access, see [Controlling access to and for IAM users and roles using tags \(p. 521\)](#).

Permissions required for tagging SAML identity providers

You must configure permissions to allow an IAM entity (users or roles) to tag SAML 2.0-based Identity Providers (IdPs). You can specify one or all of the following IAM tag actions in an IAM policy:

- `iam>ListSAMLProviderTags`
- `iam:TagSAMLProvider`
- `iam:UntagSAMLProvider`

To allow an IAM entity (user or role) to add, list, or remove a tag for a SAML identity provider

Add the following statement to the permissions policy for the IAM entity that needs to manage tags. Use your account number and replace `<SAMLProviderName>` with the name of the SAML provider whose tags need to be managed. To learn how to create a policy using this example JSON policy document, see [the section called "Creating policies using the JSON editor" \(p. 583\)](#).

```
{  
    "Effect": "Allow",  
    "Action": [  
        "iam>ListSAMLProviderTags",  
        "iam:TagSAMLProvider",  
        "iam:UntagSAMLProvider"  
    ],  
    "Resource": "arn:aws:iam::<account-number>:saml-provider/<SAMLProviderName>"  
}
```

To allow an IAM entity (user or role) to add a tag to a specific SAML identity provider

Add the following statement to the permissions policy for the IAM entity that needs to add, but not remove, tags for a specific SAML provider.

Note

The `iam:TagSAMLProvider` action requires that you also include the `iam>ListSAMLProviderTags` action.

To use this policy, replace `<SAMLProviderName>` with the name of the SAML provider whose tags need to be managed. To learn how to create a policy using this example JSON policy document, see [the section called "Creating policies using the JSON editor" \(p. 583\)](#).

```
{  
    "Effect": "Allow",  
    "Action": [  
        "iam>ListSAMLProviderTags",  
        "iam:TagSAMLProvider"  
    ],  
    "Resource": "arn:aws:iam::<account-number>:saml-provider/<SAMLProviderName>"  
}
```

}

Alternatively, you can use an AWS managed policy such as [IAMFullAccess](#) to provide full access to IAM.

Managing tags on IAM SAML identity providers (console)

You can manage tags for IAM SAML Identity Providers from the AWS Management Console.

Note

You can manage tags using the new Identity providers console experience only.

To manage tags on SAML identity providers (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 **Identity providers** and then choose the name of the SAML identity provider that you want to edit.
3. In the **Tags** section, choose **Manage tags** and then complete one of the following actions:
 - Choose **Add tag** if the SAML identity provider does not yet have tags or to add a new tag.
 - Edit existing tag keys and values.
 - Choose **Remove tag** to remove a tag.
4. Add or remove tags to complete the set of tags. Then choose **Save changes**.

Managing tags on IAM SAML identity providers (AWS CLI or AWS API)

You can list, attach, or remove tags for IAM SAML identity providers. You can use the AWS CLI or the AWS API to manage tags for IAM SAML identity providers.

To list the tags currently attached to an SAML identity provider (AWS CLI or AWS API)

- AWS CLI: [aws iam list-saml-provider-tags](#)
- AWS API: [ListSAMLProviderTags](#)

To attach tags to a SAML identity provider (AWS CLI or AWS API)

- AWS CLI: [aws iam tag-saml-provider](#)
- AWS API: [TagSAMLProvider](#)

To remove tags from a SAML identity provider (AWS CLI or AWS API)

- AWS CLI: [aws iam untag-saml-provider](#)
- AWS API: [UntagSAMLProvider](#)

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 \(p. 1298\)](#).

Tagging instance profiles for Amazon EC2 roles

When you launch an Amazon EC2 instance, you specify an IAM role to associate with the instance. An instance profile is a container for an IAM role that you can use to pass role information to an Amazon

EC2 instance when the instance starts. You can tag instance profiles when you use the AWS CLI or AWS API.

You can use IAM tag key-value pairs to add custom attributes to an instance profile. For example, to add department information to an instance profile, you can add the tag key **access-team** and the tag value **eng**. Doing this gives principals with matching tags access to instance profiles with the same tag. You could use multiple tag key-value pairs to specify a team and project: **access-team = eng**, and **project = peg**. You can use tags to control a user's access to resources or to control what tags can be attached to a user. To learn more about using tags to control access, see [Controlling access to and for IAM users and roles using tags \(p. 521\)](#).

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 \(p. 417\)](#).

Permissions required for tagging instance profiles

You must configure permissions to allow an IAM entity (user or role) to tag instance profiles. You can specify one or all of the following IAM tag actions in an IAM policy:

- `iam>ListInstanceProfileTags`
- `iam:TagInstanceProfile`
- `iam:UntagInstanceProfile`

To allow an IAM entity (user or role) to add, list, or remove a tag for an instance profile

Add the following statement to the permissions policy for the IAM entity that needs to manage tags. Use your account number and replace `<InstanceProfileName>` with the name of the instance profile whose tags need to be managed. To learn how to create a policy using this example JSON policy document, see [the section called "Creating policies using the JSON editor" \(p. 583\)](#).

```
{  
    "Effect": "Allow",  
    "Action": [  
        "iam>ListInstanceProfileTags",  
        "iam:TagInstanceProfile",  
        "iam:UntagInstanceProfile"  
    ],  
    "Resource": "arn:aws:iam::<account-number>:instance-profile/<InstanceProfileName>"  
}
```

To allow an IAM entity (user or role) to add a tag to a specific instance profile

Add the following statement to the permissions policy for the IAM entity that needs to add, but not remove, tags for a specific instance profile.

Note

The `iam:TagInstanceProfile` action requires that you also include the `iam>ListInstanceProfileTags` action.

To use this policy, replace `<InstanceProfileName>` with the name of the instance profile whose tags need to be managed. To learn how to create a policy using this example JSON policy document, see [the section called "Creating policies using the JSON editor" \(p. 583\)](#).

```
{  
    "Effect": "Allow",  
    "Action": [  
        "iam>ListInstanceProfileTags",  
        "iam:TagInstanceProfile"  
    ],  
    "Resource": "arn:aws:iam::<account-number>:instance-profile/<InstanceProfileName>"  
}
```

```
    "Resource": "arn:aws:iam::<account-number>:instance-profile/<InstanceProfileName>"  
}
```

Alternatively, you can use an AWS managed policy such as [IAMFullAccess](#) to provide full access to IAM.

Managing tags on instance profiles (AWS CLI or AWS API)

You can list, attach, or remove tags for instance profiles. You can use the AWS CLI or the AWS API to manage tags for instance profiles.

To list the tags currently attached to an instance profile (AWS CLI or AWS API)

- AWS CLI: [aws iam list-instance-profile-tags](#)
- AWS API: [ListInstanceProfileTags](#)

To attach tags to an instance profile (AWS CLI or AWS API)

- AWS CLI: [aws iam tag-instance-profile](#)
- AWS API: [TagInstanceProfile](#)

To remove tags from an instance profile (AWS CLI or AWS API)

- AWS CLI: [aws iam untag-instance-profile](#)
- AWS API: [UntagInstanceProfile](#)

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 \(p. 1298\)](#).

Tagging server certificates

If you use IAM to manage SSL/TLS certificates, you can tag server certificates in IAM using the AWS CLI or AWS API. For certificates in a Region supported by AWS Certificate Manager (ACM), we recommend that you use ACM instead of IAM 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*.

You can use IAM tag key-value pairs to add custom attributes to a server certificate. For example, to add information about the owner or administrator of a server certificate, add the tag key **owner** and the tag value **net-eng**. Or you can specify a cost center by adding the tag key **CostCenter** and the tag value **1234**. You can use tags to control access to resources or to control what tags can be attached to resources. To learn more about using tags to control access, see [Controlling access to and for IAM users and roles using tags \(p. 521\)](#).

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 \(p. 417\)](#).

Permissions required for tagging server certificates

You must configure permissions to allow an IAM entity (user or role) to tag server certificates. You can specify one or all of the following IAM tag actions in an IAM policy:

- **iam>ListServerCertificateTags**

- iam:TagServerCertificate
- iam:UntagServerCertificate

To allow an IAM entity (user or role) to add, list, or remove a tag for a server certificate

Add the following statement to the permissions policy for the IAM entity that needs to manage tags. Use your account number and replace `<CertificateName>` with the name of the server certificate whose tags need to be managed. To learn how to create a policy using this example JSON policy document, see [the section called "Creating policies using the JSON editor" \(p. 583\)](#).

```
{  
    "Effect": "Allow",  
    "Action": [  
        "iam>ListServerCertificateTags",  
        "iam:TagServerCertificate",  
        "iam:UntagServerCertificate"  
    ],  
    "Resource": "arn:aws:iam::<account-number>:server-certificate/<CertificateName>"  
}
```

To allow an IAM entity (user or role) to add a tag to a specific server certificate

Add the following statement to the permissions policy for the IAM entity that needs to add, but not remove, tags for a specific server certificate.

Note

The `iam:TagServerCertificate` action requires that you also include the `iam>ListServerCertificateTags` action.

To use this policy, replace `<CertificateName>` with the name of the server certificate whose tags need to be managed. To learn how to create a policy using this example JSON policy document, see [the section called "Creating policies using the JSON editor" \(p. 583\)](#).

```
{  
    "Effect": "Allow",  
    "Action": [  
        "iam>ListServerCertificateTags",  
        "iam:TagServerCertificate"  
    ],  
    "Resource": "arn:aws:iam::<account-number>:server-certificate/<CertificateName>"  
}
```

Alternatively, you can use an AWS managed policy such as [IAMFullAccess](#) to provide full access to IAM.

Managing tags on server certificates (AWS CLI or AWS API)

You can list, attach, or remove tags for server certificates. You can use the AWS CLI or the AWS API to manage tags for server certificates.

To list the tags currently attached to a server certificate (AWS CLI or AWS API)

- AWS CLI: [aws iam list-server-certificate-tags](#)
- AWS API: [ListServerCertificateTags](#)

To attach tags to a server certificate(AWS CLI or AWS API)

- AWS CLI: [aws iam tag-server-certificate](#)
- AWS API: [TagServerCertificate](#)

To remove tags from a server certificate (AWS CLI or AWS API)

- AWS CLI: [aws iam untag-server-certificate](#)
- AWS API: [UntagServerCertificate](#)

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 \(p. 1298\)](#).

Tagging virtual MFA devices

You can use IAM tag key-value pairs to add custom attributes to a virtual MFA device. For example, to add cost center information for a user's virtual MFA device, you can add the tag key **CostCenter** and the tag value **1234**. You can use tags to control access to resources or to control what tags can be attached to an object. To learn more about using tags to control access, see [Controlling access to and for IAM users and roles using tags \(p. 521\)](#).

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 \(p. 417\)](#).

Permissions required for tagging virtual MFA devices

You must configure permissions to allow an IAM entity (user or role) to tag virtual MFA devices. You can specify one or all of the following IAM tag actions in an IAM policy:

- `iam>ListMFADeviceTags`
- `iam:TagMFADevice`
- `iam:UntagMFADevice`

To allow an IAM entity (user or role) to add, list, or remove a tag for a virtual MFA device

Add the following statement to the permissions policy for the IAM entity that needs to manage tags. Use your account number and replace `<MFATokenID>` with the name of the virtual MFA device whose tags need to be managed. To learn how to create a policy using this example JSON policy document, see [the section called "Creating policies using the JSON editor" \(p. 583\)](#).

```
{  
    "Effect": "Allow",  
    "Action": [  
        "iam>ListMFADeviceTags",  
        "iam:TagMFADevice",  
        "iam:UntagMFADevice"  
    ],  
    "Resource": "arn:aws:iam::<account-number>:mfa/<MFATokenID>"  
}
```

To allow an IAM entity (user or role) to add a tag to a specific virtual MFA device

Add the following statement to the permissions policy for the IAM entity that needs to add, but not remove, tags for a specific MFA device.

Note

The `iam:TagMFADevice` action requires that you also include the `iam>ListMFADeviceTags` action.

To use this policy, replace `<MFATokenID>` with the name of the virtual MFA device whose tags need to be managed. To learn how to create a policy using this example JSON policy document, see [the section called "Creating policies using the JSON editor" \(p. 583\)](#).

```
{  
    "Effect": "Allow",  
    "Action": [  
        "iam>ListMFADeviceTags",  
        "iam:TagMFADevice"  
    ],  
    "Resource": "arn:aws:iam:<account-number>:mfa/<MFATokenID>"  
}
```

Alternatively, you can use an AWS managed policy such as [IAMFullAccess](#) to provide full access to IAM.

Managing tags on virtual MFA devices (AWS CLI or AWS API)

You can list, attach, or remove tags for a virtual MFA device. You can use the AWS CLI or the AWS API to manage tags for a virtual MFA device.

To list the tags currently attached to a virtual MFA device (AWS CLI or AWS API)

- AWS CLI: [aws iam list-mfa-device-tags](#)
- AWS API: [ListMFADeviceTags](#)

To attach tags to a virtual MFA device (AWS CLI or AWS API)

- AWS CLI: [aws iam tag-mfa-device](#)
- AWS API: [TagMFADevice](#)

To remove tags from a virtual MFA device (AWS CLI or AWS API)

- AWS CLI: [aws iam untag-mfa-device](#)
- AWS API: [UntagMFADevice](#)

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 \(p. 1298\)](#).

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 AWS 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 \(p. 1279\)](#) includes the [aws:PrincipalTag \(p. 1352\)](#) 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** – The tags passed when you assume the role or federate the user using the AWS CLI or AWS API. For more information about these operations, see [Session tagging operations \(p. 418\)](#).

2. **Incoming transitive session tags** – The tags inherited from a previous session in a role chain. For more information, see [Chaining roles with session tags \(p. 424\)](#) later in this topic.
3. **IAM tags** – The tags attached to your IAM assumed role.

Topics

- [Session tagging operations \(p. 418\)](#)
- [Things to know about session tags \(p. 419\)](#)
- [Permissions required to add session tags \(p. 419\)](#)
- [Passing session tags using AssumeRole \(p. 421\)](#)
- [Passing session tags using AssumeRoleWithSAML \(p. 422\)](#)
- [Passing session tags using AssumeRoleWithWebIdentity \(p. 423\)](#)
- [Passing session tags using GetFederationToken \(p. 423\)](#)
- [Chaining roles with session tags \(p. 424\)](#)
- [Using session tags for ABAC \(p. 425\)](#)
- [Viewing session tags in CloudTrail \(p. 425\)](#)

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 \(p. 282\)](#) 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 \(p. 424\)](#).

Comparing methods for passing session tags

Operation	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 a 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 under the following conditions:

- You pass more than 50 session tags.

- The plaintext of your session tag keys exceeds 128 characters.
- The plaintext of your session tag values exceeds 256 characters.
- The total size of the plaintext of session policies exceeds 2048 characters.
- The total packed size of the combined session policies and tags is too large. If the operation fails, the error message shows how close the policies and tags combined come to the upper size limit, by percentage.

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) passing tags must have the [sts:TagSession \(p. 419\)](#) permission. For roles without this permission in the trust policy, the AssumeRole operation fails.
- When you request a session, you can specify principal tags as the session tags. The tags apply to requests that you make using the session's credentials.
- Session tags use 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 \(p. 401\)](#). 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 \(p. 185\)](#). By default, AWS STS does not pass tags to subsequent role sessions. However, you can set session tags as transitive. Transitive tags persist during role chaining and replace matching ResourceTag values during the evaluation of the role trust policy. For more information, see [Chaining roles with session tags \(p. 424\)](#).
- 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 \(p. 63\)](#).
- You can view the principal tags for your session, including the session tags, in the AWS CloudTrail logs. For more information, see [Viewing session tags in CloudTrail \(p. 425\)](#).
- You must pass a single value for each session tag. AWS STS does not support multi-valued session tags.
- 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 AWS STS quotas \(p. 1220\)](#).
- An AWS conversion compresses the passed session policies and session tags combined into a packed binary format with a separate limit. If you exceed this limit, the AWS CLI or AWS API error message shows how close the policies and tags combined come to the upper size limit, by percentage.

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 fails for any role connected to an IdP 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 connected to the separate IdP.

You can use the `sts:TagSession` action with the following condition keys.

- [aws:PrincipalTag \(p. 1352\)](#) – Compares the tag attached to the principal making the request with the tag you specified 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 \(p. 1355\)](#) – Compares the tag key-value pair passed in the request with the tag pair you specified in the policy. For example, you can allow the principal to pass the specified session tags, but only with the specified values.
- [aws:ResourceTag \(p. 1361\)](#) – Compares the tag key-value pair you specified in the policy with the key-value pair attached to the resource. For example, you can allow the principal to pass session tags only if the role they assume includes the specified tags.
- [aws:TagKeys \(p. 1364\)](#) – Compares the tag keys in a request with the keys you specified 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 \(p. 1383\)](#) – Compares 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 \(p. 424\)](#).

For example, the following [role trust policy \(p. 186\)](#) allows the `test-session-tags` user to assume the role with the attached policy. 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 \(p. 191\)](#). 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/Project": "*",
                    "aws:RequestTag/CostCenter": "*",
                    "aws:RequestTag/Department": "*"
                },
                "StringEquals": {"sts:ExternalId": "Example987"}
            }
        },
        {
            "Sid": "AllowPassSessionTagsAndTransitive",
            "Effect": "Allow",
            "Action": "sts:TagSession",
            "Principal": {"AWS": "arn:aws:iam::123456789012:user/test-session-tags"},
            "Condition": {
                "StringLike": {
                    "aws:RequestTag/Project": "*",
                    "aws:RequestTag/CostCenter": "*"
                },
                "StringEquals": {
                    "aws:RequestTag/Department": [
                        "Engineering",
                        "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 with the attached policy. When that user assumes the role, they must pass the required session tags and [external ID \(p. 191\)](#).
- The first condition block of this statement requires the user to pass the Project, CostCenter, and Department session tags. The tag values do not matter in this statement, so you can use wildcards (*) for the tag values. This block ensures that user passes at least these three session tags. Otherwise, the operation fails. The user can pass additional tags.
- The second condition block requires the user to pass an [external ID \(p. 191\)](#) 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 \(p. 1282\)](#) 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 you can set as transitive. The user can choose to set a subset or no tags as transitive. 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 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 \(p. 424\)](#).

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 authenticates with SAML-based federation. This operation returns a set of temporary credentials 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 \(p. 230\)](#). For details about AWS CLI or AWS API access, see [About SAML 2.0-based federation \(p. 205\)](#). For a tutorial on configuring 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 \(p. 223\)](#).
2. [Create a SAML provider in IAM \(p. 218\)](#)
3. [Configure a role and permissions in AWS for your federated users \(p. 268\)](#)
4. [Finish configuring the SAML IdP and create assertions for the SAML authentication response \(p. 225\)](#)

AWS includes identity providers with certified 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 \(p. 223\)](#).

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 preceding tags 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 \(p. 424\)](#).

To set the Project and Department tags as transitive, use the following multi-valued 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

Use OpenID Connect(OIDC)-compliant web identity federation to authenticate the AssumeRoleWithWebIdentity operation. This operation returns a set of temporary credentials 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 \(p. 199\)](#).

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 uses a token 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 \(p. 424\)](#).

Example Example decoded JSON Web Token

```
{
  "sub": "johndoe",
  "aud": "ac_oic_client",
  "jti": "ZYUCeRMQVtqHypVPWAN3VB",
  "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 allows you to federate your user. This operation returns a set of temporary credentials 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, because you can't use the temporary credentials to assume a role. You cannot use role chaining in this case.

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 returned by the GetFederationToken operation, the session principal tags include the user 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 \(p. 185\)](#). 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 shows how AWS STS passes session tags, transitive tags, and role tags into subsequent sessions in a role chain.

In this example role chaining scenario, you use an IAM user access key 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 command also attaches the tag Heart = 1 to the role and applies 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. The resulting session principal tags include both of these tags, and sets them 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 command attaches the tag Sun = 2 to the second role and applies as a principal tag when you use the second session. The Heart and Star tags inherit the transitive session tags in the first session. The second session resulting principal tags are Heart = 1, Star = 1, and Sun = 2. Heart and Star continue to be transitive. The Sun tag attached to Role2 is not marked as transitive because it is not a session tag. Future sessions do not inherit this tag.

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 are inherited from the transitive session tag in the first session. If you try to pass the Sun = 2 session tag, the operation fails. The inherited Star = 1 session tag overrides the role Star = 3 tag. In role chaining, the value of a transitive tag overrides the role matching the ResourceTag value before the evaluation of the role trust policy. In this example, if Role3 uses Star as a ResourceTag in the role trust policy, and sets ResourceTag value to the transitive tag value from the calling role session. The role Lightning tag also applies to the third session, and 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) uses an authorization strategy that defines permissions based on tag attributes.

If your company uses an OIDC or SAML-based identity provider (IdP) to manage user identities, you can configure your assertion to pass session tags to AWS. For example, with corporate user identities, when your employees federate into AWS, AWS applies their attributes to their resulting principal. 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 \(p. 63\)](#).

For more information about using IAM Identity Center with ABAC, see [Attributes for access control](#) in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*.

Viewing session tags in CloudTrail

You can use AWS CloudTrail to view the requests used 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 \(p. 471\)](#).

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 \(p. 476\)](#)
- [Example SAML AWS STS API event in CloudTrail log file \(p. 479\)](#)
- [Example web identity AWS STS API event in CloudTrail log file \(p. 480\)](#)

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 long-term access key credentials, 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.

As a result, temporary credentials have the following advantages over long-term 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 \(p. 183\)](#).
- 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 \(p. 461\)](#).

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 \(p. 198\)](#).

- **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 different sign-in credentials. This is known as the *single sign-on* 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 \(p. 205\)](#).
- **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 \(p. 233\)](#).
- **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 \(p. 205\)](#).
- **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 \(p. 199\)](#).

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 AWS SDKs for mobile development 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 [Authentication with Amplify](#) in the *Amplify Documentation*.

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 \(p. 251\)](#). 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 \(p. 374\)](#).

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 \(p. 1224\)](#).

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 \(p. 426\)](#). To learn about the different methods that you can use to request temporary security credentials by assuming a role, see [Using IAM roles \(p. 274\)](#).

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 permissions are the intersection of the role's identity-based policies and the session policies. For more information about session policies, see [Session policies \(p. 487\)](#). For more information about session tags, see [Passing session tags in AWS STS \(p. 417\)](#).

Note

The size of the session 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 method to specify a Region before you make the API call. If you manually construct 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 \(p. 461\)](#).

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 active credentials. To learn who can call this operation, see [Comparing the AWS STS API operations \(p. 436\)](#). For more information, see [Creating a role to delegate permissions to an IAM user \(p. 251\)](#) and [Configuring MFA-protected API access \(p. 145\)](#).

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 \(p. 276\)](#). 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 \(p. 233\)](#).
- Role session name. Use this string value to identify the session when a role is used by different principals. For security purposes, administrators can view this field in [AWS CloudTrail logs \(p. 473\)](#) to help identify 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 [sts:RoleSessionName \(p. 1381\)](#).
- (Optional) Source identity. You can require users to specify a source identity when they assume a role. After the source identity is set, the value cannot be changed. It is present in the request for all actions that are taken during the role session. The source identity value persists across [chained role \(p. 185\)](#) sessions. You can use source identity information in AWS CloudTrail logs to determine who took actions with a role. For more information about using source identity, see [Monitor and control actions taken with assumed roles \(p. 444\)](#).
- (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 \(p. 487\)](#).
- (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 \(p. 417\)](#).
- (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 \(p. 191\)](#).

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 \(p. 487\)](#), [session tags \(p. 417\)](#), [external ID \(p. 191\)](#), and [source identity \(p. 444\)](#). 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*%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
&SourceIdentity=DevUser123
&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>
<SourceIdentity>DevUser123</SourceIdentity>
<Credentials>
<SessionToken>
AQoDYXdzEPT///////////wEXAMPLEt764bNrC9SAPBSM22wD0k4x4HIZ8j4FZTwdQW
LWsKWHGBuFqwAeMicRXmxfpSPfIeoIYRqTf1fKD8YUuwtAx7mSEI/qkPpKPi/kMcGd
QrmGdeehM4IC1NtBmUpp2wUE8phUZampKsburEDy0KPkyQDYwT7WZ0wq5VSxDvp75YU
9HFv1Rd8Tx6q6fE8YQcHNVXAk1Y9q6d+xo0rKwT38xVqr7ZD0u0iPPkUL641I2bqBAz
+scqKmlzm8FDrypNC9Yjc8fp0Ln9FX9KSvKTr4rvx3iSiLTjabIQwj2ICCR/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>AR0123EXAMPLE123:John</AssumedRoleId>
</AssumedRoleUser>
<PackedPolicySize>8</PackedPolicySize>
</AssumeRoleResult>
<ResponseMetadata>
<RequestId>c6104cbe-af31-11e0-8154-ccb7ccf896c7</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 plaintext 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 \(p. 199\)](#).

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 [Authentication with Amplify](#) in the *Amplify Documentation*.

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 \(p. 417\)](#).
- (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 \(p. 276\)](#). 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 \(p. 233\)](#).
- Role session name. Use this string value to identify the session when a role is used by different principals. For security purposes, administrators can view this field in [AWS CloudTrail logs \(p. 473\)](#) 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 [sts:RoleSessionName \(p. 1381\)](#).
- (Optional) Source identity. You can require federated users to specify a source identity when they assume a role. After the source identity is set, the value cannot be changed. It is present in the request for all actions that are taken during the role session. The source identity value persists across [chained role \(p. 185\)](#) sessions. You can use source identity information in AWS CloudTrail logs to determine who took actions with a role. For more information about using source identity, see [Monitor and control actions taken with assumed roles \(p. 444\)](#).
- (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 \(p. 487\)](#).

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 don't use 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 \(p. 205\)](#).

Note

A call to `AssumeRoleWithSAML` 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.

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 \(p. 417\)](#).
- (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 \(p. 276\)](#). 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 \(p. 233\)](#).
- (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 \(p. 487\)](#).
- Role session name. Use this string value to identify the session when a role is used by different principals. For security purposes, administrators can view this field in [AWS CloudTrail logs \(p. 473\)](#) 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 [sts:RoleSessionName \(p. 1381\)](#).

- (Optional) Source identity. You can require federated users to specify a source identity when they assume a role. After the source identity is set, the value cannot be changed. It is present in the request for all actions that are taken during the role session. The source identity value persists across [chained role \(p. 185\)](#) sessions. You can use source identity information in AWS CloudTrail logs to determine who took actions with a role. For more information about using source identity, see [Monitor and control actions taken with assumed roles \(p. 444\)](#).

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 \(p. 225\)](#).

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 \(p. 428\)](#).

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 \(p. 487\)](#).

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 \(p. 417\)](#).

The `GetFederationToken` call returns temporary security credentials that consist of the session 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 \(p. 487\)](#) ARN and [session tags \(p. 417\)](#). 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///////////wEXAMPLEtc764bNrC9SAPBSM22wD0k4x4H1Z8j4FZTwdQW  
LWsKWHGBuFqwAeMicRXmxfpSPfIeoIYRqTf1fKD8YUuwthAx7mSEI/qkPpKPi/kMcGd  
QrmGdeehM4IC1NtBmUpp2wUE8phUZampKsburEDy0KPkyQDYwT7WZ0wq5VSxDvp75YU  
9HFv1Rd8Tx6g6fE8YQcHNVXAkjY9q6d+x0@rKwT38xVqz7ZD0u0iPPkUL641IZbqBAz  
+scqKmlzm8FDrypNC9Yjc8fP0Ln9FX9KSvKTr4rvx3iSiLTjabIQwj2ICCEXAMPLE==  
</SessionToken>  
<SecretAccessKey>  
wJalrXUtnFEMI/K7MDENG/bPxRfiCYzEXAMPLEKEY  
</SecretAccessKey>  
<Expiration>2019-04-15T23:28:33.359Z</Expiration>  
<AccessKeyId>AKIAIOSF0DNN7EXAMPLE;</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 plaintext 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 \(p. 428\)](#) 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 session 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>
AQoEXAMPLEH4aoAH0gNCAPyJxz4B1CFFxWNE10PTgk5TthT+FvwqnKwRc0IfRh3c/L
To6UDdyJw00vEPVpVLXCrriUtdnNiCEXAMPLE/IvU1dYUg2RVAJBanLiHb4IgRmpRV3z
rkuWJ0gQs8IZZaIv2BXIa2R40lgkBN9bkUDNCJiBeb/AX1zBBko7b15fjrBs2+cTQtp
Z3CYWFxG8C5zqx37wn0E49mR1/+0tkIKG07fAE
</SessionToken>
<SecretAccessKey>
wJalrXUtnFEMI/K7MDENG/bPxRfiCYzEXAMPLEKEY
</SecretAccessKey>
<Expiration>2011-07-11T19:55:29.611Z</Expiration>
<AccessKeyId>AKIAIOSFODNN7EXAMPLE</AccessKeyId>
</Credentials>
</GetSessionTokenResult>
<ResponseMetadata>
<RequestId>58c5dbae-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 \(p. 274\)](#). To learn about the different AWS STS API operations that allow you to pass session tags, see [Passing session tags in AWS STS \(p. 417\)](#).

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 GetFederationToken or GetSessionToken.
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 GetFederationToken or GetSessionToken.
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 GetFederationToken or GetSessionToken.
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. This limitation does not apply to console sessions. Cannot call AWS STS operations except GetCallerIdentity. 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 AssumeRole or GetCallerIdentity. 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 \(p. 145\)](#).

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 \(p. 487\)](#).

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 \(p. 276\)](#).

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 \(p. 1172\)](#).

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 \(p. 230\)](#) 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 \(p. 233\)](#).

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 as 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.
- Temporary credentials expire after a specified interval. After temporary credentials expire, any calls that you make with those credentials will fail, so you must generate a new set of temporary credentials. Temporary credentials cannot be extended or refreshed beyond the original specified interval.
- 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 \(p. 417\)](#).

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 \(p. 426\)](#). AWS STS is a global service that has a default endpoint at <https://sts.amazonaws.com>. This endpoint is in the US East (N. Virginia) 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 \(p. 461\)](#).

Contents

- [Using temporary credentials in Amazon EC2 instances \(p. 439\)](#)
- [Using temporary security credentials with the AWS SDKs \(p. 439\)](#)
- [Using temporary security credentials with the AWS CLI \(p. 440\)](#)
- [Using temporary security credentials with API operations \(p. 440\)](#)
- [More information \(p. 441\)](#)

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 (IMDS) 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 \(AWS API\) \(p. 291\)](#). 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=ASIAIOSFODNN7EXAMPLE
$ export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
$ export AWS_SESSION_TOKEN=AQoDYXdzEJr...<remainder of session token>
$ aws ec2 describe-instances --region us-west-1
```

Windows

```
C:\> SET AWS_ACCESS_KEY_ID=ASIAIOSFODNN7EXAMPLE
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 User Guide* .
- **Amazon SNS.** See [Using Temporary Security Credentials](#) in the *Amazon Simple Notification Service Developer Guide*.
- **Amazon SQS.** See [Using identity-based policies with Amazon SNS](#) in the *Amazon Simple Queue Service Developer Guide*.
- **Amazon SimpleDB.** See [Identity and access management in Amazon SQS](#) 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 \(p. 426\)](#). 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 \(p. 484\)](#).

Topics

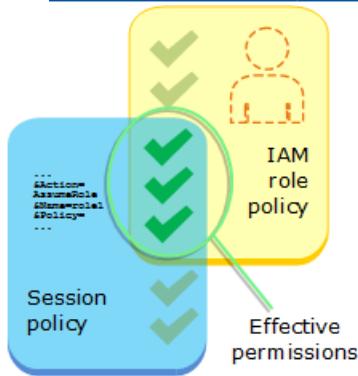
- [Permissions for AssumeRole, AssumeRoleWithSAML, and AssumeRoleWithWebIdentity \(p. 441\)](#)
- [Monitor and control actions taken with assumed roles \(p. 444\)](#)
- [Permissions for GetFederationToken \(p. 452\)](#)
- [Permissions for GetSessionToken \(p. 455\)](#)
- [Disabling permissions for temporary security credentials \(p. 456\)](#)
- [Granting permissions to create temporary security credentials \(p. 460\)](#)

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 \(p. 487\)](#) 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 \(p. 1306\)](#).



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 \(p. 487\)](#).

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 \(p. 1306\)](#).

Monitor and control actions taken with assumed roles

An [IAM role \(p. 183\)](#) is an object in IAM that is assigned [permissions \(p. 485\)](#). When you [assume that role \(p. 274\)](#) using an IAM identity or an identity from outside of AWS, you receive a session with the permissions that are assigned to the role.

When you perform actions in AWS, the information about your session can be logged to AWS CloudTrail for your account administrator to monitor. Administrators can configure roles to require identities to pass a custom string that identifies the person or application that is performing actions in AWS. This identity information is stored as the *source identity* in AWS CloudTrail. When the administrator reviews activity in CloudTrail, they can view the source identity information to determine who or what performed actions with assumed role sessions.

After a source identity is set, it is present in requests for any AWS action taken during the role session. The value that is set persists when a role is used to assume another role through the AWS CLI or AWS API, known as [role chaining \(p. 185\)](#). The value that is set cannot be changed during the role session. Administrators can configure granular permissions based on the presence or value of the source identity to further control AWS actions that are taken with shared roles. You can decide whether the source identity attribute can be used, whether it is required, and what value can be used.

The way that you use source identity differs from role session name and session tags in an important way. The source identity value can't be changed after it is set, and it persists for any additional actions that are taken with the role session. Here's how you can use session tags and role session name:

- **Session tags** – You can pass session tags when you assume a role or federate a user. Session tags are present when a role is assumed. You can define policies that use tag condition keys to grant permissions to your principals based on their tags. Then you can use CloudTrail to view the requests made to assume roles or federate users. To learn more about session tags, see [Passing session tags in AWS STS \(p. 417\)](#).
- **Role session name** – You can use the `sts:RoleSessionName` condition key in a role trust policy to require that your users provide a specific session name when they assume a role. Role session name can be used to differentiate role sessions when a role is used by different principals. To learn more about role session name, see [sts:RoleSessionName \(p. 1381\)](#).

We recommend that you use source identity when you want to control the identity that assumes a role. Source identity is also useful for mining CloudTrail logs to determine who used the role to perform actions.

Topics

- [Setting up to use source identity \(p. 445\)](#)
- [Things to know about source identity \(p. 445\)](#)
- [Permissions required to set source identity \(p. 446\)](#)
- [Specifying a source identity when assuming a role \(p. 447\)](#)
- [Using source identity with AssumeRole \(p. 447\)](#)
- [Using source identity with AssumeRoleWithSAML \(p. 448\)](#)

- [Using source identity with AssumeRoleWithWebIdentity \(p. 448\)](#)
- [Control access using source identity information \(p. 449\)](#)
- [Viewing source identity in CloudTrail \(p. 451\)](#)

Setting up to use source identity

The way that you set up to use source identity depends on the method used when your roles are assumed. For example, your IAM users might assume roles directly using the AssumeRole operation. If you have enterprise identities, also known as workforce identities, they might access your AWS resources using AssumeRoleWithSAML. If end users access your mobile or web applications, they might do so using AssumeRoleWithWebIdentity. The following is a high-level workflow overview to help you understand how you can set up to utilize source identity information in your existing environment.

1. **Configure test users and roles** – Using a preproduction environment, configure test users and roles and configure their policies to allow setting a source identity.

If you use an identity provider (IdP) for your federated identities, configure your IdP to pass a user attribute of your choice for source identity in the assertion or token.

2. **Assume the role** – Test assuming roles and passing a source identity with the users and roles that you set up for testing.
3. **Review CloudTrail** – Review the source identity information for your test roles in your CloudTrail logs.
4. **Train your users** – After you've tested in your preproduction environment, ensure that your users know how to pass in the source identity information, if necessary. Set a deadline for when you will require your users to provide a source identity in your production environment.
5. **Configure production policies** – Configure your policies for your production environment, and then add them to your production users and roles.
6. **Monitor activity** – Monitor your production role activity using CloudTrail logs.

Things to know about source identity

Keep the following in mind when working with source identity.

- Trust policies for all roles connected to an identity provider (IdP) must have the sts:SetSourceIdentity permission. For roles that don't have this permission in the role trust policy, the AssumeRole* operation will fail. If you don't want to update the role trust policy for each role, you can use a separate IdP instance for passing source identity. Then add the sts:SetSourceIdentity permission to only the roles that are connected to the separate IdP.
- When an identity sets a source identity, the sts:SourceIdentity key is present in the request. For subsequent actions taken during the role session, the aws:SourceIdentity key is present in the request. AWS doesn't control the value of the source identity in either the sts:SourceIdentity or aws:SourceIdentity keys. If you choose to require a source identity, you must choose an attribute that you want your users or IdP to provide. For security purposes, you must ensure that you can control how those values are provided.
- The value of source identity must be between 2 and 64 characters long, can contain only alphanumeric characters, underscores, and the following characters: . , + = @ - (hyphen). You cannot use a value that begins with the text **aws:**. This prefix is reserved for AWS internal use.
- The source identity information is not captured by CloudTrail when an AWS service or service-linked role carries out an action on behalf of a federated or workforce identity.

Important

You cannot switch to a role in the AWS Management Console that requires a source identity to be set when the role is assumed. To assume such a role, you can use the AWS CLI or AWS API to call the AssumeRole operation and specify the source identity parameter.

Permissions required to set source identity

In addition to the action that matches the API operation, you must have the following permissions-only action in your policy:

```
sts:SetSourceIdentity
```

- To specify a source identity, principals (IAM users and roles) must have permissions to `sts:SetSourceIdentity`. As the administrator, you can configure this in the role trust policy and in the principal's permissions policy.
- When you assume a role with another role, called [role chaining \(p. 185\)](#), permissions for `sts:SetSourceIdentity` are required in both the permissions policy of the principal who is assuming the role and in the role trust policy of the target role. Otherwise, the assume role operation will fail.
- When using source identity, the role trust policies for all roles connected to an IdP must have the `sts:SetSourceIdentity` permission. The `AssumeRole*` operation will fail for any role connected to an IdP 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 source identity and add the `sts:SetSourceIdentity` permission to only the roles that are connected to the separate IdP.
- To set a source identity across account boundaries, you must include the `sts:SetSourceIdentity` permission in two places. It must be in the permissions policy of the principal in the originating account and in the role trust policy of the role in the target account. You might need to do this, for example, when a role is used to assume a role in another account with [role chaining \(p. 185\)](#).

As the account administrator, imagine that you want to allow the IAM user DevUser in your account to assume the `Developer_Role` in the same account. But you want to allow this action only if the user has set the source identity to their IAM user name. You can attach the following policy to the IAM user.

Example Example identity-based policy attached to DevUser

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AssumeRole",
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Resource": "arn:aws:iam::123456789012:role/Developer_Role"
        },
        {
            "Sid": "SetAwsUserNameAsSourceIdentity",
            "Effect": "Allow",
            "Action": "sts:SetSourceIdentity",
            "Resource": "arn:aws:iam::123456789012:role/Developer_Role",
            "Condition": {
                "StringLike": {
                    "sts:SourceIdentity": "${aws:username}"
                }
            }
        }
    ]
}
```

To enforce the acceptable source identity values, you can configure the following role trust policy. The policy gives the IAM user DevUser permissions to assume the role and set a source identity. The `sts:SourceIdentity` condition key defines the acceptable source identity value.

Example Example role trust policy for source identity

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowDevUserAssumeRole",  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": "arn:aws:iam::123456789012:user/DevUser"  
            },  
            "Action": [  
                "sts:AssumeRole",  
                "sts:SetSourceIdentity"  
            ],  
            "Condition": {  
                "StringEquals": {  
                    "sts:SourceIdentity": "DevUser"  
                }  
            }  
        }  
    ]  
}
```

Using the credentials for the IAM user DevUser, the user attempts to assume the DeveloperRole using the following AWS CLI request.

Example Example AssumeRole CLI request

```
aws sts assume-role \  
--role-arn arn:aws:iam::123456789012:role/Developer_Role \  
--role-session-name Dev-project \  
--source-identity DevUser \  

```

When AWS evaluates the request, the request context contains the sts:SourceIdentity of DevUser.

Specifying a source identity when assuming a role

You can specify a source identity when you use one of the AWS STS AssumeRole* API operations to get temporary security credentials for a role. The API operation that you use differs depending on your use case. For example, if you use IAM roles to give IAM users access to AWS resources that they don't normally have access to, you might use the AssumeRole operation. If you use enterprise identity federation to manage your workforce users, you might use the AssumeRoleWithSAML operation. If you use web identity federation to allow end users to access your mobile or web applications, you might use the AssumeRoleWithWebIdentity operation. The following sections explain how to use source identity with each operation. To learn more about common scenarios for temporary credentials, see [Common scenarios for temporary credentials \(p. 426\)](#).

Using source identity with 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 source identity while assuming a role, use the --source-identity AWS CLI option or the SourceIdentity AWS API parameter. The following example shows how to specify the source identity using the AWS CLI.

Example Example AssumeRole CLI request

```
aws sts assume-role \  
--role-arn arn:aws:iam::123456789012:role/developer \  
--role-session-name Audit \  
--source-identity Admin \  

```

Using source identity with AssumeRoleWithSAML

The principal calling 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 \(p. 230\)](#). For details about AWS CLI or AWS API access, see [About SAML 2.0-based federation \(p. 205\)](#). 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 a SAML provider in your organization \(p. 223\)](#).
2. [Create a SAML provider in IAM \(p. 218\)](#).
3. [Configure a role and its permissions in AWS for your federated users \(p. 268\)](#).
4. [Finish configuring the SAML IdP and create assertions for the SAML authentication response \(p. 225\)](#).

To set a SAML attribute for source identity, include the Attribute element with the Name attribute set to `https://aws.amazon.com/SAML/Attributes/SourceIdentity`. Use theAttributeValue element to specify the value of the source identity. For example, assume that you want to pass the following identity attribute as the source identity.

`SourceIdentity:DiegoRamirez`

To pass this attribute, include the following element in your SAML assertion.

Example Example snippet of a SAML assertion

```
<Attribute Name="https://aws.amazon.com/SAML/Attributes/SourceIdentity">
<AttributeValue>DiegoRamirez</AttributeValue>
</Attribute>
```

Using source identity with AssumeRoleWithWebIdentity

The principal calling 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 \(p. 199\)](#).

To pass source identity from OpenID Connect (OIDC), you must include the source identity in the JSON Web Token (JWT). Include source identity in the `https://aws.amazon.com/` source_identity 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 Admin source identity.

Example Example decoded JSON Web Token

```
{
  "sub": "johndoe",
  "aud": "ac_oic_client",
  "jti": "ZYUCeRMQVtqHypVPWAN3VB",
  "iss": "https://xyz.com",
  "iat": 1566583294,
```

```

    "exp": 1566583354,
    "auth_time": 1566583292,
    "https://aws.amazon.com/source_identity":"Admin"
}

```

Control access using source identity information

When a source identity is initially set, the [sts:SourceIdentity \(p. 1382\)](#) key is present in the request. After a source identity is set, the [aws:SourceIdentity \(p. 1362\)](#) key is present in all subsequent requests made during the role session. As the administrator, you can write policies that grant conditional authorization to perform AWS actions based on the existence or value of the source identity attribute.

Imagine that you want to require your developers to set a source identity to assume a critical role that has permission to write to a production critical AWS resource. Also imagine that you grant AWS access to your workforce identities using AssumeRoleWithSAML. You only want senior developers Saanvi and Diego to have access to the role, so you create the following trust policy for the role.

Example Example role trust policy for source identity (SAML)

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "SAMLProviderAssumeRoleWithSAML",
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::111122223333:saml-provider/name-of-identity-provider"
            },
            "Action": [
                "sts:AssumeRoleWithSAML"
            ],
            "Condition": {
                "StringEquals": {
                    "SAML:aud": "https://signin.aws.amazon.com/saml"
                }
            }
        },
        {
            "Sid": "SetSourceIdentitySrEngs",
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::111122223333:saml-provider/name-of-identity-provider"
            },
            "Action": [
                "sts:SetSourceIdentity"
            ],
            "Condition": {
                "StringLike": {
                    "sts:SourceIdentity": [
                        "Saanvi",
                        "Diego"
                    ]
                }
            }
        }
    ]
}

```

The trust policy contains a condition for `sts:SourceIdentity` that requires a source identity of Saanvi or Diego to assume the critical role.

Alternatively, if you use an OIDC provider for web identity federation and users are authenticated with `AssumeRoleWithWebIdentity`, your role trust policy might look as follows.

Example Example role trust policy for source identity (OIDC provider)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::111122223333:oidc-provider/server.example.com"
      },
      "Action": [
        "sts:AssumeRoleWithWebIdentity",
        "sts:SetSourceIdentity"
      ],
      "Condition": {
        "StringEquals": {
          "server.example.com:aud": "oidc-audience-id"
        },
        "StringLike": {
          "sts:SourceIdentity": [
            "Saanvi",
            "Diego"
          ]
        }
      }
    }
  ]
}
```

Role chaining and cross-account requirements

Imagine that you want to allow users who have assumed `CriticalRole` to assume a `CriticalRole_2` in another account. The role session credentials that were obtained to assume `CriticalRole` are used to [role chain \(p. 185\)](#) to a second role, `CriticalRole_2`, in a different account. The role is being assumed across an account boundary. Therefore, the `sts:SetSourceIdentity` permission must be granted in both the permissions policy on `CriticalRole` and in the role trust policy on `CriticalRole_2`.

Example Example permissions policy on CriticalRole

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AssumeRoleAndSetSourceIdentity",
      "Effect": "Allow",
      "Action": [
        "sts:AssumeRole",
        "sts:SetSourceIdentity"
      ],
      "Resource": "arn:aws:iam::222222222222:role/CriticalRole_2"
    }
  ]
}
```

To secure setting source identity across the account boundary, the following role trust policy trusts only the role principal for `CriticalRole` to set the source identity.

Example Example role trust policy on CriticalRole_2

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

```

"Statement": [
    {
        "Effect": "Allow",
        "Principal": {
            "AWS": "arn:aws:iam::111111111111:role/CriticalRole"
        },
        "Action": [
            "sts:AssumeRole",
            "sts:SetSourceIdentity"
        ],
        "Condition": {
            "StringLike": {
                "aws:SourceIdentity": ["Saanvi","Diego"]
            }
        }
    }
]
}

```

The user makes the following call using role session credentials obtained from assuming CriticalRole. The source identity was set during the assumption of CriticalRole, so it does not need to be explicitly set again. If the user attempts to set a source identity that is different from the value set when CriticalRole was assumed, the assume role request will be denied.

Example Example AssumeRole CLI request

```

aws sts assume-role \
--role-arn arn:aws:iam::222222222222:role/CriticalRole_2 \
--role-session-name Audit \

```

When the calling principal assumes the role, the source identity in the request persists from the first assumed role session. Therefore, both the aws:SourceIdentity and sts:SourceIdentity keys are present in the request context.

Viewing source identity in CloudTrail

You can use CloudTrail to view the requests made to assume roles or federate users. You can also view the role or user requests to take actions in AWS. The CloudTrail log file includes information about the source identity set for the assumed-role or federated user session. For more information, see [Logging IAM and AWS STS API calls with AWS CloudTrail \(p. 471\)](#)

For example, assume that a user makes an AWS STS AssumeRole request, and sets a source identity. You can find the sourceIdentity information in the requestParameters key in your CloudTrail log.

Example Example requestParameters section in an AWS CloudTrail log

```

"eventVersion": "1.05",
"userIdentity": {
    "type": "AWSAccount",
    "principalId": "AIDAJ45Q7YFFAREEXAMPLE",
    "accountId": "111122223333"
},
"eventTime": "2020-04-02T18:20:53Z",
"eventSource": "sts.amazonaws.com",
"eventName": "AssumeRole",
"awsRegion": "us-east-1",
"sourceIPAddress": "203.0.113.64",
"userAgent": "aws-cli/1.16.96 Python/3.6.0 Windows/10 botocore/1.12.86",
"requestParameters": {
    "roleArn": "arn:aws:iam::123456789012:role/DevRole",
    "roleSessionName": "Dev1",
}

```

```
        "sourceIdentity": "source-identity-value-set"
    }
```

If the user uses the assumed role session to perform an action, the source identity information is present in the `userIdentity` key in the CloudTrail log.

Example Example `userIdentity` key in an AWS CloudTrail log

```
{
  "eventVersion": "1.08",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AROAJ45Q7YFFAREXAMPLE:Dev1",
    "arn": "arn:aws:sts::123456789012:assumed-role/DevRole/Dev1",
    "accountId": "123456789012",
    "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AROAJ45Q7YFFAREXAMPLE",
        "arn": "arn:aws:iam::123456789012:role/DevRole",
        "accountId": "123456789012",
        "userName": "DevRole"
      },
      "webIdFederationData": {},
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2021-02-21T23:46:28Z"
      }
    },
    "sourceIdentity": "source-identity-value-present"
  }
}
```

To see example AWS STS API events in CloudTrail logs, see [Example IAM API events in CloudTrail log \(p. 474\)](#). For more details about the information contained in CloudTrail log files, see [CloudTrail Event Reference](#) in the *AWS CloudTrail User Guide*.

Permissions for GetFederationToken

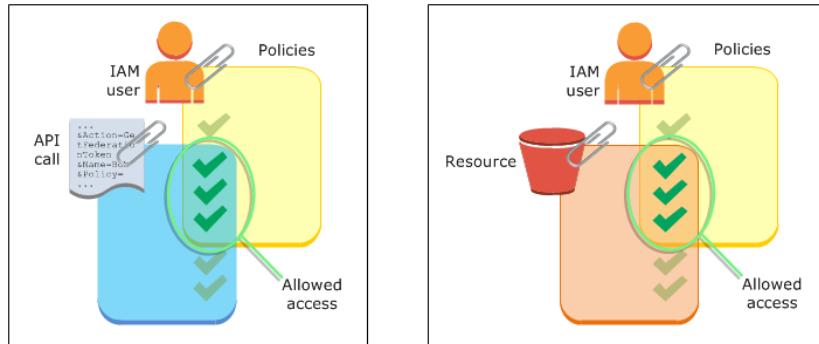
The `GetFederationToken` operation is called by an IAM user and returns temporary credentials for that user. This operation *federates* the user. The permissions assigned a federated user are defined in one of two places:

- 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 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": "*"
    }
  ]
}
```

```

        "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 \(p. 1224\)](#) 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. They can also be granted (within the account) 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:federated-user/Carol"},  
         "Effect": "Allow",  
         "Action": [  
             "s3:GetObject",  
             "s3:PutObject",  
             "s3:DeleteObject"  
         ],  
         "Resource": ["arn:aws:s3:::productionapp/*"]  
    }  
}
```

For more information about how policies are evaluated see [Policy evaluation logic \(p. 1306\)](#).

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 \(p. 1032\)](#) 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 \(p. 233\)](#).
- 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 \(p. 436\)](#)

For more information about MFA-protected API access using `GetSessionToken`, see [Configuring MFA-protected API access \(p. 145\)](#).

Disabling permissions for temporary security credentials

Temporary security credentials are valid until they expire. These credentials are valid for the specified duration, from 900 seconds (15 minutes) up to a maximum of 129,600 seconds (36 hours). The default session duration is 43,200 seconds (12 hours). You can revoke these credentials, but you must also change permissions for the role to stop the use of compromised credentials for malicious account activity. Permissions assigned to temporary security credentials are evaluated each time they are used to make an AWS request. Once you remove all permissions from the credentials, AWS requests that use them fail.

It might take a few minutes for policy updates to take effect. [Revoke the role's temporary security credentials](#) to force all users assuming the role to reauthenticate and request new credentials.

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.

Important

For users in IAM Identity Center, see [Disable user access](#) in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*. You can also [Remove user access](#) to cloud applications or custom SAML 2.0 applications in the IAM Identity Center console.

Topics

- [Deny access to all sessions associated with a role \(p. 457\)](#)
- [Deny access to a specific session \(p. 458\)](#)
- [Deny a user session with condition context keys \(p. 458\)](#)
- [Deny a session user with resource-based policies \(p. 459\)](#)

Deny access to all sessions associated with a role

Use this approach when you are concerned about suspicious access by:

- Principals from another account using cross-account access
- External user identities with permissions to access AWS resources in your account
- Users who have been authenticated in a mobile or web application with a web identity provider

This procedure denies permissions to **all** users that have permissions to assume a role.

To change or remove the permissions assigned to the temporary security credentials obtained by calling AssumeRole, AssumeRoleWithSAML, or AssumeRoleWithWebIdentity, GetFederationToken, or GetSessionToken, you can edit or delete the permissions policy that defines the permissions for the role.

Important

If there's a resource-based policy that allows the principal access, you must also add an explicit deny for that resource. See [Deny a session user with resource-based policies \(p. 459\)](#) for details.

1. Sign in to the AWS Management Console and open the IAM console.
2. In the navigation pane, choose the name of the role to edit. You can use the search box to filter the list.
3. Select the relevant policy.
4. Choose the **Permissions** tab.
5. Choose the **JSON** tab and update the policy to deny all resources and actions.

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

6. On the **Review** page, review the policy **Summary** and then choose **Save changes** to save your work.

When you update the policy, the changes affect the permissions of all temporary security credentials associated with the role, including credentials that were issued before you changed the role's permissions policy. After you update the policy, you can [revoke the role's temporary security credentials](#) to immediately revoke all permissions to the role's issued credentials.

Deny access to a specific session

When you update the roles that are assumable from an IdP with a deny-all policy or delete the role entirely, all users that have access to the role are disrupted. You can deny access based on the `Principal` element without impacting the permissions of all other sessions associated with the role.

The `Principal` can be denied permissions using [condition context keys \(p. 458\)](#) or [resource-based policies \(p. 459\)](#).

Tip

You can find the ARNs of federated users using AWS CloudTrail logs. For more information, see [How to Easily Identify Your Federated Users by Using AWS CloudTrail](#).

Deny a user session with condition context keys

You can use condition context keys in situations where you want to deny access to specific temporary security credential sessions without affecting the permissions of the IAM user or role that created the credentials.

For more information about condition context keys, see [AWS global condition context keys \(p. 1338\)](#).

Note

If there's a resource-based policy that allows the principal access, you must also add an explicit deny statement on the resource-based policy after you complete these steps.

After you update the policy, you can [revoke the role's temporary security credentials](#) to immediately revoke all issued credentials.

aws:PrincipalArn

You can use condition context key [aws:PrincipalArn \(p. 1347\)](#) to deny access to a specific principal ARN. You do this by specifying the unique identifier (ID) of the IAM user, role, or federated user the temporary security credentials are associated with in the Condition element of a policy.

1. In the IAM console navigation pane, choose the name of the role to edit. You can use the search box to filter the list.
2. Select the relevant policy.
3. Choose the **Permissions** tab.
4. Choose the **JSON** tab and add a deny statement for the principal ARN as shown in the following example.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Deny",  
            "Action": "*",  
            "Resource": "*",  
            "Condition": {  
                "ArnEquals": {  
                    "aws:PrincipalArn": [  
                        "arn:aws:iam::222222222222:role/ROLENAMESPACE",  
                        "arn:aws:iam::222222222222:user/USERNAME",  
                        "arn:aws:sts::222222222222:federated-user/USERNAME"  
                    ]  
                }  
            }  
        }  
    ]  
}
```

5. On the **Review** page, review the policy **Summary** and then choose **Save changes** to save your work.

aws:userid

You can use condition context key [aws:userid \(p. 1366\)](#) to deny access to all or specific temporary security credential sessions associated with the IAM user or role. You do this by specifying the unique identifier (ID) of the IAM user, role, or federated user the temporary security credentials are associated with in the Condition element of a policy.

The following policy shows an example of how you can deny access to temporary security credential sessions using condition context key aws :userid.

- AIDAXUSER1 represents the unique identifier for an IAM user. Specifying the unique identifier of an IAM user as a value for context key aws :userid will deny all sessions associated with the IAM user.
- AROAXROLE1 represents the unique identifier for an IAM role. Specifying the unique identifier of an IAM role as a value for context key aws :userid will deny all sessions associated with the role.
- AROAXROLE2 represents the unique identifier for an assumed-role session. In the caller-specified-role-session-name portion of the assumed-role unique identifier you can specify a role session name or a wildcard character if the StringLike condition operator is used. If you specify the role session name, it will deny the named role session without affecting the permissions of the role that created the credentials. If you specify a wildcard for the role session name, it will deny all sessions associated with the role.
- account-id:<federated-user-caller-specified-name> represents the unique identifier for a federated user session. A federated user is created by an IAM user calling the GetFederationToken API. If you specify the unique identifier for a federated user, it will deny the named federated user session without affecting the permissions of the role that created the credentials.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Deny",  
            "Action": "*",  
            "Resource": "*",  
            "Condition": {  
                "StringLike": {  
                    "aws:userId": [  
                        "AIDAXUSER1",  
                        "AROAXROLE1",  
                        "AROAXROLE2:<caller-specified-role-session-name>",  
                        "account-id:<federated-user-caller-specified-name>"  
                    ]  
                }  
            }  
        }  
    ]  
}
```

For specific examples of principal key values, see [Principal key values \(p. 1303\)](#). For information about IAM unique identifiers, see [Unique identifiers \(p. 1218\)](#).

Deny a session user with resource-based policies

If the principal ARN is also included in any resource-based policies, you must also revoke access based on the specific user's principalId or sourceIdentity values in the Principal element of a resource-based policy. If you only update the permissions policy for the role, the user can still perform actions allowed in the resource-based policy.

1. Refer to [AWS services that work with IAM \(p. 1224\)](#) to see if the service supports resource-based policies.

2. Sign in to the AWS Management Console and open the console for the service. Each service has a different location in the console for attaching policies.
3. Edit the policy statement to specify the identifying information of the credential:
 - a. In Principal, enter the ARN of the credential to deny.
 - b. In Effect, enter "Deny."
 - c. In Action, enter the service namespace and the name of the action to deny. To deny all actions, use the wildcard (*) character. For example: "s3:*."
 - d. In Resource, enter the ARN of the target resource. For example: "arn:aws:s3:::EXAMPLE-BUCKET."

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Principal": [  
                "arn:aws:iam::222222222222:role/ROLENAMESPACE",  
                "arn:aws:iam::222222222222:user/USERNAME",  
                "arn:aws:sts::222222222222:federated-user/USERNAME"  
            ],  
            "Effect": "Deny",  
            "Action": "s3:*",  
            "Resource": "arn:aws:s3:::EXAMPLE-BUCKET"  
        }  
    ]  
}
```

4. Save your work.

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 \(p. 428\)](#).

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 \(p. 271\)](#). For more information about controlling permissions in temporary security credentials, see [Controlling permissions for temporary security credentials \(p. 441\)](#).

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 limit the effects of a failure within a workload to a limited number of components with a predictable scope of impact containment. Using regional AWS STS endpoints lets you align the scope of your components with the scope of your session tokens. For more information about this reliability pillar, see [Use fault isolation to protect your workload](#) in the *AWS Well-Architected Framework*.
- **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 AWS STS endpoints. If you choose to use the global endpoint, you must change the Region compatibility of AWS 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 AWS 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 a 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. Under **Security Token Service (STS)** section **Session Tokens from the STS endpoints**. The **Global endpoint** indicates Valid only in AWS Regions enabled by default. Choose **Change**.
4. In the **Change region compatibility** dialog box, select **All AWS Regions**. Then choose **Save changes**.

Note

Tokens that are valid in all AWS Region 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 session 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 session 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. For Regions that are enabled by default, you must activate the Regional STS endpoint 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 Regions that are manually enabled, you must activate the Region in both the account making the request and the account where the temporary credentials are generated.

For example, imagine a user in account A wants to send an `sts:AssumeRole` API request to the AWS STS Regional endpoint `https://sts.us-east-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-east-2` Region. Users from account A (or any other account) can call the `us-east-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` \(p. 1353\)](#) 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 a user with permissions to perform IAM administration tasks.
2. Open the [IAM console](#) and in the navigation pane choose [Account settings](#).
3. In the **Security Token Service (STS)** section **Endpoints**, find the Region that you want to configure, and then choose **Active** or **Inactive** in the **STS status** column.
4. In the dialog box that opens, choose **Activate** or **Deactivate**.

For Regions that must be enabled, we activate AWS 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 AWS Region as the parameters.

To learn how to set AWS STS regional endpoints using an environment variable for AWS SDKs, see [AWS STS Regionalized endpoints](#) in the [AWS SDKs and Tools Reference Guide](#).

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
Asia Pacific (Hong Kong)	sts.ap-east-1.amazonaws.com	✗ No ¹	✗ No
Asia Pacific (Hyderabad)	sts.ap-south-2.amazonaws.com	✗ No ¹	✗ No
Asia Pacific (Jakarta)	sts.ap-southeast-3.amazonaws.com	✗ No ¹	✗ No
Asia Pacific (Melbourne)	sts.ap-southeast-4.amazonaws.com	✗ No ¹	✗ No
Asia Pacific (Mumbai)	sts.ap-south-1.amazonaws.com	✓ Yes	✓ Yes
Asia Pacific (Osaka)	sts.ap-northeast-3.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
China (Beijing)	sts.cn-north-1.amazonaws.com.cn	✓ Yes ²	✗ No
China (Ningxia)	sts.cn-northwest-1.amazonaws.com.cn	✓ Yes ²	✓ Yes
Europe (Frankfurt)	sts.eu-central-1.amazonaws.com	✓ Yes	✓ Yes

Region name	Endpoint	Active by default	Manually activate/deactivate
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 (Spain)	sts.eu-south-2.amazonaws.com	✗ No ¹	✗ No
Europe (Stockholm)	sts.eu-north-1.amazonaws.com	✓ Yes	✓ Yes
Europe (Zurich)	sts.eu-central-2.amazonaws.com	✗ No ¹	✗ No
Israel (Tel Aviv)	sts.il-central-1.amazonaws.com	✗ No ¹	✗ No
Middle East (Bahrain)	sts.me-south-1.amazonaws.com	✗ No ¹	✗ No
Middle East (UAE)	sts.me-central-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 AWS STS. You cannot manually activate or deactivate AWS STS in these Regions.

²To use AWS in China, you need an account and credentials specific to AWS in China.

AWS CloudTrail and Regional endpoints

Calls to regional and global endpoints are logged in the `tlsDetails` field in AWS CloudTrail. Calls to regional endpoints, such as `us-east-2.amazonaws.com`, are logged in CloudTrail to their appropriate region. Calls to the global endpoint, `sts.amazonaws.com`, are logged as calls to a global service. Events for global AWS STS endpoints are logged to `us-east-1`.

Note

`tlsDetails` can only be viewed for services that support this field. See [Services that support TLS details in CloudTrail](#) in the *AWS CloudTrail User Guide*. For more information, see [Logging IAM and AWS STS API calls with AWS CloudTrail \(p. 471\)](#).

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)
- Africa (Cape Town)
- Asia Pacific (Hong Kong)
- Asia Pacific (Mumbai)
- Asia Pacific (Osaka)
- Asia Pacific (Seoul)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- Canada (Central)
- China (Beijing)
- China (Ningxia)
- Europe (Frankfurt)
- Europe (Ireland)
- Europe (London)
- Europe (Milan)
- Europe (Paris)
- Europe (Stockholm)
- Middle East (Bahrain)
- South America (São Paulo)
- AWS GovCloud (US-East)
- AWS GovCloud (US-West)

Create a VPC endpoint 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 \(p. 461\)](#).

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": "*"  
        }  
    ]  
}
```

For a service bearer token example, see [Using identity-based policies for AWS CodeArtifact](#) in the *AWS CodeArtifact* user guide.

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 \(p. 426\)](#). 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:

- [Enabling Federation to AWS Using Windows Active Directory, ADFS, and SAML 2.0](#). Demonstrates how to delegate access using enterprise federation to AWS using Windows Active Directory (AD), Active Directory Federation Services (ADFS) 2.0, and SAML (Security Assertion Markup Language) 2.0.
- [Enabling custom identity broker access to the AWS console \(p. 233\)](#). Demonstrates how to create a custom federation proxy that enables single sign-on (SSO) so that existing Active Directory users can sign in to the AWS Management Console.
- [How to Use Shibboleth for Single Sign-On to the AWS Management Console..](#) Shows how to use [Shibboleth](#) and [SAML \(p. 205\)](#) 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 [Authentication with Amplify](#) in the [Amplify Documentation](#).
- [Web Identity Federation Playground](#). This website provides an interactive demonstration of [web identity federation \(p. 199\)](#) and the AssumeRoleWithWebIdentity API.

Additional resources for temporary security credentials

The following scenarios and applications can guide you in using temporary security credentials:

- [How to integrate AWS STS SourceIdentity with your identity provider](#). This post shows you how to set up the AWS STS SourceIdentity attribute when using Okta, Ping, or OneLogin as your IdP.
- [About web identity federation \(p. 199\)](#). This section discusses how to configure IAM roles when you use web identity federation and the AssumeRoleWithWebIdentity API.
- [Configuring MFA-protected API access \(p. 145\)](#). 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 \(p. 484\)](#)
- [Policy evaluation logic \(p. 1306\)](#).
- [Managing Access Permissions to Your Amazon S3 Resources](#) in [Amazon Simple Storage Service User 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* 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. As a [best practice \(p. 1032\)](#), safeguard your root user credentials and don't use them for everyday tasks. Root user credentials are only used to perform a few account and service management tasks.

To view the tasks that require you to sign in as the root user, see [Tasks that require root user credentials](#).

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. Inactive keys still 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.

Tasks

- [Create or delete an AWS account \(p. 469\)](#)
- [Enable MFA on the AWS account root user \(p. 469\)](#)
- [Creating access keys for the root user \(p. 470\)](#)
- [Deleting access keys for the root user \(p. 470\)](#)
- [Changing the password for the root user \(p. 471\)](#)
- [Securing the credentials for the root user \(p. 471\)](#)
- [Transferring the root user owner \(p. 471\)](#)

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

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. We recommend that you enable multiple MFA devices to your AWS account root user and IAM users in your AWS accounts. This allows you to raise the security bar in your AWS accounts, including your AWS account root user. You can register up to eight MFA devices of any combination of the currently supported MFA types for your AWS account root user and IAM users.

With multiple MFA devices, only one MFA device is needed to sign in to the AWS Management Console or create a session using the AWS CLI as that user. For more information, see [How do I use an MFA token to authenticate access to my AWS resources through the AWS CLI?](#)

For more information about enabling MFA, see the following:

- [Enable a virtual MFA device for your AWS account root user \(console\) \(p. 118\)](#)
- [Enable a hardware TOTP token for the AWS account root user \(console\) \(p. 132\)](#)

Creating access keys for the root user

Although we don't recommend it, 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

As the root user, you can't sign in to the **Sign in as IAM user** page. If you see the **Sign in as IAM user** page, choose **Sign in using root user email** near the bottom of the page. For help signing in as the root user, see [Signing in to the AWS Management Console as the root user](#) in the *AWS Sign-In User Guide*.

2. Choose your account name in the navigation bar, and then choose **Security credentials**.
3. If you see a warning about accessing the security credentials for your AWS account, choose **Continue to Security credentials**.
4. Review the alternatives. We don't recommend that you create root user access keys. If you choose to continue to create an access key, select the check box to indicate that you understand that this is not a best practice, and then choose **Create access key**.
5. On the **Retrieve access key** page, choose either **Show** or **Download .csv file**. This is your only opportunity to save your secret access key. After you've saved your secret access key in a secure location, chose **Done**.

If you choose **Download .csv 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 \(p. 1035\)](#), or at least deactivate it by choosing **Actions** and then **Deactivate** 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.

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 **Security credentials**.
3. If you see a warning about accessing the security credentials for your AWS account, choose **Continue to Security credentials**.
4. On the next screen, choose **Deactivate**. This deactivates the access key. We recommend that you verify that the access key is no longer in use before you permanently delete it. To confirm deletion, copy the access key ID, paste the access key ID in the text input field, and then choose **Delete**.

Changing the password for the root user

You can change the email address and password from either the [Security Credentials](#) or the [Account](#) page. You can also choose **Forgot password?** on the AWS sign-in page to reset your password.

For more information, see [Change the password for the AWS account root user](#) in the *AWS Account Management Reference Guide*.

To change the root user, you must log in using the root user credentials.

Securing the credentials for the root user

For more information about securing the credentials for the AWS account root user, see [Safeguard your root user credentials and don't use them for everyday tasks \(p. 1035\)](#).

Transferring the root user owner

To transfer ownership of the root user, see [How do I transfer my AWS account to another person or business?](#)

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 \(p. 472\)](#)
- [Logging IAM and AWS STS API requests \(p. 472\)](#)
- [Logging API requests to other AWS services \(p. 472\)](#)

- [Logging user sign-in events \(p. 473\)](#)
- [Logging sign-in events for temporary credentials \(p. 473\)](#)
- [Example IAM API events in CloudTrail log \(p. 474\)](#)
- [Example AWS STS API events in CloudTrail log \(p. 475\)](#)
- [Example sign-in events in CloudTrail log \(p. 481\)](#)

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 non-authenticated 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.

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 AWS 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 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.

To view sample CloudTrail events for successful and unsuccessful root user sign-ins, see [Example event records for root users in the AWS CloudTrail User Guide](#).

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 \(p. 482\)](#), 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 the sign-in page of one AWS account, but then type the account number for a different AWS account.
- 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 `sts:SourceIdentity` condition key in the role trust policy to require users to specify an identity when they assume a role. For example, you can require that IAM users specify their own user name as their source identity. This can help you determine which user performed a specific action in AWS. For more information, see [sts:SourceIdentity \(p. 1382\)](#). You can also use [sts:RoleSessionName \(p. 1381\)](#) to require users to specify a session name when they assume a role. This can help you differentiate between role sessions for a role that is used by different principals when you review AWS CloudTrail logs.

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

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	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
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	AssumeRoleWithSAMLM/a		SAML user identity	Role identity only (no user)
Externally authenticated user	AssumeRoleWithWebIdentity		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`. The account administrator requires users to set a source identity equal to their user name when assuming the role. The user passes in the source identity value of `JohnDoe`.

```
{
  "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.meta1.x86_64 botocore/1.4.67",
  "requestParameters": {
    "roleArn": "arn:aws:iam::111122223333:role/EC2-dev",
    "roleSessionName": "JohnDoe-EC2-dev",
    "sourceIdentity": "JohnDoe",
    "serialNumber": "arn:aws:iam::777788889999:mfa"
  },
  "responseElements": {
    "credentials": {
      "sessionToken": "<encoded session token blob>",
      "accessKeyId": "AKIAQRSTUVWXYZEXAMPLE",
      "expiration": "Jul 18, 2023, 4:07:39 PM"
    },
    "assumedRoleUser": {
      "type": "AWS"
    }
  }
}
```

```

        "assumedRoleId": "AIDAQRSTUVWXYZEXAMPLE:JohnDoe-EC2-dev",
        "arn": "arn:aws:sts::111122223333:assumed-role/EC2-dev/JohnDoe-EC2-dev"
    },
    "sourceIdentity": "JohnDoe"
},
"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.metall.x86_64
botocore/1.4.67",
    "requestParameters": {
        "roleArn": "arn:aws:iam::111122223333:role/EC2-dev",
        "roleSessionName": "JohnDoe-EC2-dev",
        "sourceIdentity": "JohnDoe",
        "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"
        },
        "sourceIdentity": "JohnDoe"
    },
    "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 JohnRole2 role. This is known as [role chaining \(p. 185\)](#). The source identity that he set when he assumed the JohnDoe1 role persists in the request to assume JohnRole2. If John tries to set a different source identity when assuming the role, the request is denied. John passes two [session tags \(p. 417\)](#) 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 source identity, see [Monitor and control actions taken with assumed roles \(p. 444\)](#). For more information about transitive keys in role chains, see [Chaining roles with session tags \(p. 424\)](#).

```
{
    "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",
                "creationDate": "2019-10-02T21:50:54Z"
            },
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIN5ATK5U7KEXAMPLE",
                "arn": "arn:aws:iam::111111111111:role/JohnRole1",
                "accountId": "111111111111",
                "userName": "JohnDoe"
            },
            "sourceIdentity": "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",
        "sourceIdentity": "JohnDoe",
        "transitiveTagKeys": [
            "Email",
            "CostCenter"
        ],
        "durationSeconds": 3600
    },
    "responseElements": {
        "credentials": {

```

```

    "accessKeyId": "ASIAWHOJDLGPOEXAMPLE",
    "expiration": "Oct 2, 2019, 11:12:29 PM",
    "sessionToken": "AgoJb3JpZ2luX2VjEB4aCXVzLXd1c3QtMSJHMEXAMPLETOKEN
+//rJb8Lo30mFc5M1hFCEbubZvEj0wHB/mDMwIgSEe9gk/Zjr09tZV7F1HDTMhmEXAMPLETOKEN/iEJ/
rkqngII9///////////
ARABGgw0MjgzMDc4NjM5NjYiDLZjZFKwP4qxQG5sFCryAS04UPz5qE97wPPH1eLMvs7CgSDBSWfonmRTcfokm2FN1+hWUdQQH6adjbb
+C+WFZb701eiv9J5La2EXAMPLETOKEN/c7S5Iro1WUJ0q3Cxuo/8HuoSxVhQHM7zF7mWLhXLEQ52ivL
+F6qdpXu4aTFedpMfnJa8JtkWwG9x1Axj0Ypy2ok8v5unpQGwych1vwdvj6ez1Dm8Xg1+qIZXILiEXAMPLETOKEN/
vQGqu8H+nxp3kabcr0vTFTvxX6vsc80GwUfHzAfYGEEXAMPLETOKEN/
Lv1yMM3B10wF0rQBno1HEjf1oNI8RnQiMNFdU0twYj7HUZIOCZmjfn8PPHq77N7GJ191zvIZKQA00wcjg
+mc78zHCj8y0siY8C96paEXAMPLETOKEN/
E3cpksxWdgs91HRzJWScjN2+r2LTGjYhyPqcmFzzo2mCE7mBNEXAMPLETOKEN/oJy
+2o83YNW5t0iDmczgDzJZ4UKR84yGYOMfSnF4XcEJrDgAJ30JFwmTcTQICAlSwLEXAMPLETOKEN"
},
"assumedRoleUser": {
    "assumedRoleId": "AROAIFR7WHDTSOYQYHFUE:Role2WithTags",
    "arn": "arn:aws:sts::111111111111:assumed-role/test-role/Role2WithTags"
},
"sourceIdentity": "JohnDoe"
},
"requestID": "b96b0e4e-e561-11e9-8b3f-7b396EXAMPLE",
"eventID": "1917948f-3042-46ec-98e2-62865EXAMPLE",
"resources": [
{
    "ARN": "arn:aws:iam::111111111111: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.meta1.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 \(p. 417\)](#). Those tags are set as transitive so that they [persist in role chaining scenarios \(p. 424\)](#). The request includes the SAML attribute sourceIdentity, which is passed in the SAML assertion. If someone uses the resulting role session credentials to assume another role, this source identity persists.

```

{
    "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.meta1.x86_64 botocore/1.12.253",
    "requestParameters": {
        "SAMLAssertionID": "_c0046cEXAMPLEb9d4b8eEXAMPLE2619aEXAMPLE",
        "roleSessionName": "MyAssignedRoleSessionName",
        "sourceIdentity": "MySAMLUser",
        "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",
        "sourceIdentity": "MySAMLUser"
        "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 \(p. 417\)](#). Those tags are set as transitive so that they [persist in role chaining \(p. 424\)](#). The request includes the sourceIdentity attribute from the identity provider token. If someone uses the resulting role session credentials to assume another role, this source identity persists.

```
{
    "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": {
        "sourceIdentity": "MyWebIdentityUser",
        "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>",
    "sourceIdentity": "MyWebIdentityUser",
    "audience": "<id of application>.apps.googleusercontent.com",
    "credentials": {
        "accessKeyId": "ASIAACQRSTUVWRAOEXAMPLE",
        "expiration": "Mar 23, 2016, 2:39:51 AM",
        "sessionToken": "<encoded session token blob>"
    },
    "assumedRoleUser": {
        "assumedRoleId": "AROACQRSTUVWRAOEXAMPLE: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_REASON` to help prevent exposing potentially sensitive information.

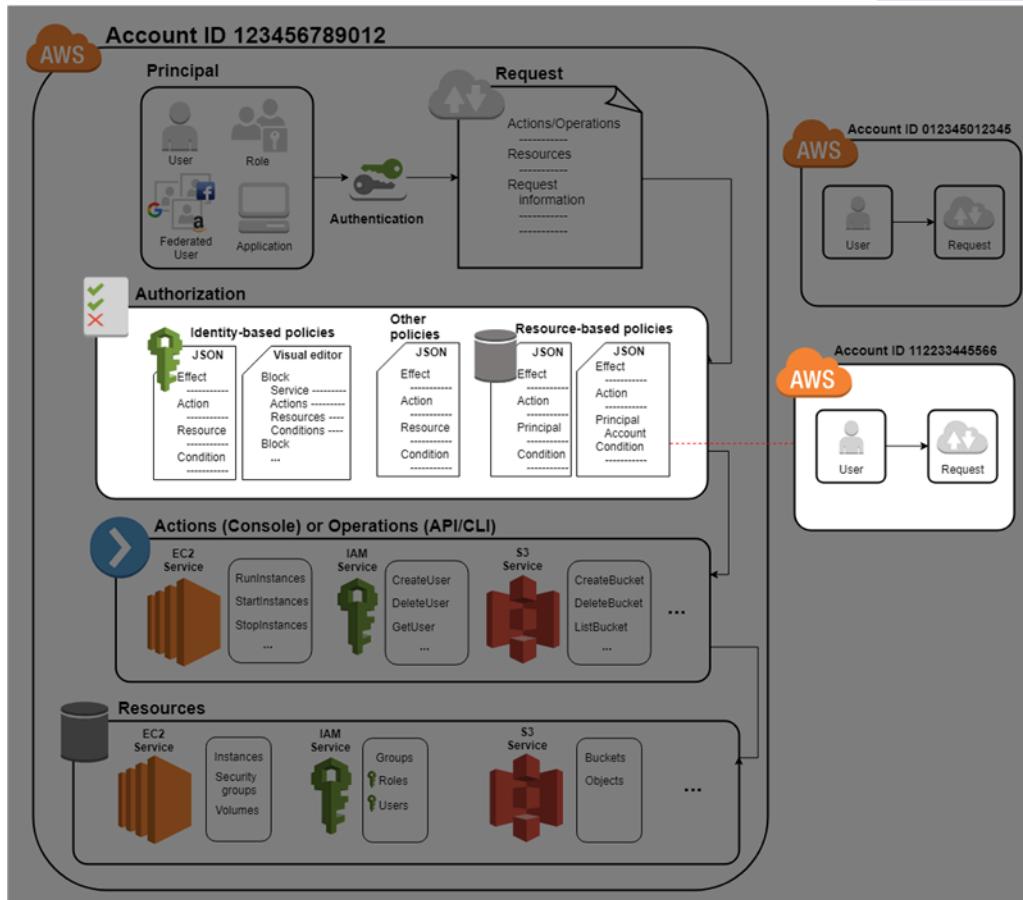
```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "accountId": "123456789012",
    "accessKeyId": "",
    "userName": "HIDDEN_DUE_TO_SECURITY_REASON"
  },
}
```

```
"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 \(p. 7\)](#) 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 \(p. 485\)](#).

For details about the rest of the authentication and authorization process, see [How IAM works \(p. 6\)](#).



During authorization, the AWS enforcement code uses values from the [request context \(p. 8\)](#) 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 \(p. 1306\)](#) 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 their sign-in credentials. 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 from most frequently used to less frequently used, are available for use in AWS. For more details, see the sections below for each policy type.

- [Identity-based policies \(p. 486\)](#) – Attach [managed \(p. 486\)](#) and [inline \(p. 486\)](#) policies to IAM identities (users, groups to which users belong, or roles). Identity-based policies grant permissions to an identity.

- **Resource-based policies (p. 486)** – 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. 487)** – 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. 487)** – 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. 487)** – 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. 487)** – 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 \(p. 498\)](#).

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. For step-by-step instructions for granting cross-service access, see [IAM tutorial: Delegate access across AWS accounts using IAM roles \(p. 40\)](#).

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 [Cross account resource access in IAM \(p. 526\)](#).

To see which other services support resource-based policies, see [AWS services that work with IAM \(p. 1224\)](#). To learn more about resource-based policies, see [Identity-based policies and resource-based policies \(p. 511\)](#). 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 \(p. 501\)](#).

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

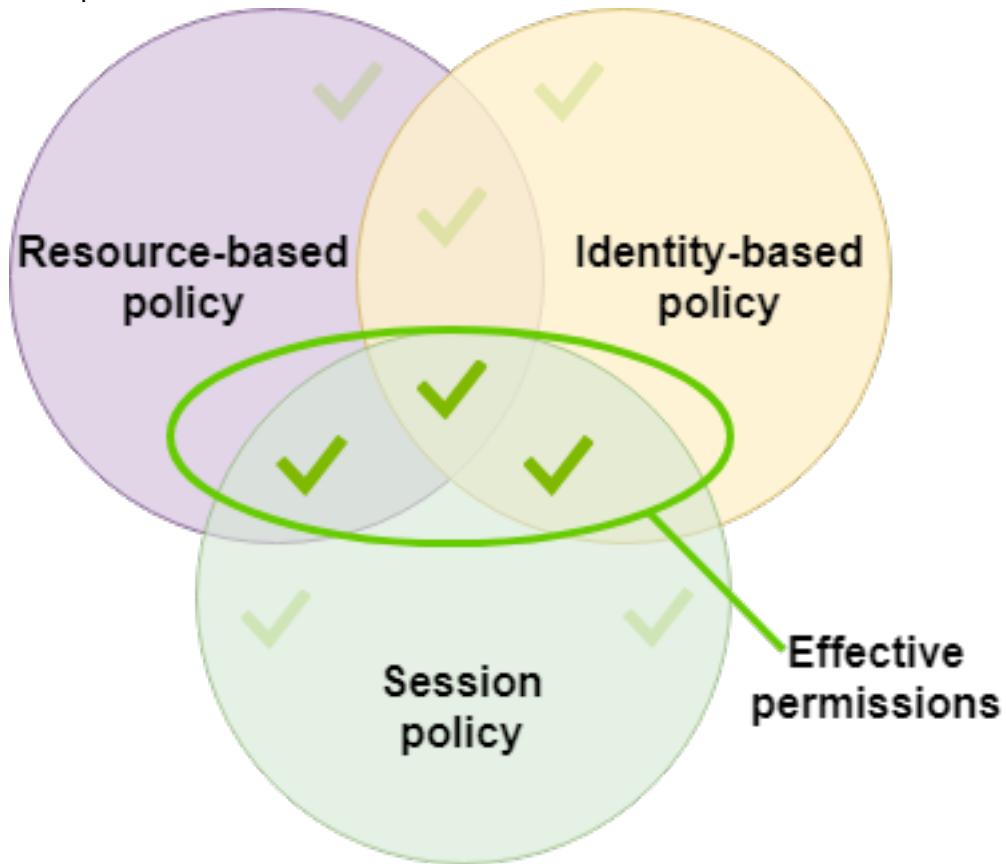
Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The permissions for a session are the intersection of the identity-based policies for the IAM entity (user or role) used to create the session and the 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

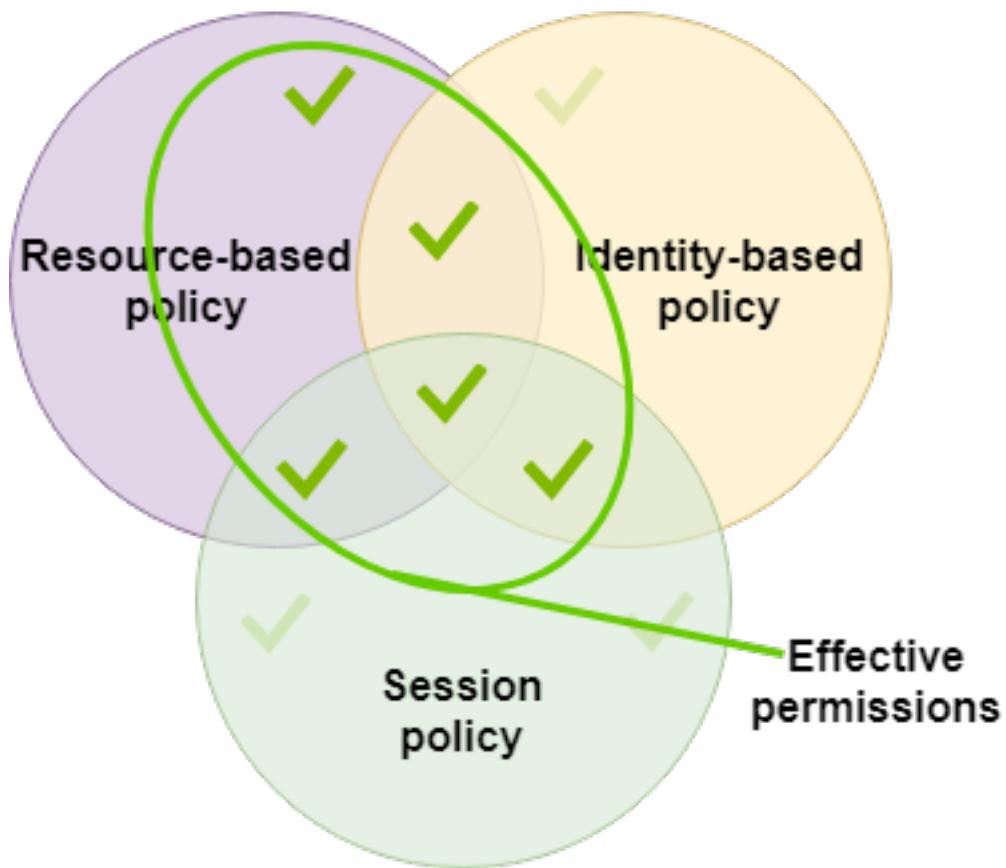
to specify up to 10 managed session policies. For more information about creating a role session, see [Requesting temporary security credentials \(p. 428\)](#).

When you create a federated user session, you use the access keys of the IAM user to programmatically call the `GetFederationToken` API operation. You must also pass session policies. The resulting session's permissions are the intersection of the identity-based policy and the session policy. For more information about creating a federated user session, see [GetFederationToken—federation through a custom identity broker \(p. 433\)](#).

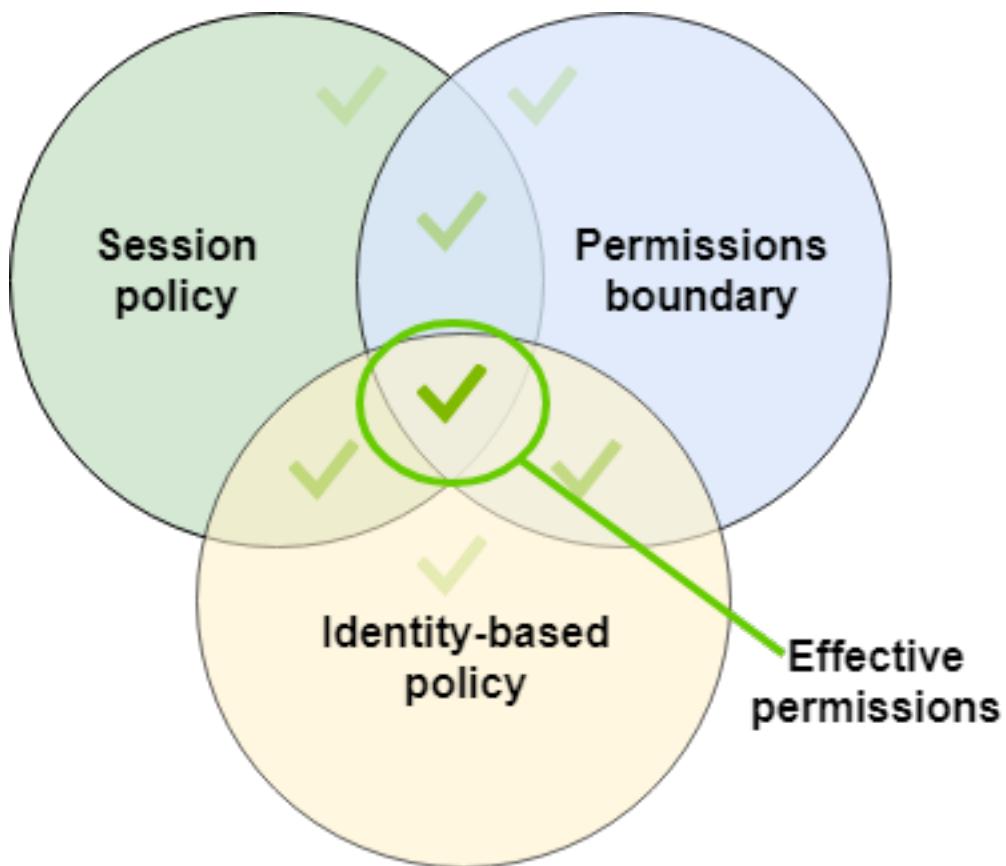
A resource-based policy can specify the ARN of the user or role as a principal. In that case, the permissions from the resource-based policy are added to the role or user's identity-based policy before the session is created. The session policy limits the total permissions granted by the resource-based policy and the identity-based policy. The resulting session's permissions are the intersection of the session policies and the resource-based policies plus the intersection of the session policies and identity-based policies.



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. A root user is still the member of an account. If that account is a member of an organization in AWS Organizations, 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 \(p. 581\)](#) and [Editing IAM policies \(p. 609\)](#).

When you create or edit a JSON policy, IAM can perform policy validation to help you create an effective policy. IAM identifies JSON syntax errors, while IAM Access Analyzer provides additional policy checks with recommendations to help you further refine your policies. To learn more about policy validation, see

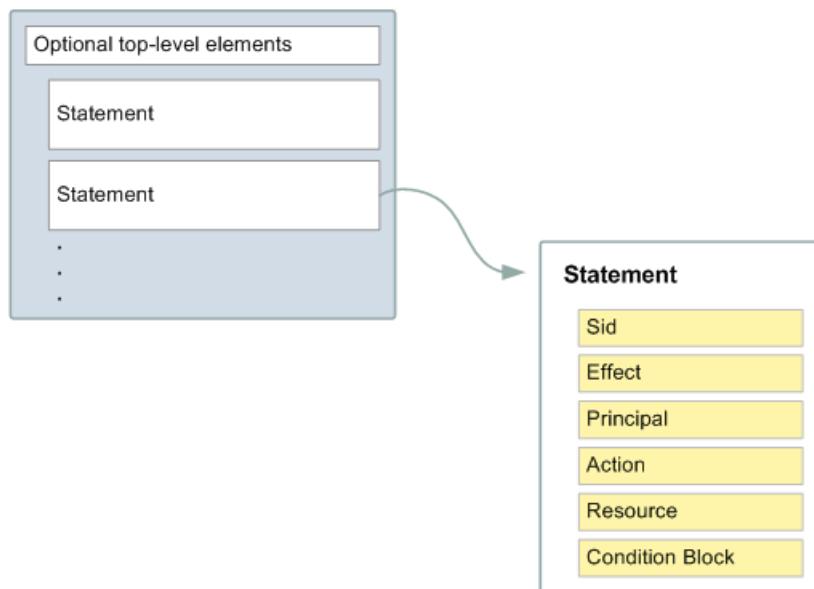
[Validating IAM policies \(p. 588\)](#). To learn more about IAM Access Analyzer policy checks and actionable recommendations, see [IAM Access Analyzer policy validation](#).

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. We recommend that you use the latest 2012-10-17 version. For more information, see [IAM JSON policy elements: Version \(p. 1261\)](#)
- **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 \(p. 1260\)](#).

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. We recommend that you break up policies by resource type.

Because of the [limited size of policies \(p. 1220\)](#), 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 \(p. 1306\)](#) 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*"  
      ],  
      "Condition": {"StringEquals": {"aws:MultiFactorAuthPresent": "true"}  
    }  
  ]  
}
```

```
    "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",
    "Statement": [
        {
            "Sid": "1",
            "Effect": "Allow",
            "Principal": {"AWS": ["arn:aws:iam::account-id: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 \(p. 529\)](#).

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.

As an alternative to least privilege, you can use [AWS managed policies \(p. 495\)](#) or policies with wildcard * permissions to get started with policies. Consider the security risk of granting your principals more permissions than they need to do their job. Monitor those principals to learn which permissions they are using. Then write least privilege policies.

IAM provides several options to help you refine the permissions that you grant.

- **Understand access level groupings** – You can use access level groupings to understand the level of access that a policy grants. [Policy actions \(p. 1273\)](#) 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 [Understanding access level summaries within policy summaries \(p. 643\)](#).
- **Validate your policies** – You can perform policy validation using IAM Access Analyzer when you create and edit JSON policies. We recommend that you review and validate all of your existing policies. IAM Access Analyzer provides over 100 policy checks to validate your policies. It generates security warnings when a statement in your policy allows access we consider overly permissive. You can use the actionable recommendations that are provided through the security warnings as you work toward granting least privilege. To learn more about policy checks provided by IAM Access Analyzer, see [IAM Access Analyzer policy validation](#).
- **Generate a policy based on access activity** – To help you refine the permissions that you grant, you can generate an IAM policy that is based on the access activity for an IAM entity (user or role). IAM Access Analyzer reviews your AWS CloudTrail logs and generates a policy template that contains the permissions that have been used by the entity in your specified time frame. You can use the template to create a managed policy with fine-grained permissions and then attach it to the IAM entity. That way, you grant only the permissions that the user or role needs to interact with AWS resources for your specific use case. To learn more, see [Generate policies based on access activity \(p. 589\)](#).
- **Use last accessed information** – Another feature that can help with least privilege 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 that were last accessed for some services, such as Amazon EC2, IAM, Lambda, and 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 \(p. 616\)](#).
- **Review account events in AWS CloudTrail** – 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 policy topics for individual services, which provide examples of how to write policies for service-specific resources.

- [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 User Guide*
- [Access Control List \(ACL\) Overview](#) in the *Amazon Simple Storage Service User Guide*

Managed policies and inline policies

When you 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 \(p. 495\)](#)
- [Customer managed policies \(p. 496\)](#)

- [Inline policies \(p. 497\)](#)
- [Choosing between managed policies and inline policies \(p. 498\)](#)
- [Getting started with managed policies \(p. 500\)](#)
- [Converting an inline policy to a managed policy \(p. 500\)](#)
- [Deprecated AWS managed policies \(p. 501\)](#)

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 \(p. 1213\)](#).

AWS managed policies make it convenient for you to assign appropriate permissions to users, groups, and roles. It is faster than writing the policies yourself, and includes permissions for many common use cases.

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.

Full access AWS managed policies define permissions for service administrators by granting full access to a service.

- [AmazonDynamoDBFullAccess](#)
- [IAMFullAccess](#)

Power-user AWS managed policies provide full access to AWS services and resources, but do not allow managing users and groups.

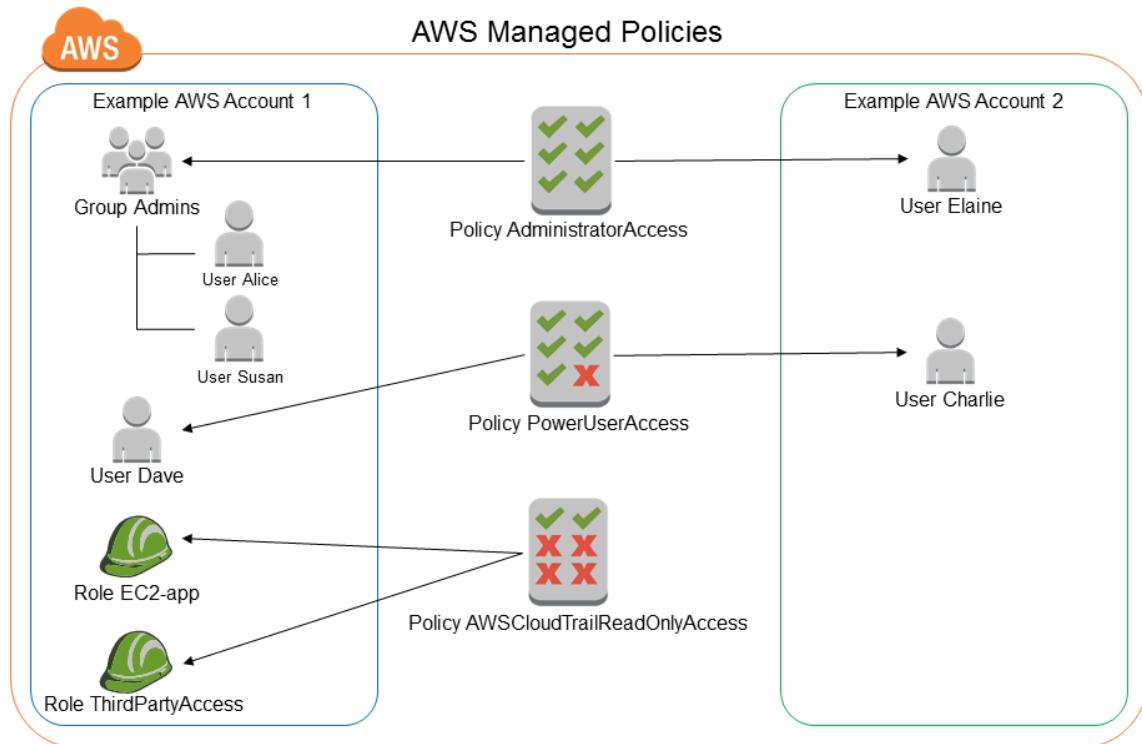
- [AWSCodeCommitPowerUser](#)
- [AWSKeyManagementServicePowerUser](#)

Partial-access AWS managed policies provide specific levels of access to AWS services without allowing [permissions management \(p. 645\)](#) access level permissions.

- [AmazonMobileAnalyticsWriteOnlyAccess](#)
- [AmazonEC2ReadOnlyAccess](#)

One particularly useful category of AWS managed policies are those designed for job functions. These policies align closely with commonly used job functions in the IT industry and facilitate granting permissions for these job functions. 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 you use this policy 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 \(p. 1329\)](#).

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

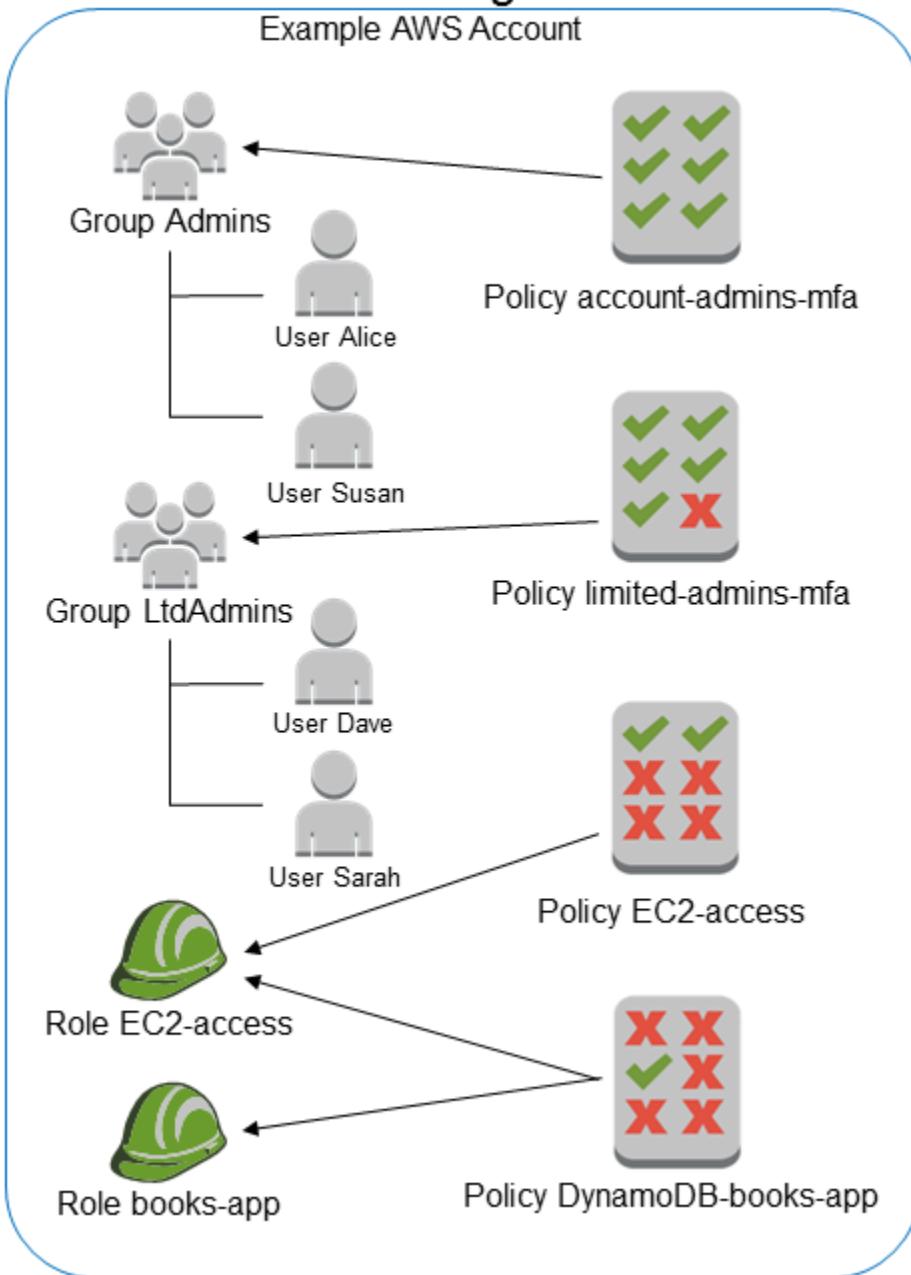
You can create standalone policies in your own AWS account that you can attach to principal entities (users, groups, and roles). You create these *customer managed policies* for your specific use cases, and you can change and update them as often as you like. Like AWS managed policies, when you attach a policy to a principal entity, you give the entity the permissions that are defined in the policy. When you update permissions in the policy, the changes are applied to all principal entities that the policy is attached to.

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. 1213) 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.

For more information, see [Creating IAM policies \(p. 581\)](#)

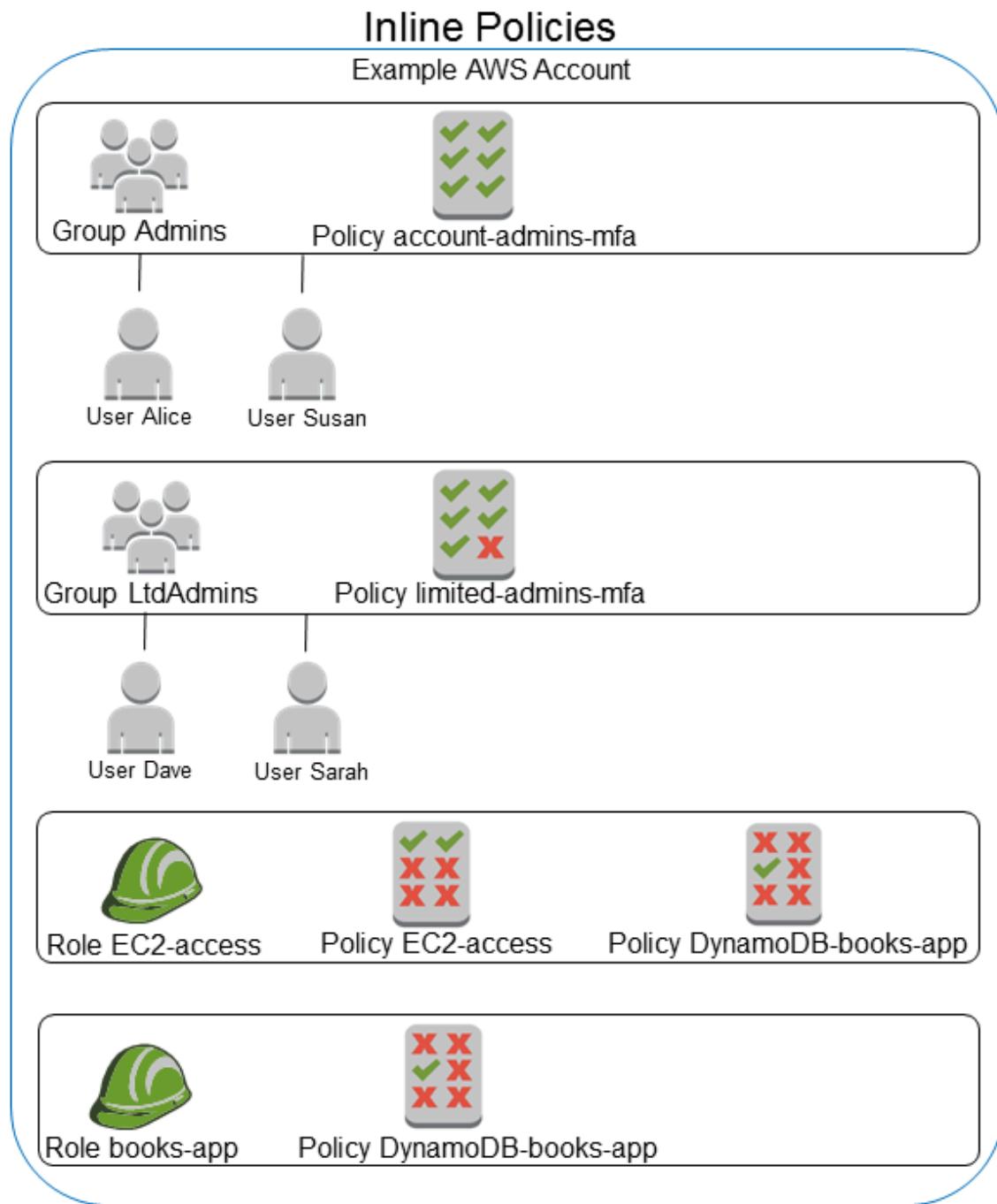
Customer Managed Policies



Inline policies

An inline policy is a policy created for a single IAM identity (a 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. You can create a policy and embed it in an identity, either when you create the identity or later. If a policy could apply to more than one entity, it's better to use a managed policy.

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

Consider your use cases when deciding between managed and inline policies. In most cases, we recommend that you use managed policies instead of inline policies.

Note

You can use both managed and inline policies together to define common and unique permissions for a principal entity.

Managed policies provide the following features:

Reusability

A single managed policy can be attached to multiple principal entities (users, groups, and roles). You can create a library of policies that define useful permissions 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 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 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 as needed.

Note

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" \(p. 606\)](#). To learn more about the `Version` policy element see [IAM JSON policy elements: Version \(p. 1261\)](#).

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. To do this, 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. Those limited administrators 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 \(p. 517\)](#).

Larger policy character limits

The maximum character size limit for managed policies is greater than the character limit for inline policies. If you reach the inline policy's character size limit, you can create more IAM groups and attach the managed policy to the group.

For more information on quotas and limits, see [IAM and AWS STS quotas \(p. 1220\)](#).

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 to which it is applied. For example, if 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 because they are part of the principal entity.

Getting started with managed policies

We recommend using policies that [grant least privilege \(p. 493\)](#), or granting only the permissions required to perform a task. The most secure way to grant least privilege is to write a customer managed policy with only the permissions needed by your team. You must create a process to allow your team to request more permissions when necessary. It takes time and expertise to [create IAM customer managed policies \(p. 582\)](#) that provide your team with only the permissions they need.

To get started adding permissions to your IAM identities (users, groups of users, and roles), you can use [AWS managed policies \(p. 495\)](#). AWS managed policies don't grant least privilege permissions. You must consider the security risk of granting your principals more permissions than they need to do their job.

You can attach AWS managed policies, including job functions, to any IAM identity. For more information, see [Adding and removing IAM identity permissions \(p. 598\)](#).

To switch to least privilege permissions, you can run AWS Identity and Access Management Access Analyzer to monitor the principals with AWS managed policies. After learning which permissions they are using, then you can write or generate a customer managed policy with only the required permissions for your team. This is less secure, but provides more flexibility as you learn how your team is using AWS. For more information, see [IAM Access Analyzer policy generation \(p. 1154\)](#).

AWS managed policies are designed to provide permissions for many common use cases. For more information about AWS managed policies that are designed for specific job functions, see [AWS managed policies for job functions \(p. 1329\)](#).

For a list of AWS managed policies, see [AWS Managed Policy Reference Guide](#).

Converting an inline policy to a managed policy

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.

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 **User groups, Users, or Roles**.
3. In the list, choose the name of the user group, user, or role that has the policy you want to remove.
4. Choose the **Permissions** tab.
5. For user groups, select the name of the inline policy that you want to remove. For users and roles, choose **Show n more**, if necessary, and then expand the inline policy that you want to remove.
6. Choose **Copy** to copy the JSON policy document for the policy.
7. In the navigation pane, choose **Policies**.
8. Choose **Create policy** and then choose the **JSON** option.
9. Replace the existing text with your JSON policy text, and then choose **Next**.
10. Enter a name and optional description for your policy and choose **Create policy**.

11. In the navigation pane, choose **User groups**, **Users**, or **Roles**, and again choose the name of the user group, user, or role that has the policy you want to remove.
 12. Choose the **Permissions** tab and then choose **Add permissions**.
 13. For user groups, select the check box next to the name of your new policy, choose **Add permissions**, 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**, and then choose **Add permissions**.
- You are returned to the **Summary** page for your user group, user, or role.
14. Select the check box next to the inline policy that you want to remove and choose **Remove**.

Deprecated AWS managed policies

To simplify the assignment of permissions, AWS provides [managed policies \(p. 494\)](#)—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 \(p. 485\)](#).

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": "*"  
        }  
    ]  
}
```

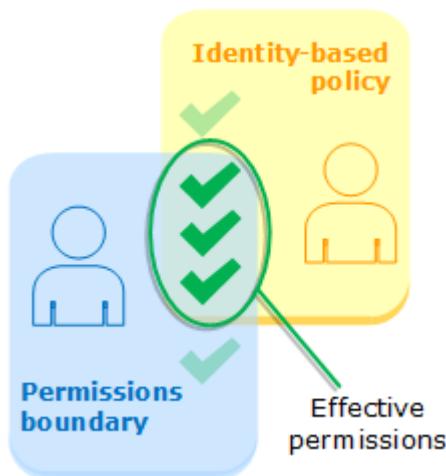
This policy allows creating a user in IAM. If you attach this permissions policy to the ShirleyRodriguez user, and Shirley tries to create a user, the operation fails. It fails because the permissions boundary does not allow the `iam:CreateUser` operation. Given these two policies, Shirley does not have 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 \(p. 485\)](#).

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 \(p. 1306\)](#).

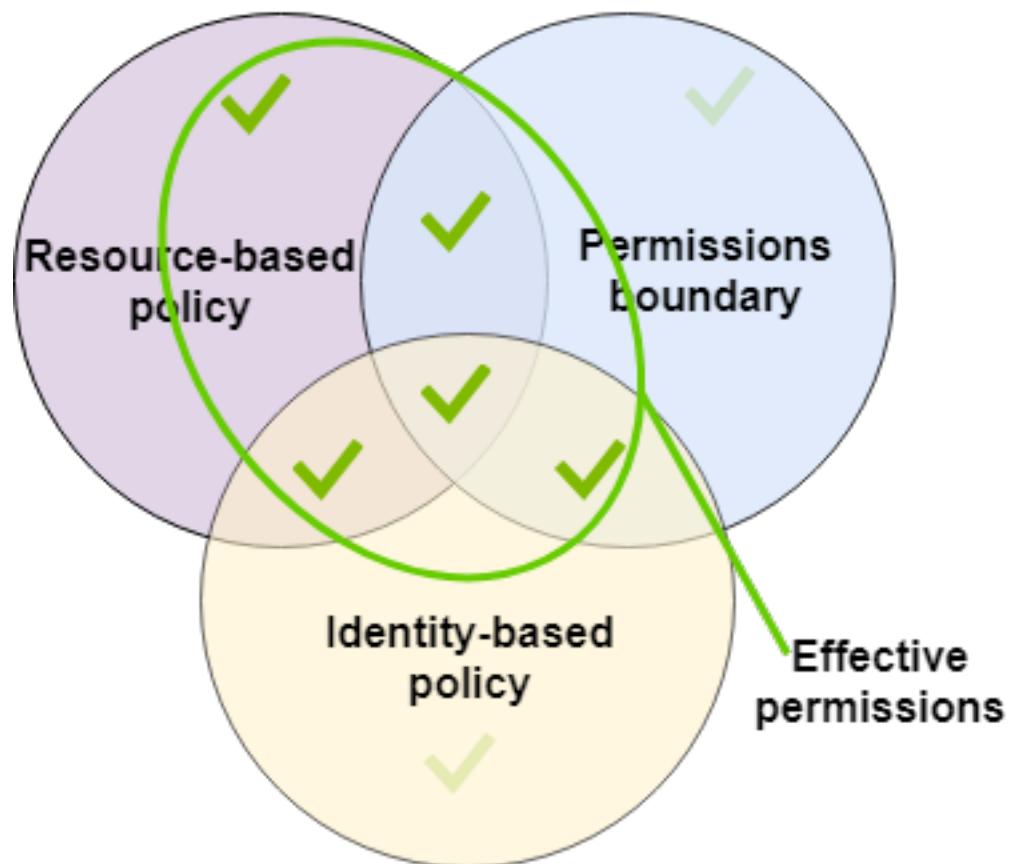
Identity-based policies with boundaries – Identity-based policies are inline or managed policies that are attached to a user, group of users, or role. Identity-based policies grant permission to the entity, and permissions boundaries limit those permissions. The effective permissions are the intersection 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 the same account, resource-based policies that grant permissions to an IAM user ARN (that is not a federated user session) are not limited by an implicit deny in an identity-based policy or permissions boundary.



Resource-based policies for IAM roles

IAM role – Resource-based policies that grant permissions to an IAM role ARN are limited by an implicit deny in a permissions boundary or session policy.

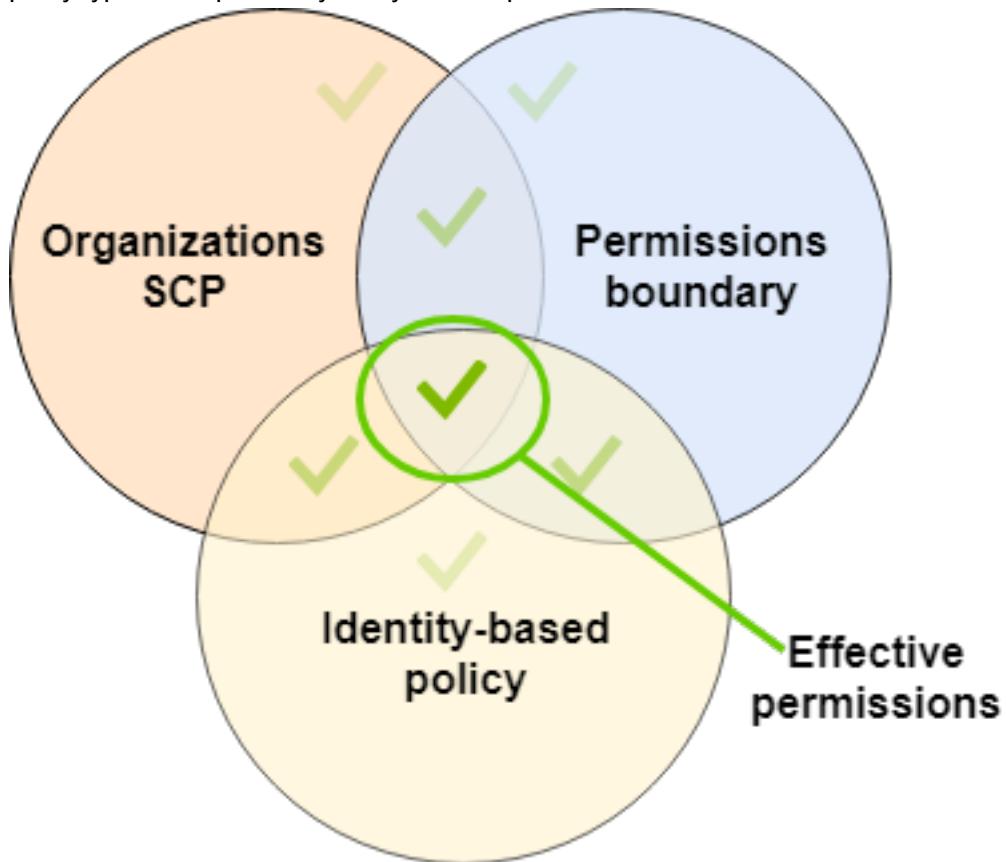
IAM role session – Within the same account, resource-based policies that grant permissions to an IAM role session ARN grant permissions directly to the assumed role session. Permissions granted directly to a session are not limited by an implicit deny in an identity-based policy, a permissions boundary, or session policy. When you assume a role and make a request, the principal making the request is the IAM role session ARN and not the ARN of the role itself.

Resource-based policies for IAM federated user sessions

IAM federated user sessions – An IAM federated user session is a session created by calling [GetFederationToken \(p. 433\)](#). When a federated user makes a request, the principal making the request is the federated user ARN and not the ARN of the IAM user who federated. Within the same account, resource-based policies that grant permissions to a federated user ARN grant permissions directly to the session. Permissions granted directly to a session are not limited by an implicit deny in an identity-based policy, a permissions boundary, or session policy.

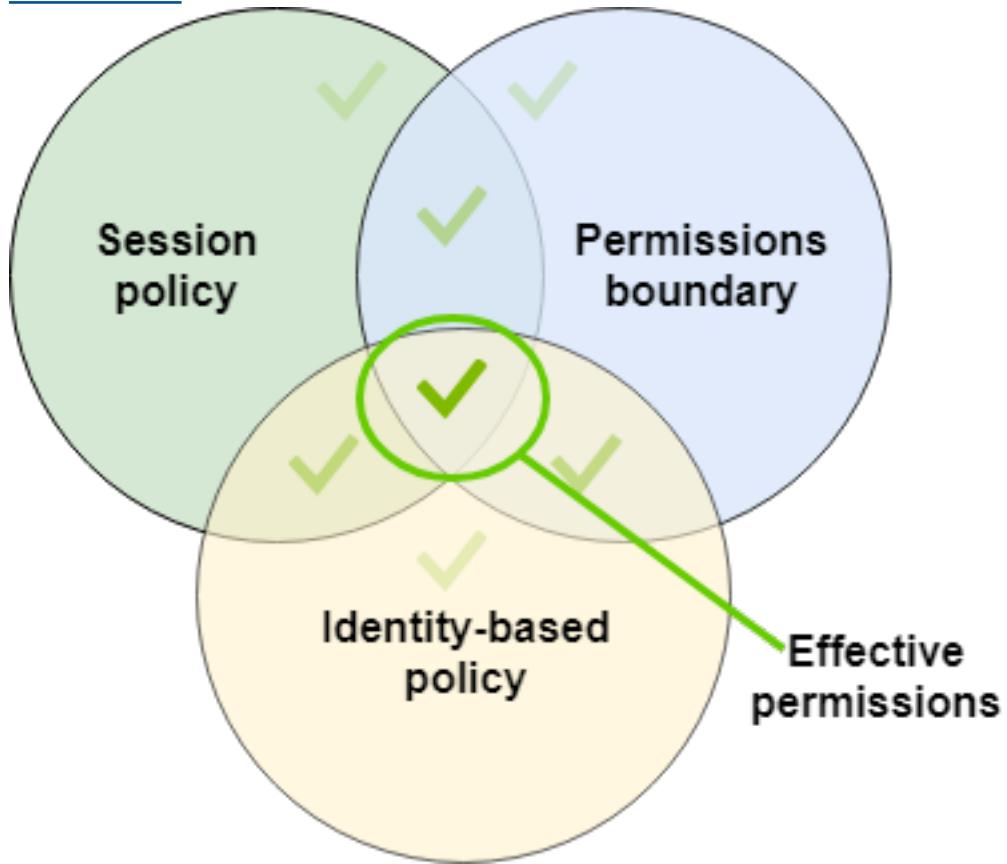
However, if a resource-based policy grants permission to the ARN of the IAM user who federated, then requests made by the federated user during the session are limited by an implicit deny in a permission boundary or session policy.

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 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>SetDefaultLoginProfile"
            ],
            "Resource": "*"
        }
    ]
}
```

```

        "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"
}
]
}

```

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:AttachUserPolicy",
                "iam>CreateUser",
                "iam>DeleteUserPolicy",
                "iam:DetachUserPolicy",
                "iam:PutUserPermissionsBoundary",
                "iam:PutUserPolicy"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {

```

```

        "iam:PermissionsBoundary": "arn:aws:iam::123456789012:policy/
XCompanyBoundaries"
    }
}
{
    "Sid": "CloudWatchAndOtherIAMTasks",
    "Effect": "Allow",
    "Action": [
        "cloudwatch:*",
        "iam:CreateAccessKey",
        "iam:CreateGroup",
        "iam:CreateLoginProfile",
        "iam:CreatePolicy",
        "iam:DeleteGroup",
        "iam:DeletePolicy",
        "iam:DeletePolicyVersion",
        "iam:DeleteUser",
        "iam:GetAccountPasswordPolicy",
        "iam:GetGroup",
        "iam:GetLoginProfile",
        "iam:GetPolicy",
        "iam:GetPolicyVersion",
        "iam:GetRolePolicy",
        "iam:GetUser",
        "iam:GetUserPolicy",
        "iam>ListAccessKeys",
        "iam>ListAttachedRolePolicies",
        "iam>ListAttachedUserPolicies",
        "iam>ListEntitiesForPolicy",
        "iam>ListGroups",
        "iam>ListGroupsForUser",
        "iam>ListMFADevices",
        "iam>ListPolicies",
        "iam>ListPolicyVersions",
        "iam>ListRolePolicies",
        "iam>ListSSHPublicKeys",
        "iam>ListServiceSpecificCredentials",
        "iam>ListSigningCertificates",
        "iam>ListUserPolicies",
        "iam>ListUsers",
        "iam:SetDefaultPolicyVersion",
        "iam:SimulateCustomPolicy",
        "iam:SimulatePrincipalPolicy",
        "iam:UpdateGroup",
        "iam:UpdateLoginProfile",
        "iam:UpdateUser"
    ],
    "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 `NotResource` policy element. 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.

Maria then assigns the `DelegatedUserBoundary` policy [as the permissions boundary \(p. 91\)](#) for the Zhang user.

Task 3: Because the permissions boundary limits the maximum permissions, but does not grant access on its own, Maria 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 \(p. 494\)](#).

Resource-based policies are attached to a resource. For example, you can attach resource-based policies to Amazon S3 buckets, Amazon SQS queues, VPC endpoints, and AWS Key Management Service encryption keys. For a list of services that support resource-based policies, see [AWS services that work with IAM \(p. 1224\)](#).

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 \(p. 1213\)](#) 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 \(p. 1224\)](#).

To learn how identity-based policies and resource-based policies interact within the same account, see [Evaluating policies within a single account \(p. 1307\)](#).

To learn how the policies interact across accounts, see [Cross-account policy evaluation logic \(p. 1317\)](#).

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 \(p. 1306\)](#).
- **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 [Cross account resource access in IAM \(p. 526\)](#).

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 \(p. 1306\)](#).

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 User 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 \(p. 485\)](#) 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 \(p. 490\)](#) and specify the permissions for principal entities. For more information about policy types and uses, see [Policies and permissions in IAM \(p. 485\)](#).

AWS authorizes the request only if each part of your request is allowed by the policies. To view a diagram of this process, see [How IAM works \(p. 6\)](#). For details about how AWS determines whether a request is allowed, see [Policy evaluation logic \(p. 1306\)](#).

When you create an IAM policy, you can control access to the following:

- [Principals \(p. 514\)](#) – Control what the person making the request (the [principal \(p. 7\)](#)) is allowed to do.
- [IAM Identities \(p. 515\)](#) – Control which IAM identities (user groups, users, and roles) can be accessed and how.
- [IAM Policies \(p. 517\)](#) – Control who can create, edit, and delete customer managed policies, and who can attach and detach all managed policies.
- [AWS Resources \(p. 520\)](#) – Control who has access to resources using an identity-based policy or a resource-based policy.
- [AWS Accounts \(p. 521\)](#) – 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 user 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 user group, all users in that user group get those permissions. For example, you can give the Administrators user group permission to perform any of the IAM actions on any of the AWS account resources. Another example: You can give the Managers user group permission to describe the Amazon EC2 instances of the AWS account.

For information about how to delegate basic permissions to your users, user groups, and roles, see [Permissions required to access IAM resources \(p. 662\)](#). For additional examples of policies that illustrate basic permissions, see [Example policies for administering IAM resources \(p. 665\)](#).

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, user group, or role). You can also use a [permissions boundary \(p. 501\)](#) 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 user 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 **DOC-EXAMPLE-BUCKET1** 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 **DOC-EXAMPLE-BUCKET1** 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": "PermissionsBoundarySomeServices",  
            "Effect": "Allow",  
            "Action": [  
                "cloudwatch:*",  
                "dynamodb:*",  
                "ec2:*",  
                "s3:*"  
            ],  
            "Resource": "*"  
        },  
        {  
            "Sid": "PermissionsBoundaryNoConfidentialBucket",  
            "Effect": "Deny",  
            "Action": "s3:*",  
            "Resource": [  
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET1/*"  
            ]  
        }  
    ]  
}
```

```
        "arn:aws:s3:::DOC-EXAMPLE-BUCKET1",
        "arn:aws:s3:::DOC-EXAMPLE-BUCKET1/*"
    ]
}
}
```

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 \(p. 501\)](#).

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 \(p. 581\)](#).
- To learn how to attach an IAM policy to a principal, see [Adding and removing IAM identity permissions \(p. 598\)](#).
- 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 \(p. 553\)](#).
- 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 \(p. 580\)](#).
- 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 \(p. 533\)](#).

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 user 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 user group named **AllUsers**, and then attach that user group to all users. When you create the user 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 user 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 user group management actions for everyone in the user group. Finally, you attach this policy to the user 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 user 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 **Policy editor** section, choose the **Visual** option.
5. In **Select a service** choose **IAM**.
6. In **Actions allowed**, 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.

7. Choose **Resources** to specify resources for your policy. Based on the actions you chose, you should see **group** and **user** resource types.

- **group** – Choose **Add ARNs**. For **Resource in**, select the **Any account** option. Select the **Any group name with path** check box and then type the user group name **AllUsers**. Then choose **Add ARNs**.
- **user** – Select the check box next to **Any in this account**.

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 in the **JSON** editor, you can see that IAM automatically creates a new permission block granting this action permission on all resources.

8. To add another permission block, choose **Add more permissions**.

9. Choose **Select a service** and then choose **IAM**.

10. Choose **Actions allowed** and then choose **Switch to deny permissions**. When you do that, the entire block is used to deny permissions.

11. 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**

12. 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 ARNs**. For **Resource in**, select the **Any account** option. For **Any group name with path**, type the user group name **AllUsers**. Then choose **Add ARNs**.

13. Choose **Request conditions - optional** and then choose **Add another condition**. Complete the form with the following values:

- **Condition key** – Choose **aws:username**
- **Qualifier** – Choose **Default**
- **Operator** – Choose **StringNotEquals**
- **Value** – Type **srodriguez** and then choose **Add** to add another value. Type **mjackson** and then choose **Add** to add another value. Type **adesai** and then choose **Add condition**.

This condition ensures that access will be denied to the specified user 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 user group.

14. When you are finished, choose **Next**.

Note

You can switch between the **Visual** and **JSON** editor options any time. However, if you make changes or choose **Next** in the **Visual** editor option, IAM might restructure your policy to optimize it for the visual editor. For more information, see [Policy restructuring \(p. 1181\)](#).

15. On the **Review and create** page, for the **Policy Name**, type **LimitAllUserGroupManagement**. For the **Description**, type **Allows all users read-only access to a specific user group, and allows only specific users access to make changes to the user group**. Review **Permissions defined in this policy** to make sure that you have granted the intended permissions. Then choose **Create policy** to save your new policy.
16. Attach the policy to your user group. For more information, see [Adding and removing IAM identity permissions \(p. 598\)](#).

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 \(p. 565\)](#). For detailed instructions for creating a policy using a JSON document, see [the section called "Creating policies using the JSON editor" \(p. 583\)](#).

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 user 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, 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 (user 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 \(p. 581\)](#).
- To learn how to attach an IAM policy to a principal, see [Adding and removing IAM identity permissions \(p. 598\)](#).
- 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 \(p. 570\)](#).

Controlling permissions for creating, updating, and deleting customer managed policies

You can use [IAM policies \(p. 485\)](#) 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 using the JSON editor" \(p. 583\)](#).

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 using the JSON editor" \(p. 583\)](#).

For more information about using paths in the names of customer managed policies, see [Friendly names and paths \(p. 1213\)](#).

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: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 user groups and roles that include the path /TEAM-A/. The user 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 user 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 using the JSON editor” \(p. 583\)](#).

Example Example policy that allows attaching managed policies to only specific user groups or roles

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {"Effect": "Allow",  
         "Action": [  
             "iam:AttachGroupPolicy",  
             "iam:AttachRolePolicy"  
         ],  
         "Resource": [  
             "arn:aws:iam::account-id:group/TEAM-A/*",  
             "arn:aws:iam::account-id: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 \(p. 1278\)](#) 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 user 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 using the JSON editor” \(p. 583\)](#).

```
{
```

```
"Version": "2012-10-17",
"Statement": [
    "Effect": "Allow",
    "Action": [
        "iam:AttachGroupPolicy",
        "iam:AttachRolePolicy"
    ],
    "Resource": [
        "arn:aws:iam::account-id:group/TEAM-A/*",
        "arn:aws:iam::account-id:role/TEAM-A/*"
    ],
    "Condition": {"ArnLike":
        {"iam:PolicyARN": "arn:aws:iam::account-id:policy/TEAM-A/*"}
    }
}
```

This policy uses the ArnLike condition operator, but you can also use the ArnEquals condition operator because these two condition operators behave identically. For more information about ArnLike and ArnEquals, see [Amazon Resource Name \(ARN\) condition operators \(p. 1288\)](#) 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 \(p. 511\)](#).

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 \(p. 581\)](#).
- To learn how to attach an IAM policy to a principal, see [Adding and removing IAM identity permissions \(p. 598\)](#).
- 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 \(p. 188\)](#).

Note

Some services support resource-based policies as described in [Identity-based policies and resource-based policies \(p. 511\)](#) (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 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 of controlling access to other AWS resources, including other IAM resources, see [Tagging IAM resources \(p. 399\)](#).

Note

For details about case sensitivity for tag keys and tag key values, see [Case sensitivity](#).

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 \(p. 1278\)](#) 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 \(p. 523\)](#) – Control access to user or role resources based on their tags. To do this, use the `aws:ResourceTag/key-name` condition key to specify which tag key-value pair must be attached to the resource. For more information, see [Controlling access to AWS resources \(p. 523\)](#).
- [Request \(p. 524\)](#) – 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 \(p. 524\)](#).
- [Principal \(p. 522\)](#) – 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 \(p. 522\)](#) – 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 \(p. 525\)](#).

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 \(p. 581\)](#).

Note

You can also pass [session tags \(p. 417\)](#) when you assume an IAM role or federate a user. These are valid only for the length of the session.

Controlling access for IAM principals

You can 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 an identity-based policy that allows any user in this account to view the group membership for any user, including themselves, as long as they are working on the same project. This operation is allowed only when the user's resource tag and the principal's tag have the same value for the tag key `project`. 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 \(p. 581\)](#) or [edit a policy \(p. 609\)](#).

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "VisualEditor0",  
            "Effect": "Allow",  
            "Action": "iam>ListGroupsForUser",  
            "Resource": "arn:aws:iam::111222333444:user/*",  
            "Condition": {  
                "StringEquals": {"aws:ResourceTag/project": "${aws:PrincipalTag/project}"}  
            }  
        }  
    ]  
}
```

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 an identity-based 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 \(p. 581\)](#) or [edit a policy \(p. 609\)](#).

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "iam:UntagUser",  
            "Resource": "*",  
            "Condition": {"ForAllValues:StringEquals": {"aws:TagKeys": ["temporary"]}}  
        }  
    ]  
}
```

Controlling access to AWS resources using tags

You can use tags to control access to your AWS resources that support tagging, including IAM resources. You can tag IAM users and roles to control what they can access. To learn how to tag IAM users and roles, see [Tagging IAM resources \(p. 399\)](#). Additionally, you can control access to the following IAM resources: customer managed policies, IAM identity providers, instance profiles, server certificates, and virtual MFA devices. 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 \(p. 51\)](#). Use the information in the following section to control access to other AWS resources, including IAM resources, 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 \(p. 490\)](#) and specify the permissions for principal entities. For more information about policy types and uses, see [Policies and permissions in IAM \(p. 485\)](#).

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 [How IAM works \(p. 6\)](#). For details about how IAM determines whether a request is allowed, see [Policy evaluation logic \(p. 1306\)](#).

Tags are another consideration in 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 \(p. 1278\)](#) of a policy. To learn whether an AWS service supports controlling access using tags, see [AWS services that work with IAM \(p. 1224\)](#) 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 \(p. 523\)](#) – 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 \(p. 524\)](#) – 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 \(p. 525\)](#) – 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 \(p. 581\)](#).

Note

Some services allow users to specify tags when they create the resource if they have permissions to use the action that creates the resource.

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. Some services support only the service-specific version of this key and not the global version.

Warning

Do not try to control who can pass a role by tagging the role and then using the `ResourceTag` condition key in a policy with the `iam:PassRole` action. This approach does not have reliable results. 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 \(p. 279\)](#).

This example shows how you might create an identity-based policy that allows starting or stopping Amazon EC2 instances. These operations are allowed only if the instance tag `Owner` has the value of the user name. This policy defines permissions for programmatic and console access.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "ec2:StartInstances",  
                "ec2:StopInstances"  
            ],  
            "Resource": "arn:aws:ec2:*::instance/*",  
            "Condition": {  
                "StringEquals": {"aws: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 \(p. 1278\)](#).

This example shows how you might create an identity-based policy that uses the `team` principal tag in the resource ARN. The policy grants permission to delete Amazon Simple Queue Service queues, but only if the queue name starts with the team name followed by `-queue`. For example, `qa-queue` if `qa` is the team name for the team principal tag.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllQueueActions",  
            "Effect": "Allow",  
            "Action": "sns:DeleteQueue",  
            "Resource": "arn:aws:sns:us-east-2:${aws:PrincipalTag/team}-queue"  
        }  
    ]  
}
```

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 an identity-based 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.

We recommend that when you use policies to control access using tags, you use the [aws:TagKeys condition key \(p. 1364\)](#). 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 `"aws: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, or use tag policies, available with AWS Organizations. For more information see [Tag Policies](#) in the *Organizations User Guide*.

This example shows how you might create an identity-based policy that allows creating and tagging a Secrets Manager secret, but only with the tag keys `environment` or `cost-center`. The `Null` condition ensures that the condition evaluates to `false` if there are no tags in the request.

```
{
    "Effect": "Allow",
    "Action": [
        "secretsmanager:CreateSecret",
        "secretsmanager:TagResource"
    ],
    "Resource": "*",
    "Condition": {
        "Null": {
            "aws:TagKeys": "false"
        },
        "ForAllValues:StringEquals": {
            "aws:TagKeys": [
                "environment",
                "cost-center"
            ]
        }
    }
}
```

```
}
```

Cross account resource access in IAM

For some AWS services, you can grant cross-account access to your resources using IAM. To do this, you can attach a resource policy directly to the resource that you want to share, or use a role as a proxy.

To share the resource directly, the resource that you want to share must support [resource-based policies](#). Unlike an identity-based policy for a role, a resource-based policy specifies who (which principal) can access that resource.

Use a role as a proxy when you want to access resources in another account that do not support resource-based policies.

For details about the differences between these policy types, see [Identity-based policies and resource-based policies \(p. 511\)](#).

Note

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

Cross-account access using roles

Not all AWS services support resource-based policies. For these services, you can use cross-account IAM roles to centralize permission management when providing cross-account access to multiple services. A cross-account IAM role is an IAM role that includes a [trust policy](#) that allows IAM principals in another AWS account to assume the role. Put simply, you can create a role in one AWS account that delegates specific permissions to another AWS account.

For information about attaching a policy to an IAM identity, see [Managing IAM policies \(p. 581\)](#).

Note

When a principal switches to a role to temporarily use the permissions of the role, they give up their original permissions and take on the permissions assigned to the role they've assumed.

Let's take a look at the overall process as it applies to APN Partner software that needs to access a customer account.

1. The customer creates an IAM role in their own account with a policy that allows access the Amazon S3 resources that the APN partner requires. In this example, the role name is APNPartner.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:*",
            "Resource": [
                "arn:aws:s3:::bucket-name"
            ]
        }
    ]
}
```

2. Then, the customer specifies that the role can be assumed by the partner's AWS account by providing the by providing the APN Partner's AWS account ID in the [trust policy](#) for the APNPartner role.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": "arn:aws:iam::APN-account-ID:user/APN-user-name"  
            },  
            "Action": "sts:AssumeRole"  
        }  
    ]  
}
```

3. The customer gives the Amazon Resource Name (ARN) of the role to the APN partner. The ARN is the fully qualified name of the role.

```
arn:aws:iam::APN-ACCOUNT-ID:role/APNPartner
```

Note

We recommend using an external ID in multi-tenant situations. For details, see [How to use an external ID when granting access to your AWS resources to a third party \(p. 191\)](#).

4. When the APN Partner's software needs to access the customer's account, the software calls the [AssumeRole](#) API in the AWS Security Token Service with the ARN of the role in the customer's account. STS returns a temporary AWS credential that allows the software to do its work.

For another example of granting cross-account access using roles, see [Providing access to an IAM user in another AWS account that you own \(p. 188\)](#). You can also follow the [IAM tutorial: Delegate access across AWS accounts using IAM roles \(p. 40\)](#).

Cross-account access using resource-based policies

When an account accesses a resource through another account using a resource-based policy, the principal still works in the trusted account and does not have to give up their permissions to receive the role permissions. In other words, the principal continues to have access to resources in the trusted account while having 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.

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](#).

To learn whether principals in accounts outside of your zone of trust (trusted organization or account) have access to assume your roles, see [Identifying resources shared with an external entity](#).

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 \(p. 1224\)](#) 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, see [Access Control](#) in the *Amazon Simple Storage Service User 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 User Guide*.
- **Amazon Simple Notification Service (Amazon SNS) topics** — For more information, go to [Example cases for Amazon SNS access control](#) 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*.

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.

Important

In cross account access, a principal needs an Allow in the identity policy **and** the resource-based policy.

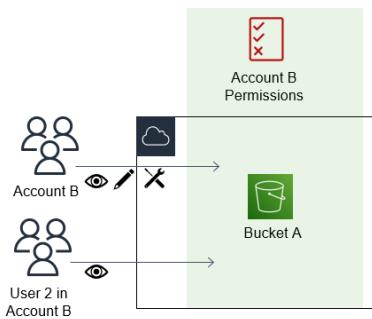
Assume that a resource-based policy allows all principals in your account full administrative access to a resource. Then you can delegate full access, read-only access, or any other partial access to principals in your AWS account. Alternatively, if the resource-based policy allows only list permissions, 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 more information about how these decisions are made, see [Determining whether a request is allowed or denied within an account](#).

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 an Amazon S3 bucket named BucketA.



1. 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.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "PrincipalAccess",  
            "Effect": "Allow",  
            "Principal": {"AWS": "arn:aws:iam::AccountB:root"},  
            "Action": "s3:*",  
            "Resource": "arn:aws:s3:::BucketA/*"  
        }  
    ]  
}
```

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.

The AccountB root user has all of the permissions that are granted to the account. Therefore, the root user has full access to BucketA.

2. In AccountB, 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.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect" : "Allow",  
            "Action" : [  
                "s3:Get*",  
                "s3>List*" ],  
            "Resource" : "arn:aws:s3:::BucketA/*"  
        }  
    ]  
}
```

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.

The AccountB administrator does not give access to User1. By default, users do not have any permissions except those that are explicitly granted, so User1 does not have access to BucketA.

IAM evaluates a principal's permissions at the time the principal makes a request. 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.

For more information about complex uses of roles, such as granting access to applications and services, see [Common scenarios for roles: Users, applications, and services \(p. 187\)](#).

Important

Give access only to entities you trust, and give the minimum level of access necessary. Whenever the trusted entity is another AWS account, any IAM principal can be granted access to your resource. 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 \(p. 484\)](#).

Example IAM identity-based policies

A [policy \(p. 485\)](#) 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 using the JSON editor" \(p. 583\)](#).

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 \(p. 532\)](#).)
- Allows enabling and disabling AWS Regions. ([View this policy \(p. 533\)](#).)
- Allows MFA-authenticated users to manage their own credentials on the **My security credentials** page. ([View this policy \(p. 533\)](#).)
- Allows specific access when using MFA during a specific range of dates. ([View this policy \(p. 537\)](#).)
- Allows users to manage their own credentials on the **My security credentials** page. ([View this policy \(p. 537\)](#).)
- Allows users to manage their own MFA device on the **My security credentials** page. ([View this policy \(p. 539\)](#).)
- Allows users to manage their own password on the **My security credentials** page. ([View this policy \(p. 541\)](#).)
- Allows users to manage their own password, access keys, and SSH public keys on the **My security credentials** page. ([View this policy \(p. 542\)](#).)
- Denies access to AWS based on the requested Region. ([View this policy \(p. 543\)](#).)
- Denies access to AWS based on the source IP address. ([View this policy \(p. 544\)](#).)

Example policy: AWS Data Exchange

- Deny access to Amazon S3 resources outside of your account except AWS Data Exchange. ([View this policy \(p. 545\)](#).)

Example policies: AWS Data Pipeline

- Denies access to pipelines that a user did not create ([View this policy \(p. 547\)](#).)

Example policies: Amazon DynamoDB

- Allows access to a specific Amazon DynamoDB table ([View this policy \(p. 547\)](#).)
- Allows access to specific Amazon DynamoDB attributes ([View this policy \(p. 548\)](#).)
- Allows item-level access to Amazon DynamoDB based on an Amazon Cognito ID ([View this policy \(p. 549\)](#).)

Example policies: Amazon EC2

- Allows attaching or detaching Amazon EBS volumes to Amazon EC2 instances based on tags ([View this policy \(p. 550\)](#).)

- Allows launching Amazon EC2 instances in a specific subnet, programmatically and in the console ([View this policy \(p. 550\)](#).)
- Allows managing Amazon EC2 security groups associated with a specific VPC, programmatically and in the console ([View this policy \(p. 551\)](#).)
- Allows starting or stopping Amazon EC2 instances a user has tagged, programmatically and in the console ([View this policy \(p. 552\)](#).)
- Allows starting or stopping Amazon EC2 instances based on resource and principal tags, programmatically and in the console ([View this policy \(p. 552\)](#).)
- Allows starting or stopping Amazon EC2 instances when the resource and principal tags match ([View this policy \(p. 553\)](#).)
- Allows full Amazon EC2 access within a specific Region, programmatically and in the console. ([View this policy \(p. 553\)](#).)
- Allows starting or stopping a specific Amazon EC2 instance and modifying a specific security group, programmatically and in the console ([View this policy \(p. 554\)](#).)
- Denies access to specific Amazon EC2 operations without MFA ([View this policy \(p. 555\)](#).)
- Limits terminating Amazon EC2 instances to a specific IP address range ([View this policy \(p. 555\)](#).)

Example policies: AWS Identity and Access Management (IAM)

- Allows access to the policy simulator API ([View this policy \(p. 556\)](#).)
- Allows access to the policy simulator console ([View this policy \(p. 557\)](#).)
- Allows assuming any roles that have a specific tag, programmatically and in the console ([View this policy \(p. 557\)](#).)
- Allows and denies access to multiple services, programmatically and in the console ([View this policy \(p. 558\)](#).)
- Allows adding a specific tag to an IAM user with a different specific tag, programmatically and in the console ([View this policy \(p. 559\)](#).)
- Allows adding a specific tag to any IAM user or role, programmatically and in the console ([View this policy \(p. 560\)](#).)
- Allows creating a new user only with specific tags ([View this policy \(p. 561\)](#).)
- Allows generating and retrieving IAM credential reports ([View this policy \(p. 562\)](#).)
- Allows managing a group's membership, programmatically and in the console ([View this policy \(p. 562\)](#).)
- Allows managing a specific tag ([View this policy \(p. 563\)](#).)
- Allows passing an IAM role to a specific service ([View this policy \(p. 564\)](#).)
- Allows read-only access to the IAM console without reporting ([View this policy \(p. 564\)](#).)
- Allows read-only access to the IAM console ([View this policy \(p. 565\)](#).)
- Allows specific users to manage a group, programmatically and in the console ([View this policy \(p. 565\)](#).)
- Allows setting the account password requirements, programmatically and in the console ([View this policy \(p. 566\)](#).)
- Allows using the policy simulator API for users with a specific path ([View this policy \(p. 567\)](#).)
- Allows using the policy simulator console for users with a specific path ([View this policy \(p. 567\)](#).)
- Allows IAM users to self-manage an MFA device. ([View this policy \(p. 568\)](#).)
- Allows IAM users to rotate their own credentials, programmatically and in the console. ([View this policy \(p. 569\)](#).)
- Allows viewing service last accessed information for an AWS Organizations policy in the IAM console. ([View this policy \(p. 570\)](#).)
- Limits managed policies that can be applied to an IAM user, group, or role ([View this policy \(p. 570\)](#).)

- Allows access to IAM policies only in your account ([View this policy \(p. 571\)](#).)

Example policies: AWS Lambda

- Allows an AWS Lambda function to access an Amazon DynamoDB table ([View this policy \(p. 571\)](#).)

Example policies: Amazon RDS

- Allows full Amazon RDS database access within a specific Region. ([View this policy \(p. 572\)](#).)
- Allows restoring Amazon RDS databases, programmatically and in the console ([View this policy \(p. 573\)](#).)
- Allows tag owners full access to Amazon RDS resources that they have tagged ([View this policy \(p. 573\)](#).)

Example policies: Amazon S3

- Allows an Amazon Cognito user to access objects in their own Amazon S3 bucket ([View this policy \(p. 575\)](#).)
- Allows federated users to access their own home directory in Amazon S3, programmatically and in the console ([View this policy \(p. 576\)](#).)
- 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 \(p. 577\)](#).)
- Allows IAM users to access their own home directory in Amazon S3, programmatically and in the console ([View this policy \(p. 578\)](#).)
- Allows a user to manage a single Amazon S3 bucket and denies every other AWS action and resource ([View this policy \(p. 579\)](#).)
- Allows Read and Write access to a specific Amazon S3 bucket ([View this policy \(p. 579\)](#).)
- Allows Read and Write access to a specific Amazon S3 bucket, programmatically and in the console ([View this policy \(p. 580\)](#).)

AWS: Allows access based on date and time

This example shows how you might create an identity-based 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 programmatically from the AWS API or AWS CLI. 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 \(p. 581\)](#) or [edit a policy \(p. 609\)](#).

To learn about using multiple conditions within the Condition block of an IAM policy, see [Multiple values in a condition \(p. 1281\)](#).

```
{  
    "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"}  
            }  
        }  
    ]  
}
```

```
        }
    ]
}
```

Note

You cannot use a policy variable with the Date condition operator. To learn more see [Condition element \(p. 1302\)](#)

AWS: Allows enabling and disabling AWS Regions

This example shows how you might create an identity-based policy that allows an administrator to enable and disable the Asia Pacific (Hong Kong) Region (ap-east-1). This policy defines permissions for programmatic and console access. 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 \(p. 581\)](#) or [edit a policy \(p. 609\)](#).

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": [
                "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 an identity-based policy that allows IAM users that are authenticated using [multi-factor authentication \(MFA\) \(p. 114\)](#) 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 \(p. 537\)](#).

To learn how users can access the **My security credentials** page, see [How IAM users change their own password \(console\) \(p. 102\)](#).

Note

- This example policy does not allow users to reset a password while signing in to the AWS Management Console for the first time. We recommend 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? \(p. 1173\)](#). This also prevents users with an expired password from resetting their password during sign in. You can allow this by adding `iam:ChangePassword` and `iam:GetAccountPasswordPolicy` to the statement `DenyAllExceptListedIfNoMFA`. However, we do not recommend this because allowing users to change their password without MFA can be a security risk.
- If you intend to use this policy for programmatic access you must call [GetSessionToken](#) to authenticate with MFA. For more information, see [Configuring MFA-protected API access \(p. 145\)](#).

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.
 - `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 user can also retrieve information about when the specified access key was last used.
- 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 their own virtual MFA device. The resource ARN in this statement allows the user to create an MFA device with any name, but the other statements in the policy only allow the user to attach the device to the currently signed-in user.
- 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 \(p. 1274\)](#). 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 \(p. 1344\)](#) 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:GetLoginProfile` and `iam:UpdateLoginProfile` 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 `iam:UpdateLoginProfile` action to the `DenyAllExceptListedIfNoMFA` statement.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowViewAccountInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetAccountPasswordPolicy",
                "iam>ListVirtualMFADevices"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowManageOwnPasswords",
            "Effect": "Allow",
            "Action": [
                "iam:ChangePassword",
                "iam:GetUser"
            ],
            "Resource": "arn:aws:iam::*:user/${aws:username}"
        },
        {
            "Sid": "AllowManageOwnAccessKeys",
            "Effect": "Allow",
            "Action": [
                "iam>CreateAccessKey",
                "iam>DeleteAccessKey",
                "iam>ListAccessKeys",
                "iam:UpdateAccessKey",
                "iam:GetAccessKeyLastUsed"
            ],
            "Resource": "arn:aws:iam::*:user/${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/${aws:username}"
},
{
    "Sid": "AllowManageOwnVirtualMFADevice",
    "Effect": "Allow",
    "Action": [
        "iam>CreateVirtualMFADevice"
    ],
    "Resource": "arn:aws:iam::*:mfa/*"
},
{
    "Sid": "AllowManageOwnUserMFA",
    "Effect": "Allow",
    "Action": [
        "iam:DeactivateMFADevice",
        "iam:EnableMFADevice",
        "iam>ListMFADevices",
        "iam:ResyncMFADevice"
    ],
    "Resource": "arn:aws:iam::*:user/${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 an identity-based 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 programmatically from the AWS API or AWS CLI. 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 \(p. 581\)](#) or [edit a policy \(p. 609\)](#).

To learn about using multiple conditions within the Condition block of an IAM policy, see [Multiple values in a condition \(p. 1281\)](#).

```
{  
    "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 an identity-based 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 \(p. 533\)](#).

To learn how users can access the **My security credentials** page, see [How IAM users change their own password \(console\) \(p. 102\)](#).

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 user can also retrieve information about when the specified access key was last used.
- 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**, **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:ChangePassword",
                "iam:GetUser"
            ],
            "Resource": "arn:aws:iam::*:user/${aws:username}"
        },
        {
            "Sid": "AllowManageOwnAccessKeys",
            "Effect": "Allow",
            "Action": [
                "iam:CreateAccessKey",
                "iam:DeleteAccessKey",
                "iam>ListAccessKeys",
                "iam:UpdateAccessKey",
                "iam:GetAccessKeyLastUsed"
            ],
            "Resource": "arn:aws:iam::*:user/${aws:username}"
        },
        {
            "Sid": "AllowManageOwnSigningCertificates",
            "Effect": "Allow",
            "Action": [
                "iam>DeleteSigningCertificate",
                "iam>ListSigningCertificates",
                "iam:UpdateSigningCertificate"
            ],
            "Resource": "*"
        }
    ]
}
```

```

        "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 an identity-based policy that allows IAM users that are authenticated through [multi-factor authentication \(MFA\) \(p. 114\)](#) 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 \(p. 533\)](#).

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 \(console\) \(p. 102\)](#).

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 their own virtual MFA device. The resource ARN in this statement allows the user to create an MFA device with any name, but the other statements in the policy only allow the user to attach the device to the currently signed-in user.

- 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 \(p. 1274\)](#). 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 a virtual 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 virtual MFA device using the AWS CLI or AWS API. For more information, see [I am not authorized to perform: iam>DeleteVirtualMFADevice \(p. 1172\)](#).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowViewAccountInfo",
            "Effect": "Allow",
            "Action": "iam>ListVirtualMFADevices",
            "Resource": "*"
        },
        {
            "Sid": "AllowManageOwnVirtualMFADevice",
            "Effect": "Allow",
            "Action": [
                "iam>CreateVirtualMFADevice"
            ],
            "Resource": "arn:aws:iam::*:mfa/*"
        },
        {
            "Sid": "AllowManageOwnUserMFA",
            "Effect": "Allow",
            "Action": [
                "iam>DeactivateMFADevice",
                "iam>EnableMFADevice",
                "iam GetUser",
                "iam>ListMFADevices",
                "iam>SetMFADevice"
            ]
        }
    ]
}
```

```

        "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 IAM users to change their own console password on the My security credentials page

This example shows how you might create an identity-based 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 \(p. 533\)](#). 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 \(p. 537\)](#).

To learn how users can access the [My security credentials](#) page, see [How IAM users change their own password \(console\) \(p. 102\)](#).

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. It also does not allow users to change their password on their own user page. To allow this, add the `iam>GetLoginProfile` and `iam>UpdateLoginProfile` actions to the `ChangeOwnPasswords` 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 an identity-based 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 \(p. 533\)](#). 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 \(p. 537\)](#).

To learn how users can access the **My security credentials** page, see [How IAM users change their own password \(console\) \(p. 102\)](#).

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 user can also retrieve information about when the specified access key was last used.
- 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: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:ChangePassword",
            "iam:GetUser"
        ],
        "Resource": "arn:aws:iam::*:user/${aws:username}"
    },
    {
        "Sid": "AllowManageOwnAccessKeys",
        "Effect": "Allow",
        "Action": [
            "iam>CreateAccessKey",
            "iam>DeleteAccessKey",
            "iam>ListAccessKeys",
            "iam:UpdateAccessKey",
            "iam:GetAccessKeyLastUsed"
        ],
        "Resource": "arn:aws:iam::*:user/${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 an identity-based policy that denies access to any actions outside the Regions specified using the [aws:RequestedRegion condition key](#), except for actions in the services specified using NotAction. This policy defines permissions for programmatic and console access. 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 \(p. 581\)](#) or [edit a policy \(p. 609\)](#).

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. Some other global services, such as AWS Chatbot and AWS Device Farm, are global services with endpoints that are physically located in the us-west-2 region. 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 \(p. 1274\)](#).

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 an identity-based 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 defines permissions for programmatic and console access. 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 \(p. 581\)](#) or [edit a policy \(p. 609\)](#).

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 \(p. 1290\)](#) 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.

For more information about using the `aws:SourceIp` and `aws:ViaAWSService` condition keys, including information about when the `aws:SourceIp` key may not work in your policy, see [AWS global condition context keys \(p. 1338\)](#).

Note

AWS CloudShell originated API requests don't originate from your company's IP address ranges and are not called via an AWS service, meaning CloudShell requests are denied. Add the following condition to exclude CloudShell requests from the deny statement.

```
"StringNotLike": {  
    "aws: userAgent": "*exec-env/CloudShell*"  
}
```

This condition does not deny requests where the environment variable is defined as `AWS_EXECUTION_ENV=CloudShell`. We recommend you use specific client applications in [aws:UserAgent](#) value.

```
{  
    "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: Deny access to Amazon S3 resources outside your account except AWS Data Exchange

This example shows how you might create an identity-based policy that denies access to all resources in AWS that don't belong to your account, except for the resources that AWS Data Exchange requires for normal operation. 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 \(p. 581\)](#) or [edit a policy \(p. 609\)](#).

You can create a similar policy to restrict access to resources within an organization or an organizational unit, while accounting for AWS Data Exchange owned resources by using the condition keys `aws:ResourceOrgPaths` and `aws:ResourceOrgID`.

If you use AWS Data Exchange in your environment, the service creates and interacts with resources such as Amazon S3 buckets owned by the service account. For example, AWS Data Exchange sends requests to Amazon S3 buckets owned by the AWS Data Exchange service on behalf of the IAM principal (user or role) invoking the AWS Data Exchange APIs. In that case, using `aws:ResourceAccount`, `aws:ResourceOrgPaths`, or `aws:ResourceOrgID` in a policy, without accounting for AWS Data Exchange owned resources, denies access to the buckets owned by the service account.

- The statement, `DenyAllAwsResourcesOutsideAccountExceptS3`, uses the `NotAction` element with the [Deny](#) effect which explicitly denies access to every action not listed in the statement that also do not belong to the listed account. The `NotAction` element indicates the exceptions to this

statement. These actions are the exception to this statement because if the actions are performed on resources created by AWS Data Exchange, the policy denies them.

- The statement, DenyAllS3ResourcesOutsideAccountExceptDataExchange, uses a combination of the ResourceAccount and CalledVia conditions to deny access to the three Amazon S3 actions excluded in the previous statement. The statement denies the actions if resources do not belong in the listed account and if the calling service is not AWS Data Exchange. The statement does not deny the actions if either the resource belongs to the listed account or the listed service principal, dataexchange.amazonaws.com, performs the operations.

Important

This policy does not allow any actions. It uses the Deny effect which explicitly denies access to all of the resources listed in the statement that do not belong to the listed account. Use this policy in combination with other policies that allow access to specific resources.

The following example shows how you can configure the policy to allow access to the required Amazon S3 buckets.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyAllAwsResourcesOutsideAccountExceptAmazonS3",
      "Effect": "Deny",
      "NotAction": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:PutObjectAcl"
      ],
      "Resource": "*",
      "Condition": {
        "StringNotEquals": {
          "aws:ResourceAccount": [
            "111122223333"
          ]
        }
      }
    },
    {
      "Sid": "DenyAllS3ResourcesOutsideAccountExceptDataExchange",
      "Effect": "Deny",
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:PutObjectAcl"
      ],
      "Resource": "*",
      "Condition": {
        "StringNotEquals": {
          "aws:ResourceAccount": [
            "111122223333"
          ]
        },
        "ForAllValues:StringNotEquals": {
          "aws:CalledVia": [
            "dataexchange.amazonaws.com"
          ]
        }
      }
    }
  ]
}
```

AWS Data Pipeline: Denies access to DataPipeline pipelines that a user did not create

This example shows how you might create an identity-based 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 programmatically from the AWS API or AWS CLI.

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 an identity-based policy that allows full access to the `MyTable` DynamoDB table. This policy grants the permissions necessary to complete this action programmatically from the AWS API or AWS CLI. 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 \(p. 581\)](#) or [edit a policy \(p. 609\)](#).

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 an identity-based policy that allows access to the specific DynamoDB attributes. This policy grants the permissions necessary to complete this action programmatically from the AWS API or AWS CLI. 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 \(p. 581\)](#) or [edit a policy \(p. 609\)](#).

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 \(p. 1281\)](#).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb>GetItem",
                "dynamodb>BatchGetItem",
                "dynamodb>Select*"
            ],
            "Resource": "arn:aws:dynamodb:Region:Account:table/MyTable/*"
        }
    ]
}
```

```

        "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 an identity-based 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 programmatically from the AWS API or AWS CLI. 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 \(p. 581\)](#) or [edit a policy \(p. 609\)](#).

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 Amazon EBS volumes to EC2 instances based on tags

This example shows how you might create an identity-based 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=Development`). This policy grants the permissions necessary to complete this action programmatically from the AWS API or AWS CLI. 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 \(p. 581\)](#) or [edit a policy \(p. 609\)](#).

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:*::instance/*",  
            "Condition": {  
                "StringEquals": {"aws:ResourceTag/Department": "Development"}  
            }  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "ec2:AttachVolume",  
                "ec2:DetachVolume"  
            ],  
            "Resource": "arn:aws:ec2:*::volume/*",  
            "Condition": {  
                "StringEquals": {"aws: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 an identity-based policy that allows listing information for all EC2 objects and launching EC2 instances in a specific subnet. This policy defines permissions for programmatic and console access. 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 \(p. 581\)](#) or [edit a policy \(p. 609\)](#).

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "ec2:Describe*",  
                "ec2:GetConsole*"  
            ],  
            "Condition": {  
                "StringEquals": {"aws:ResourceTag/SubnetId": "subnet-00000000"}  
            }  
        }  
    ]  
}
```

```

        "Resource": "*"
    },
    {
        "Effect": "Allow",
        "Action": "ec2:RunInstances",
        "Resource": [
            "arn:aws:ec2:*::subnet/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 with a specific tag key-value pair programmatically and in the console

This example shows how you might create an identity-based policy that grants users permission to take certain actions for security groups that have the same tag. This policy grants permissions to view security groups in the Amazon EC2 console, add and remove inbound and outbound rules, and list and modify rule descriptions for existing security groups with the tag `Department=Test`. This policy defines permissions for programmatic and console access. 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 \(p. 581\)](#) or [edit a policy \(p. 609\)](#).

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSecurityGroupRules",
                "ec2:DescribeTags"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AuthorizeSecurityGroupIngress",
                "ec2:RevokeSecurityGroupIngress",
                "ec2:AuthorizeSecurityGroupEgress",
                "ec2:RevokeSecurityGroupEgress",
                "ec2:ModifySecurityGroupRules",
                "ec2:UpdateSecurityGroupRuleDescriptionsIngress",
                "ec2:UpdateSecurityGroupRuleDescriptionsEgress"
            ],
            "Resource": [
                "arn:aws:ec2:region:111122223333:security-group/*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Department": "Test"
                }
            }
        },
        {
            "Effect": "Allow",

```

```

        "Action": [
            "ec2:ModifySecurityGroupRules"
        ],
        "Resource": [
            "arn:aws:ec2:region:111122223333:security-group-rule/*"
        ]
    }
}

```

Amazon EC2: Allows starting or stopping EC2 instances a user has tagged, programmatically and in the console

This example shows how you might create an identity-based 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 defines permissions for programmatic and console access.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:StartInstances",
                "ec2:StopInstances"
            ],
            "Resource": "arn:aws:ec2:*:*:instance/*",
            "Condition": {
                "StringEquals": {
                    "aws: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 an identity-based 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 programmatically from the AWS API or AWS CLI. 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 \(p. 581\)](#) or [edit a policy \(p. 609\)](#).

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"
        ],
        "Resource": "arn:aws:ec2:region:account-id:instance/*",
        "Condition": {
            "StringEquals": {
                "aws:ResourceTag/ProjectDepartment

```

EC2: Start or stop instances based on matching principal and resource tags

This example shows how you might create an identity-based 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 programmatically from the AWS API or AWS CLI. 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 \(p. 581\)](#) or [edit a policy \(p. 609\)](#).

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"
            ],
            "Resource": "*",
            "Condition": {"StringEquals": {
                "aws: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 an identity-based policy that allows full EC2 access within a specific Region. This policy defines permissions for programmatic and console access. 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 \(p. 581\)](#) or [edit a policy \(p. 609\)](#). 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 an identity-based policy that allows starting or stopping a specific EC2 instance and modifying a specific security group. This policy defines permissions for programmatic and console access. 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 \(p. 581\)](#) or [edit a policy \(p. 609\)](#).

```
{  
    "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 an identity-based 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 \(MFA\) \(p. 114\)](#). 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 \(p. 435\)](#).

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 \(p. 1288\)](#).

```
{  
    "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 an identity-based 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 programmatically from the AWS API or AWS CLI. 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 \(p. 581\)](#) or [edit a policy \(p. 609\)](#).

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” \(p. 1309\)](#).

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 \(p. 1338\)](#).

```
{
    "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 an identity-based 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 programmatically from the AWS API or AWS CLI.

```
{
    "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 \(p. 557\)](#).

IAM: Access the policy simulator console

This example shows how you might create an identity-based 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 programmatically from the AWS API or AWS CLI.

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 an identity-based 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 programmatically from the AWS API or AWS CLI. 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 \(p. 581\)](#) or [edit a policy \(p. 609\)](#).

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 \(p. 188\)](#).

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Action": [  
                "iam:AssumeRole",  
                "iam:ListRoles",  
                "iam:ListRoleTags",  
                "iam:TagRole",  
                "iam:UntagRole"  
            ],  
            "Effect": "Allow",  
            "Resource": "arn:aws:iam::123456789012:role/Project=ExampleCorpABC"  
        }  
    ]  
}
```

```
{
    "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 an identity-based 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 defines permissions for programmatic and console access. 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 \(p. 581\)](#) or [edit a policy \(p. 609\)](#).

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 \(p. 505\)](#). 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:*",
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "*"
        },
        {
            "Sid": "DenyS3Logs",
            "Effect": "Deny",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:s3:::aws-logs-*/*"
        },
        {
            "Sid": "DenyEC2Production",
            "Effect": "Deny",
            "Action": [
                "ec2:RunInstances"
            ],
            "Resource": "arn:aws:ec2:us-west-2:1234567890abcdef0:instance/*"
        }
    ]
}
```

```

        "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 an identity-based 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 defines permissions for programmatic and console access. 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 \(p. 581\)](#) or [edit a policy \(p. 609\)](#).

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 [Multivalued context keys \(p. 1294\)](#).

```
{
    "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/JobFunctionDepartmentDepartment"}
        }
    ]
}
```

IAM: Add a specific tag with specific values

This example shows how you might create an identity-based 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 defines permissions for programmatic and console access. 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 \(p. 581\)](#) or [edit a policy \(p. 609\)](#).

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 [Multivalued context keys \(p. 1294\)](#).

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

```

        "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 an identity-based 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 programmatically from the AWS API or AWS CLI. 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 \(p. 581\)](#) or [edit a policy \(p. 609\)](#).

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 \(p. 1288\)](#).

The second condition uses the `ForAllValues:StringEquals` condition operator. The condition returns true if there's a match between 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 [Multivalued context keys \(p. 1294\)](#).

```
{
    "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 an identity-based 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 programmatically from the AWS API or AWS CLI.

For more information about credential reports, see [Getting credential reports for your AWS account \(p. 164\)](#).

```
{
    "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 an identity-based policy that allows updating the membership of the group called MarketingTeam. This policy defines permissions for programmatic and console access. 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 \(p. 581\)](#) or [edit a policy \(p. 609\)](#).

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 an identity-based policy that allows adding and removing the IAM tag with the tag key Department from IAM entities (users and roles). This policy does not limit the value of the Department tag. This policy grants the permissions necessary to complete this action programmatically from the AWS API or AWS CLI. 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 \(p. 581\)](#) or [edit a policy \(p. 609\)](#).

```
{
    "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 an identity-based policy that allows passing any IAM service role to the Amazon CloudWatch service. This policy grants the permissions necessary to complete this action programmatically from the AWS API or AWS CLI. 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 \(p. 581\)](#) or [edit a policy \(p. 609\)](#).

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 \(p. 1224\)](#) 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 \(p. 279\)](#).

```
{  
    "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 an identity-based policy that allows IAM users to perform any IAM action that begins with the string `Get` or `List`. 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 cannot be used to generate reports or service last accessed details. For a different policy that allows this, see [IAM: Allows read-only access to the IAM console \(p. 565\)](#).

```
{  
    "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 an identity-based 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 that includes permissions to generate reports or service last accessed details. For a different policy that does not allow generating actions, see [IAM: Allows read-only access to the IAM console without reporting \(p. 564\)](#).

```
{
    "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 an identity-based policy that allows specific IAM users to manage the AllUsers group. This policy defines permissions for programmatic and console access. 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 \(p. 581\)](#) or [edit a policy \(p. 609\)](#).

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:*Group*",
            "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 an identity-based 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 defines permissions for programmatic and console access.

To learn how to set the account password requirements policy for your account, see [Setting an account password policy for IAM users \(p. 94\)](#).

```
{
```

```
{  
    "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 an identity-based 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 programmatically from the AWS API or AWS CLI. 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 \(p. 581\)](#) or [edit a policy \(p. 609\)](#).

```
{  
    "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 \(p. 567\)](#).

IAM: Access the policy simulator console based on user path

This example shows how you might create an identity-based 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 programmatically from the AWS API or AWS CLI. 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 \(p. 581\)](#) or [edit a policy \(p. 609\)](#).

You can access the IAM Policy Simulator at: <https://pollicysim.aws.amazon.com/>

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Action": [  
                "iam:GetPolicy",  
                "iam:GetUserPolicy"  
            ],  
            "Effect": "Allow",  
            "Resource": "*"  
        },  
        {  
            "Action": [  
                "iam:ListPolicies",  
                "iam:ListUserPolicies"  
            ],  
            "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 an identity-based policy that allows IAM users to self-manage their [multi-factor authentication \(MFA\) \(p. 114\)](#) device. This policy grants the permissions necessary to complete this action programmatically from the AWS API or AWS CLI.

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. 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": "AllowUserToCreateVirtualMFADevice",  
            "Effect": "Allow",  
            "Action": [  
                "iam>CreateVirtualMFADevice"  
            ],  
            "Resource": "arn:aws:iam::*:mfa/*"  
        },  
        {  
            "Sid": "AllowUserToManageTheirOwnMFA",  
            "Effect": "Allow",  
            "Action": [  
                "iam>EnableMFADevice",  
                "iam>ListMFADevices",  
                "iam>ResyncMFADevice"  
            ],  
            "Resource": "arn:aws:iam::*:user/${aws:username}"  
        },  
        {  
            "Sid": "AllowUserToDeleteTheirOwnMFAOnlyWhenUsingMFA",  
            "Effect": "Allow",  
            "Action": [  
                "iam>DeactivateMFADevice"  
            ],  
            "Resource": [  
                "arn:aws:iam::*:user/${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 an identity-based policy that allows IAM users to rotate their own access keys, signing certificates, service specific credentials, and passwords. This policy defines permissions for programmatic and console access.

```
{
    "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"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        }
    ]
}
```

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” \(p. 102\)](#).

IAM: View service last accessed information for an Organizations policy

This example shows how you might create an identity-based 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 defines permissions for programmatic and console access. 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 \(p. 581\)](#) or [edit a policy \(p. 609\)](#).

For important information about last accessed information, including permissions required, troubleshooting, and supported Regions, see [Refining permissions in AWS using last accessed information \(p. 616\)](#).

```
{  
    "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 an identity-based 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 programmatically from the AWS API or AWS CLI. 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 \(p. 581\)](#) or [edit a policy \(p. 609\)](#).

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "iam:AttachUserPolicy",  
                "iam:DetachUserPolicy"  
            ],  
            "Resource": "<ARN of the user, group, or role>"  
        }  
    ]  
}
```

```
    "Resource": "*",
    "Condition": {
        "ArnEquals": {
            "iam:PolicyARN": [
                "arn:aws:iam::*:policy/policy-name-1",
                "arn:aws:iam::*:policy/policy-name-2"
            ]
        }
    }
}
```

AWS: Deny access to resources outside your account except AWS managed IAM policies

Using `aws:ResourceAccount` in your identity-based policies can impact the user or the role's ability to utilize some services that require interaction with resources in accounts owned by a service.

You can create a policy with an exception to allow for AWS managed IAM policies. A service-managed account outside of your AWS Organizations owns Managed IAM Policies. There are four IAM actions that list and retrieve AWS-managed policies. Use these actions in the [NotAction](#) element of the statement. `AllowAccessToS3ResourcesInSpecificAccountsAndSpecificService1` in the policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAccessToResourcesInSpecificAccountsAndSpecificService1",
            "Effect": "Deny",
            "NotAction": [
                "iam:GetPolicy",
                "iam:GetPolicyVersion",
                "iam>ListEntitiesForPolicy",
                "iam>ListPolicies"
            ],
            "Resource": "*",
            "Condition": {
                "StringNotEquals": {
                    "aws:ResourceAccount": [
                        "111122223333"
                    ]
                }
            }
        }
    ]
}
```

AWS Lambda: Allows a Lambda function to access an Amazon DynamoDB table

This example shows how you might create an identity-based 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 \(p. 581\)](#) or [edit a policy \(p. 609\)](#).

To use this policy, attach the policy to a Lambda [service role \(p. 255\)](#). 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 an identity-based policy that allows full RDS database access within a specific Region. This policy grants the permissions necessary to complete this action programmatically from the AWS API or AWS CLI. 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 \(p. 581\)](#) or [edit a policy \(p. 609\)](#).

```
{
    "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 an identity-based policy that allows restoring RDS databases. This policy defines permissions for programmatic and console access.

```
{
    "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 an identity-based policy that allows tag owners full access to RDS resources that they have tagged. This policy grants the permissions necessary to complete this action programmatically from the AWS API or AWS CLI.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "rds:Describe*",
                "rds>List*"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": [
                "rds>DeleteDBInstance",
                "rds:RebootDBInstance",
                "rds:RestoreDBInstanceToPointInTime"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

```

        "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 an identity-based 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 programmatically from the AWS API or AWS CLI. 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 \(p. 581\)](#) or [edit a policy \(p. 609\)](#).

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>/${cognito-identity.amazonaws.com:sub}/*"
                    ]
                }
            }
        },
        {
            "Sid": "ReadWriteDeleteYourObjects",
            "Effect": "Allow",
            "Action": [
                "s3>DeleteObject",
                "s3>GetObject",
                "s3>PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::<bucket-name>/cognito/<application-name>/${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 [Amazon Cognito User Guide](#).

Amazon S3: Allows federated users access to their S3 home directory, programmatically and in the console

This example shows how you might create an identity-based 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 defines permissions for programmatic and console access. 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 \(p. 581\)](#) or [edit a policy \(p. 609\)](#).

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 \(p. 1218\)](#). 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": [  
        {  
            "Sid": "S3ConsoleAccess",  
            "Effect": "Allow",  
            "Action": [  
                "s3:GetAccountPublicAccessBlock",  
                "s3:GetBucketAcl",  
                "s3:GetBucketLocation",  
                "s3:GetBucketPolicyStatus",  
                "s3:GetBucketPublicAccessBlock",  
                "s3>ListAccessPoints",  
                "s3>ListAllMyBuckets"  
            ],  
            "Resource": "*"  
        },  
        {  
            "Sid": "ListObjectsInBucket",  
            "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 an identity-based 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 \(MFA\) \(p. 114\)](#) 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 \(p. 581\)](#) or [edit a policy \(p. 609\)](#).

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 \(p. 435\)](#) 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 an identity-based 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 defines permissions for programmatic and console access. 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 \(p. 581\)](#) or [edit a policy \(p. 609\)](#).

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 \(p. 1303\)](#).

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "S3ConsoleAccess",
            "Effect": "Allow",
            "Action": [
                "s3:GetAccountPublicAccessBlock",
                "s3:GetBucketAcl",
                "s3:GetBucketLocation",
                "s3:GetBucketPolicyStatus",
                "s3:GetBucketPublicAccessBlock",
                "s3>ListAccessPoints",
                "s3>ListAllMyBuckets"
            ],
            "Resource": "*"
        },
        {
            "Sid": "ListObjectsInBucket",
            "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: Restrict management to a specific S3 bucket

This example shows how you might create an identity-based policy that restricts management of an Amazon S3 bucket to that specific bucket. This policy grants permission to perform all Amazon S3 actions, but deny access to every AWS service except Amazon S3. See the following example. According to this policy, you can only access Amazon S3 actions that you can perform on an S3 bucket or S3 object resource. This policy grants the permissions necessary to complete this action programmatically from the AWS API or AWS CLI. 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 \(p. 581\)](#) or [edit a policy \(p. 609\)](#).

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" \(p. 1309\)](#).

Warning

[NotAction \(p. 1274\)](#) and [NotResource \(p. 1277\)](#) 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 an identity-based policy that allows Read and Write access to objects in a specific S3 bucket. This policy grants the permissions necessary to complete this action programmatically from the AWS API or AWS CLI. 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 \(p. 581\)](#) or [edit a policy \(p. 609\)](#).

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 \(p. 580\)](#).

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 an identity-based policy that allows Read and Write access to objects in a specific S3 bucket. This policy defines permissions for programmatic and console access. 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 \(p. 581\)](#) or [edit a policy \(p. 609\)](#).

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": "S3ConsoleAccess",
            "Effect": "Allow",
            "Action": [
                "s3>GetAccountPublicAccessBlock",
                "s3>GetBucketAcl",
                "s3>GetBucketLocation",
                "s3>GetBucketPolicyStatus",
                "s3>GetBucketPublicAccessBlock",
                "s3>ListAccessPoints",
                "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, validate the 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 \(p. 485\)](#).
- For general information about using policies within IAM, see [Access management for AWS resources \(p. 484\)](#).
- For information about how permissions are evaluated when multiple policies are in effect for a given IAM identity, see [Policy evaluation logic \(p. 1306\)](#).
- The number and size of IAM resources in an AWS account are limited. For more information, see [IAM and AWS STS quotas \(p. 1220\)](#).

Topics

- [Creating IAM policies \(p. 581\)](#)
- [Validating IAM policies \(p. 588\)](#)
- [Generate policies based on access activity \(p. 589\)](#)
- [Testing IAM policies with the IAM policy simulator \(p. 589\)](#)
- [Adding and removing IAM identity permissions \(p. 598\)](#)
- [Versioning IAM policies \(p. 606\)](#)
- [Editing IAM policies \(p. 609\)](#)
- [Deleting IAM policies \(p. 613\)](#)
- [Refining permissions in AWS using last accessed information \(p. 616\)](#)

Creating IAM policies

A [policy \(p. 485\)](#) 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 \(p. 598\)](#).

Use customer managed policies instead of inline policies. 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 \(p. 1035\)](#), which is granting only the permissions required to perform specific job tasks.

When you create or edit IAM policies, AWS can automatically perform policy validation to help you create an effective policy with least privilege in mind. In the AWS Management Console, IAM identifies JSON syntax errors, while IAM Access Analyzer provides additional policy checks with recommendations to help you further refine your policies. To learn more about policy validation, see [Validating IAM policies \(p. 588\)](#). To learn more about IAM Access Analyzer policy checks and actionable recommendations, see [IAM Access Analyzer policy validation](#).

You can use the AWS Management Console, AWS CLI, or AWS API to create customer managed policies in IAM. For more information about using AWS CloudFormation templates to add or update policies, see [AWS Identity and Access Management resource type reference](#) in the *AWS CloudFormation User Guide*.

Topics

- [Creating IAM policies \(console\) \(p. 582\)](#)
- [Creating IAM policies \(AWS CLI\) \(p. 586\)](#)
- [Creating IAM policies \(AWS API\) \(p. 587\)](#)

Creating IAM policies (console)

A [policy \(p. 485\)](#) 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 \(p. 582\)](#)
- [Creating policies using the JSON editor \(p. 583\)](#)
- [Creating policies with the visual editor \(p. 584\)](#)
- [Importing existing managed policies \(p. 585\)](#)

Creating IAM policies

You can create a customer managed policy in the AWS Management Console using one of the following methods:

- [JSON \(p. 583\)](#) — Paste and customize a published [example identity-based policy \(p. 529\)](#).
- [Visual editor \(p. 584\)](#) — 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 \(p. 585\)](#) — 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 AWS STS quotas \(p. 1220\)](#).

Creating policies using the JSON editor

You can type or paste policies in JSON by choosing the **JSON** option. This method is useful for copying an [example policy \(p. 529\)](#) to use in your account. Or, you can type your own JSON policy document in the JSON editor. You can also use the **JSON** option to toggle between the visual editor and JSON to compare the views.

When you create or edit a policy in the JSON editor, IAM performs policy validation to help you create an effective policy. IAM identifies JSON syntax errors, while IAM Access Analyzer provides additional policy checks with actionable recommendations to help you further refine the policy.

A JSON [policy \(p. 485\)](#) 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 \(p. 485\)](#). For information about the IAM policy language, see [IAM JSON policy reference \(p. 1260\)](#).

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. In the **Policy editor** section, choose the **JSON** option.
5. Type or paste a JSON policy document. For details about the IAM policy language, see [IAM JSON policy reference \(p. 1260\)](#).
6. Resolve any security warnings, errors, or general warnings generated during [policy validation \(p. 588\)](#), and then choose **Next**.

Note

You can switch between the **Visual** and **JSON** editor options anytime. However, if you make changes or choose **Next** in the **Visual** editor, IAM might restructure your policy to optimize it for the visual editor. For more information, see [Policy restructuring \(p. 1181\)](#).

7. (Optional) When you create or edit a policy in the AWS Management Console, you can generate a JSON or YAML policy template that you can use in AWS CloudFormation templates.

To do this, in the **Policy editor** choose **Actions**, and then choose **Generate CloudFormation template**. To learn more about AWS CloudFormation see [AWS Identity and Access Management resource type reference](#) in the AWS CloudFormation User Guide.

8. When you are finished adding permissions to the policy, choose **Next**.
9. On the **Review and create** page, type a **Policy Name** and a **Description** (optional) for the policy that you are creating. Review **Permissions defined in this policy** to see the permissions that are granted by your policy.
10. (Optional) Add metadata to the policy by attaching tags as key-value pairs. For more information about using tags in IAM, see [Tagging IAM resources \(p. 399\)](#).
11. 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 \(p. 598\)](#).

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" \(p. 515\)](#).

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. In the **Policy editor** section, find the **Select a service** section, 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 more permissions**.
5. In **Actions allowed**, 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 \(p. 1306\)](#), 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 \(p. 520\)](#), all resources are allowed and you cannot edit this section.

If you chose one or more actions that support [resource-level permissions \(p. 520\)](#), 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 ARNs** 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 \(p. 1276\)](#).
 - Choose **Any in this account** next to a resource to grant permissions to any resources of that type.
 - Choose **All** to choose all resources for the service.
7. (Optional) Choose **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](#) in the *Service Authorization Reference*.

You can choose conditions in the following ways:

- Use check boxes to select commonly used conditions.
- Choose **Add another 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**. You can consider the values as being connected by a logical "OR" operator. When you are finished, choose **Add condition**.

To add more than one condition, choose **Add another 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 \(p. 1278\)](#) in the [IAM JSON policy reference \(p. 1260\)](#).

8. To add more permission blocks, choose **Add more permissions**. For each block, repeat steps 2 through 5.

Note

You can switch between the **Visual** and **JSON** editor options anytime. However, if you make changes or choose **Next** in the **Visual** editor, IAM might restructure your policy to optimize it for the visual editor. For more information, see [Policy restructuring \(p. 1181\)](#).

9. (Optional) When you create or edit a policy in the AWS Management Console, you can generate a JSON or YAML policy template that you can use in AWS CloudFormation templates.

To do this, in the **Policy editor** choose **Actions**, and then choose **Generate CloudFormation template**. To learn more about AWS CloudFormation see [AWS Identity and Access Management resource type reference](#) in the AWS CloudFormation User Guide.

10. When you are finished adding permissions to the policy, choose **Next**.
11. On the **Review and create** page, type a **Policy Name** and a **Description** (optional) for the policy that you are creating. Review the **Permissions defined in this policy** to make sure that you have granted the intended permissions.
12. (Optional) Add metadata to the policy by attaching tags as key-value pairs. For more information about using tags in IAM, see [Tagging IAM resources \(p. 399\)](#).
13. 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 \(p. 598\)](#).

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 \(p. 494\)](#).

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. In the **Policy editor**, choose **Visual** and then on the right side of the page, choose **Actions** then choose **Import policy**.
5. In the **Import policy** window, choose the managed policies that most closely match the policy that you want to include in your new policy. You can use the search box at the top to limit the results in the list of policies.
6. Choose **Import policy**.

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 **Next**.

Note

You can switch between the **Visual** and **JSON** editor options anytime. However, if you make changes or choose **Next** in the **Visual** editor, IAM might restructure your policy to optimize it for the visual editor. For more information, see [Policy restructuring \(p. 1181\)](#).

8. On the **Review and create** page, type a **Policy Name** and a **Description** (optional) for the policy that you are creating. You cannot edit these settings later. Review the **Permissions defined in this policy** and then choose **Create policy** to save your work.

To import an existing managed policy in the JSON 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. In the **Policy editor** section, choose the **JSON** option, and then on the right side of the page, choose **Actions** then choose **Import policy**.
5. In the **Import policy** window, choose the managed policies that most closely match the policy that you want to include in your new policy. You can use the search box at the top to limit the results in the list of policies.
6. Choose **Import policy**.

Statements from the imported policies are added to the bottom of your JSON policy.

7. Customize your policy in JSON. Resolve any security warnings, errors, or general warnings generated during [policy validation \(p. 588\)](#), and then choose **Next**. Or, customize your policy in the **Visual** editor. Then choose **Next**.

Note

You can switch between the **Visual** and **JSON** editor options anytime. However, if you make changes or choose **Next** in the **Visual** editor, IAM might restructure your policy to optimize it for the visual editor. For more information, see [Policy restructuring \(p. 1181\)](#).

8. On the **Review and create** page, type a **Policy Name** and a **Description** (optional) for the policy that you are creating. You cannot edit these later. Review the policy **Permissions defined in this policy** 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 \(p. 598\)](#).

Creating IAM policies (AWS CLI)

A [policy \(p. 485\)](#) 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. As a [best practice \(p. 1032\)](#), we recommend that you use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions. By [validating your policies \(p. 588\)](#) you can address any errors or recommendations before you 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 AWS STS quotas \(p. 1220\)](#).

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 \(p. 185\)](#).

To validate a customer managed policy (AWS CLI)

Use the following IAM Access Analyzer command:

- [validate-policy](#)

Creating IAM policies (AWS API)

A [policy \(p. 485\)](#) 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. As a [best practice \(p. 1032\)](#), we recommend that you use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions. By [validating your policies \(p. 588\)](#) you can address any errors or recommendations before you 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 AWS STS quotas \(p. 1220\)](#).

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 \(p. 185\)](#).

To validate a customer managed policy (AWS API)

Call the following IAM Access Analyzer operation:

- [ValidatePolicy](#)

Validating IAM policies

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, AWS automatically examines them to ensure that they comply with the IAM policy grammar. If AWS determines that a policy is not in compliance with the grammar, it prompts you to fix the policy.

IAM Access Analyzer provides additional policy checks with recommendations to help you further refine the policy. To learn more about IAM Access Analyzer policy checks and actionable recommendations, see [IAM Access Analyzer policy validation](#). To view a list of warnings, errors, and suggestions that are returned by IAM Access Analyzer, see [IAM Access Analyzer policy check reference](#).

Validation scope

AWS checks JSON policy syntax and grammar. It also verifies that your ARNs are formatted properly and action names and condition keys are correct.

Accessing policy validation

Policies are validated automatically when you create a JSON policy or edit an existing policy in the AWS Management Console. If the policy syntax is not valid, you receive a notification and must fix the problem before you can continue. The findings from the IAM Access Analyzer policy validation are automatically returned in the AWS Management Console if you have permissions for `access-analyzer:ValidatePolicy`. You can also validate policies using the AWS API or AWS CLI.

Existing policies

You might have existing policies that are not valid because they were created or last saved before the latest updates to the policy engine. As a [best practice \(p. 1032\)](#), we recommend that you use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions. We recommend that you open your existing policies and review the policy validation results that are generated. You cannot edit and save existing policies without fixing any policy syntax errors.

Generate policies based on access activity

As an administrator or developer, you might grant permissions to IAM entities (users or roles) beyond what they require. IAM provides several options to help you refine the permissions that you grant. One option is to generate an IAM policy that is based on access activity for an entity. IAM Access Analyzer reviews your AWS CloudTrail logs and generates a policy template that contains the permissions that the entity used in your specified date range. You can use the template to create a policy with fine-grained permissions that grant only the permissions that are required to support your specific use case.

For example, imagine that you are a developer and your engineering team has been working on a project to create a new application. To encourage experimentation and enable your team to move fast, you've configured a role with broad permissions while the application is in development. Now the application is ready for production. Before the application can launch in the production account, you want to identify only the permissions that the role needs for the application to function. This helps you to adhere to the best practice of [granting least privilege \(p. 1035\)](#). You can generate a policy based on the access activity of the role that you have been using for the application in the development account. You can further refine the generated policy and then attach the policy to an entity in your production account.

To learn more about IAM Access Analyzer policy generation, see [IAM Access Analyzer policy generation](#).

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 \(p. 485\)](#).

You can access the IAM Policy Simulator Console at: <https://pollicysim.aws.amazon.com/>

Important

The policy simulator results can differ from your live AWS environment. We recommend that you check your policies against your live AWS environment after testing using the policy simulator to confirm that you have the desired results. For more information, see [How the IAM policy simulator works \(p. 590\)](#).

[Getting Started with the IAM Policy Simulator](#)

With the IAM policy simulator, you can test and troubleshoot identity-based policies and IAM permissions boundaries. Here are some common things you can do with the policy simulator:

- Test identity-based policies that are attached to IAM users, user groups, or roles in your AWS account. If more than one policy is attached to the user, 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 \(p. 501\)](#) on IAM entities. You can only simulate one permissions boundary at a time.
- Test the effects of resource-based policies on IAM users that are attached to AWS resources, such as Amazon S3 buckets, Amazon SQS queues, Amazon SNS topics, or Amazon S3 Glacier vaults. To use a resource-based policy in the policy simulator for IAM users, you must include the resource in the simulation. You must also select the check box to include that resource's policy in the simulation.

Note

Simulation of resource-based policies isn't supported for IAM roles.

- 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 identity-based policies.

Note

The policy simulator doesn't evaluate SCPs that have global conditions.

- Test new identity-based policies that are not yet attached to a user, user group, or role by typing or copying them into the policy simulator. These are used only in the simulation and are not saved. You can't type or copy a resource-based policy in the policy simulator.
- Test identity-based 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.

Note

The policy simulator doesn't simulate tags provided as input if the identity-based policy in the simulation doesn't have a Condition element that explicitly checks for tags.

- Identify which specific statement in identity-based policy results in allowing or denying access to a particular resource or action.

Topics

- [How the IAM policy simulator works \(p. 590\)](#)
- [Permissions required for using the IAM policy simulator \(p. 590\)](#)
- [Using the IAM policy simulator \(console\) \(p. 593\)](#)
- [Using the IAM policy simulator \(AWS CLI and AWS API\) \(p. 597\)](#)

How the IAM policy simulator works

The policy simulator evaluates statements in the identity-based policy and the inputs that you provide during simulation. The policy simulator results can differ from your live AWS environment. We recommend that you check your policies against your live AWS environment after testing using the policy simulator to confirm that you have the desired results.

The policy simulator differs from the live AWS environment in the following ways:

- The policy 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. The policy simulator doesn't consider real context key values in production.
- Because the policy 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 in the policy simulator, these changes affect only the policy simulator. The corresponding policy in your AWS account remains unchanged.
- You can't test service control policies (SCPs) with [global condition keys \(p. 1338\)](#).
- The policy simulator doesn't support simulation for IAM roles and users for cross-account access.

Note

The IAM policy simulator doesn't determine which services support [global condition keys \(p. 1338\)](#) for authorization. For example, the policy simulator doesn't identify that a service doesn't support [aws:TagKeys \(p. 1364\)](#).

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, user group, or role by typing or copying those policies into the policy 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, user 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 \(IAM\)" \(p. 531\)](#).

Permissions required for using the policy simulator console

You can allow users to test policies that are attached to IAM users, user 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, user group, or role, see [IAM: Access the policy simulator console \(p. 557\)](#).

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 \(p. 567\)](#).

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 user 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, 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, user 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 \(p. 556\)](#).

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, user 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::777788889999: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 \(p. 567\)](#).

Using the IAM policy simulator (console)

By default, users can test policies that are not yet attached to a user, 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, 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 policy simulator, and use the policy simulator as described in the following steps.

After you have permission to use the IAM Policy Simulator Console, you can use the policy simulator to test an IAM user, user group, role, or resource policy.

To test a policy that is attached to a user, 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 policy 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 user group	Choose Groups in the Users, Groups, and Roles list. Then choose the user 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, 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 user group, you can launch the IAM policy simulator directly from the [IAM console](#): In the navigation pane, choose **User groups**. Choose the name of the group that you want to test a policy on, and then choose the **Permissions** tab. Choose **Simulate**.

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 \(p. 501\)](#) for an IAM entity (user or role), but not for user 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, 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 [AWSglobal condition keys \(p. 1338\)](#), 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 \(ARN\) \(p. 1213\)](#) 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 \(ARN\) \(p. 1213\)](#) 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 policy 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-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. 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 policy simulator allows you to make changes to policies for testing purposes only.</p>
Cannot get the resource policy. Reason: <i>detailed error message</i>	<p>The policy simulator is not able to access a requested resource-based policy. Ensure that the specified resource ARN is correct and that the user running the simulation has permission to read the resource's policy.</p>
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	<p>No action required.</p> <p>This message is informational. In the current version, the policy simulator evaluates policies</p>

Message	Steps to resolve
simulator does not support these mechanisms, so the results can differ from your production environment.	attached to users and user 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.
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 policy validation to review recommended updates to your policies.	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, review the policy validation options at Validating IAM policies (p. 588) to identify and fix these policies.
This policy must be updated to comply with the latest policy syntax rules.	This message is displayed if you have policies that do not comply with the IAM policy grammar. In order to simulate these policies, review the policy validation options at Validating IAM policies (p. 588) to identify and fix these policies.

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, 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 \(IAM\) \(p. 531\)](#).

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 policy 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, 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 \(p. 501\)](#) for only entities (users or roles) that are using the same methods. Permissions boundaries are an advanced AWS feature that control the maximum permissions that an entity can have.

Topics

- [Terminology \(p. 598\)](#)
- [View identity activity \(p. 599\)](#)
- [Adding IAM identity permissions \(console\) \(p. 599\)](#)
- [Removing IAM identity permissions \(console\) \(p. 601\)](#)
- [Adding IAM policies \(AWS CLI\) \(p. 602\)](#)
- [Removing IAM policies \(AWS CLI\) \(p. 603\)](#)
- [Adding IAM policies \(AWS API\) \(p. 604\)](#)
- [Removing IAM policies \(AWS API\) \(p. 605\)](#)

Terminology

When you associate permissions policies with identities (users, user 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, 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, 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, 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 \(p. 185\)](#) 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, 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 \(p. 185\)](#) 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 \(p. 494\)](#).
- For more information about permissions boundaries, see [Permissions boundaries for IAM entities \(p. 501\)](#).
- For general information about IAM policies, see [Policies and permissions in IAM \(p. 485\)](#).
- For information about validating IAM policies, see [Validating IAM policies \(p. 588\)](#).
- The number and size of IAM resources in an AWS account are limited. For more information, see [IAM and AWS STS quotas \(p. 1220\)](#).

View identity activity

Before you change the permissions for an identity (user, 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 \(p. 616\)](#).

Adding IAM identity permissions (console)

You can use the AWS Management Console to add permissions to an identity (user, user group, or role). To do this, attach managed policies that control permissions, or specify a policy that serves as a [permissions boundary \(p. 501\)](#). 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 radio button next to the name of the policy to attach. You can use the search box to filter the list of policies.
4. Choose **Actions**, and then choose **Attach**.
5. Select one or more identities to attach the policy to. You can use 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 search box to filter the list of policies.
4. On the policy details page, choose the **Entities attached** tab, and then, if necessary, open the **Attached as a permissions boundaries** section and choose **Set this policy as a permissions boundary**.
5. Select one or more users or roles on which to use the policy for a permissions boundary. You can use the search box to filter the list of principal entities. After selecting the principals, choose **Set permissions boundary**.

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. Choose **Add permissions** and then choose **Create inline policy**.

Note

You cannot embed an inline policy in a [service-linked role \(p. 185\)](#) 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 \(p. 1224\)](#) 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 \(p. 585\)](#) – 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 \(p. 584\)](#) – 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 using the JSON editor \(p. 583\)](#) – In the **JSON** editor option, you can use JSON syntax to create a policy. You can type a new JSON policy document or paste an [example policy \(p. 529\)](#).
7. After you create an inline policy, it is automatically embedded in your user or role.

To embed an inline policy for a user 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 **User groups**.
3. In the list, choose the name of the user group to embed a policy in.
4. Choose the **Permissions** tab, choose **Add permissions**, and then choose **Create inline policy**.
5. Do one of the following:

- Choose the **Visual** option to create the policy. For more information, see [Creating policies with the visual editor \(p. 584\)](#).
 - Choose the **JSON** option to create the policy. For more information, see [Creating policies using the JSON editor \(p. 583\)](#).
6. When you are satisfied with the policy, choose **Create 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 search box to filter the list of policies.
4. On the policy details page, choose the **Entities attached** tab, and then, if necessary, open the **Attached as a permissions boundary** section. Select the check box next to the users or roles whose boundaries you want to change and then choose **Change**.
5. Select a new policy to use for a permissions boundary. You can use the search box to filter the list of policies. After selecting the policy, choose **Set permissions boundary**.

Removing IAM identity permissions (console)

You can use the AWS Management Console to remove permissions from an identity (user, user group, or role). To do this, detach managed policies that control permissions, or remove a policy that serves as a [permissions boundary \(p. 501\)](#). 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 radio button next to the name of the policy to detach. You can use the search box to filter the list of policies.
4. Choose **Actions**, and then choose **Detach**.
5. Select the identities to detach the policy from. You can use 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 search box to filter the list of policies.
4. On the policy summary page, choose the **Entities attached** tab, and then, if necessary, open the **Attached as a permissions boundary** section and choose the entities to remove the permissions boundary from. Then choose **Remove boundary**.
5. Confirm that you want to remove the boundary and choose **Remove boundary**.

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 **User groups, Users, or Roles**.
3. In the list, choose the name of the user group, user, or role that has the policy you want to remove.
4. Choose the **Permissions** tab.
5. Select the check box next to the policy and choose **Remove**.
6. Choose **Remove** in the confirmation box.

Adding IAM policies (AWS CLI)

You can use the AWS CLI to add permissions to an identity (user, user group, or role). To do this, attach managed policies that control permissions, or specify a policy that serves as a [permissions boundary \(p. 501\)](#). 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, 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, user group, or role that is not a [service-linked role \(p. 185\)](#)), 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 \(p. 501\)](#). 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, user 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, 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, 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, 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, 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, user group, or role that is not a [service-linked role \(p. 185\)](#)), 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 \(p. 501\)](#). 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, 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, user group, or role that is not a [service-linked role \(p. 185\)](#)), 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 \(p. 501\)](#). 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, user groups, and roles) to which a managed policy is attached:
 - [ListEntitiesForPolicy](#)
 - To list the managed policies attached to an identity (a user, user group, or role), call one of the following operations:
 - [ListAttachedUserPolicies](#)
 - [ListAttachedGroupPolicies](#)
 - [ListAttachedRolePolicies](#)
3. To detach a managed policy from an identity (user, 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)

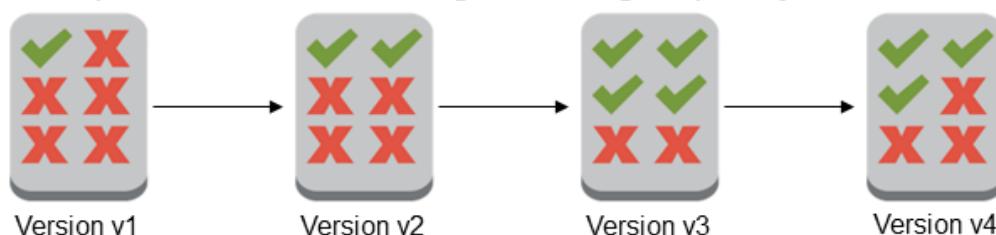
1. (Optional) To list all inline policies that are attached to an identity (user, user group, role), call one of the following operations:
 - [ListUserPolicies](#)
 - [ListGroupPolicies](#)
 - [ListRolePolicies](#)
2. (Optional) To retrieve an inline policy document that is embedded in an identity (user, user group, or role), call one of the following operations:
 - [GetUserPolicy](#)
 - [GetGroupPolicy](#)
 - [GetRolePolicy](#)
3. To delete an inline policy from an identity (user, user group, or role that is not a [service-linked role](#) (p. 185)), 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 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 \(p. 1261\)](#).

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 \(p. 607\)](#)
- [Setting the default version of customer managed policies \(p. 607\)](#)
- [Using versions to roll back changes \(p. 609\)](#)
- [Version limits \(p. 609\)](#)

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

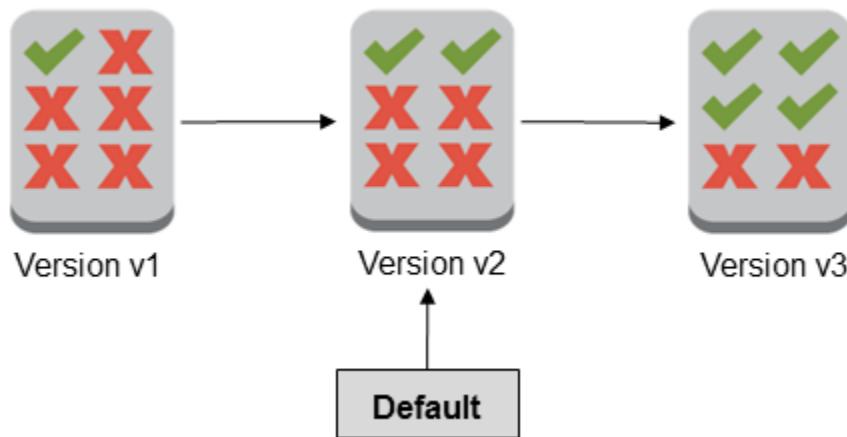
One of the versions of a managed policy is set as the *default* version. The policy's default version is the operative version—that is, it's the version that is in effect for all of the principal entities (users, user groups, and roles) that the managed policy is attached to.

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, 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 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 \(AWS CLI\) \(p. 611\)](#).

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" \(p. 609\)](#).

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 \(p. 485\)](#) 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 user group, user, or role. Identity-based policies include AWS managed policies, customer managed policies, and [inline policies \(p. 494\)](#). 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 AWS STS quotas \(p. 1220\)](#).

Topics

- [View policy access \(p. 609\)](#)
- [Editing customer managed policies \(console\) \(p. 610\)](#)
- [Editing inline policies \(console\) \(p. 611\)](#)
- [Editing customer managed policies \(AWS CLI\) \(p. 611\)](#)
- [Editing customer managed policies \(AWS API\) \(p. 612\)](#)

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 \(p. 616\)](#).

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 \(p. 606\)](#).

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 search box to filter the list of policies.
4. Choose the **Permissions** tab, and then choose **Edit**.
5. Do one of the following:
 - Choose the **Visual** option 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 **Next** to continue.
 - Choose the **JSON** option 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. Resolve any security warnings, errors, or general warnings generated during [policy validation \(p. 588\)](#), and then choose **Next**.

Note

You can switch between the **Visual** and **JSON** editor options any time. However, if you make changes or choose **Next** in the **Visual** editor, IAM might restructure your policy to optimize it for the visual editor. For more information, see [Policy restructuring \(p. 1181\)](#).

6. On the **Review and save** page, review **Permissions defined in this policy** and then choose **Save changes** to save your work.
7. If the managed policy already has the maximum of five versions, choosing **Save changes** displays a dialog box. To save your new version, the oldest non-default version of the policy is removed and replaced with this new version. Optionally, you can set the new version as the default policy version.

Choose **Save changes** 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 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 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, user group, or role (console)

1. In the navigation pane, choose **Users**, **User groups**, or **Roles**.
2. Choose the name of the user, user group, 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** option 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 **Next** to continue.
 - Choose the **JSON** option 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. Resolve any security warnings, errors, or general warnings generated during [policy validation \(p. 588\)](#), and then choose **Next**. 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** and **JSON** editor options any time. However, if you make changes or choose **Next** in the **Visual** editor, IAM might restructure your policy to optimize it for the visual editor. For more information, see [Policy restructuring \(p. 1181\)](#).

5. On the **Review** page, review the policy summary and then choose **Save changes** to save your work.

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, user 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, 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](#)
4. (Optional) To validate a customer managed policy, run the following IAM Access Analyzer command:
 - [validate-policy](#)

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, user groups, and roles) to which a managed policy is attached:
 - [ListEntitiesForPolicy](#)
 - To list the managed policies attached to an identity (a user, user group, or role):

- [ListAttachedUserPolicies](#)
 - [ListAttachedGroupPolicies](#)
 - [ListAttachedRolePolicies](#)
3. To edit a customer managed policy, call the following operation:
 - [CreatePolicyVersion](#)
 4. (Optional) To validate a customer managed policy, call the following IAM Access Analyzer operation:
 - [ValidatePolicy](#)

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.

Note

Deletion of IAM policies is permanent. After the policy is deleted it cannot be recovered.

For more information about the difference between managed and inline policies, see [Managed policies and inline policies \(p. 494\)](#).

For general information about IAM policies, see [Policies and permissions in IAM \(p. 485\)](#).

The number and size of IAM resources in an AWS account are limited. For more information, see [IAM and AWS STS quotas \(p. 1220\)](#).

Topics

- [View policy access \(p. 613\)](#)
- [Deleting IAM policies \(console\) \(p. 614\)](#)
- [Deleting IAM policies \(AWS CLI\) \(p. 614\)](#)
- [Deleting IAM policies \(AWS API\) \(p. 615\)](#)

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 \(p. 616\)](#).

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 radio button next to the customer managed policy to delete. You can use the search box to filter the list of policies.
4. Choose **Actions**, and then choose **Delete**.
5. Follow the instructions to confirm that you want to delete the policy, and then choose **Delete**.

To delete an inline policy for a user group, user, or role (console)

1. In the navigation pane, choose **User groups**, **Users**, or **Roles**.
2. Choose the name of the user group, user, or role with the policy that you want to delete. Then choose the **Permissions** tab.
3. Select the check boxes next to the policies to delete and choose **Remove**. To delete an inline policy in **Users** or **Roles**, choose **Remove** to confirm the deletion. If you are deleting a single inline policy in **User groups**, type the name of the policy and choose **Delete**. If you are deleting multiple inline policies in **User groups**, type the number of policies you are deleting followed by **inline policies** and choose **Delete**. For example, if you are deleting three inline policies, type **3 inline policies**.

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, user 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, 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, 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, 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, user group, or role that is not a [service-linked role](#) (p. 185)), 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, user 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, 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, 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, user group, or role), call one of the following operations:
 - [GetUserPolicy](#)
 - [GetGroupPolicy](#)
 - [GetRolePolicy](#)
3. To delete an inline policy from an identity (user, user group, or role that is not a [service-linked role \(p. 185\)](#)), 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. \(p. 1035\)](#) You can view last accessed information for entities or policies that exist in IAM or AWS Organizations.

Topics

- [Last accessed information types for IAM \(p. 616\)](#)
- [Last accessed information for AWS Organizations \(p. 617\)](#)
- [Things to know about last accessed information \(p. 617\)](#)
- [Permissions required \(p. 618\)](#)
- [Troubleshooting activity for IAM and Organizations entities \(p. 620\)](#)
- [Where AWS tracks last accessed information \(p. 621\)](#)
- [Viewing last accessed information for IAM \(p. 622\)](#)
- [Viewing last accessed information for Organizations \(p. 625\)](#)
- [Example scenarios for using last accessed information \(p. 629\)](#)

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 EC2, IAM, Lambda, and 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 \(p. 622\)](#).

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 \(p. 629\)](#).

To learn more about how the information for management actions is provided, see [Things to know about last accessed information \(p. 617\)](#).

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 \(p. 625\)](#).

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 \(p. 629\)](#).

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 Amazon S3 actions information began on April, 12, 2020. The tracking period for Amazon EC2, IAM, and Lambda actions began on April 7, 2021. For more information, see [Where AWS tracks last accessed information \(p. 621\)](#).
- **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 action last accessed information.
- **Action last accessed information** – Action last accessed information is available for Amazon EC2, IAM, Lambda, and Amazon S3 management actions accessed by IAM entities. IAM provides action information for Amazon EC2, IAM, Lambda, and 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](#).

Note

Action last accessed information is not available for Amazon S3 data events.

- **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 \(p. 1264\)](#).
- **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 \(p. 242\)](#). To learn how the different policy types are evaluated to allow or deny access, see [Policy evaluation logic \(p. 1306\)](#).
- **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 \(p. 1306\)](#).
- **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 Amazon EC2, IAM, Lambda, or Amazon 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 an identity-based policy that allows viewing IAM last accessed information. Additionally, it allows read-only access to all of IAM. This policy defines permissions for programmatic and console access.

```
{  
    "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 an identity-based policy that allows viewing service last accessed information for Organizations. Additionally, it allows read-only access to all of Organizations. This policy defines permissions for programmatic and console access.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "iam:GenerateOrganizationsAccessReport",  
                "iam:GetOrganizationsAccessReport",  
                "organizations:Describe*",  
                "organizations>List*"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

You can also use the [iam:OrganizationsPolicyId \(p. 1371\)](#) 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 \(p. 570\)](#).

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 \(p. 485\)](#) or [Evaluating policies within a single account \(p. 1307\)](#).
- 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 EC2, IAM, Lambda, and 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. The tracking period for Amazon EC2, IAM, and Lambda management actions began on April 7, 2021. If the Region started supporting an action at a later date, then that date is also the action's 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 (Jakarta)	ap-southeast-3	December 13, 2021
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
Middle East (UAE)	me-central-1	August 30, 2022
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 EC2, IAM, Lambda, and Amazon S3. For more information about last accessed information, see [Refining permissions in AWS using last accessed information \(p. 616\)](#).

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.
- **User group** – View information about the last time that a user 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" \(p. 617\)](#).

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 **User groups**, **Users**, **Roles**, or **Policies**.
3. Choose any user, 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:
 - **User group** – View the list of services that user group members can access. You can also view when a member last accessed the service, what user group policies they used, and which user group member made the request. Choose the name of the policy to learn whether it is a managed policy or an inline user group policy. Choose the name of the user group member to see all of the members of the user 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 user 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 EC2, AWS Identity and Access Management, AWS Lambda, or Amazon S3** to view a list of 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 EC2, IAM, Lambda, 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 Amazon S3 actions started on April 12, 2020. The tracking period for Amazon EC2, IAM, and Lambda actions started on April 7, 2021. To learn more about the tracking start dates for each AWS Region, see [Where AWS tracks last accessed information \(p. 621\)](#).

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, Amazon EC2, IAM, and Lambda actions. An IAM resource can be a user, 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, 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.
- **User group** – Returns a list of services that members of the specified user group can access using the policies attached to the user group. For each service, the operation returns the date and time

of the last attempt made by any user group member. It also returns the ARN of that user and the total number of user 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 user 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, 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 \(p. 485\)](#) or [Evaluating policies within a single account \(p. 1307\)](#).
- [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, Amazon EC2, IAM, and Lambda actions. An IAM resource can be a user, 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, 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.
- **User group** – Returns a list of services that members of the specified user group can access using the policies attached to the user group. For each service, the operation returns the date and time of the last attempt made by any user group member. It also returns the ARN of that user and the total number of user 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 user 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, 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 \(p. 485\)](#) or [Evaluating policies within a single account \(p. 1307\)](#).
- [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 \(p. 616\)](#).

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.

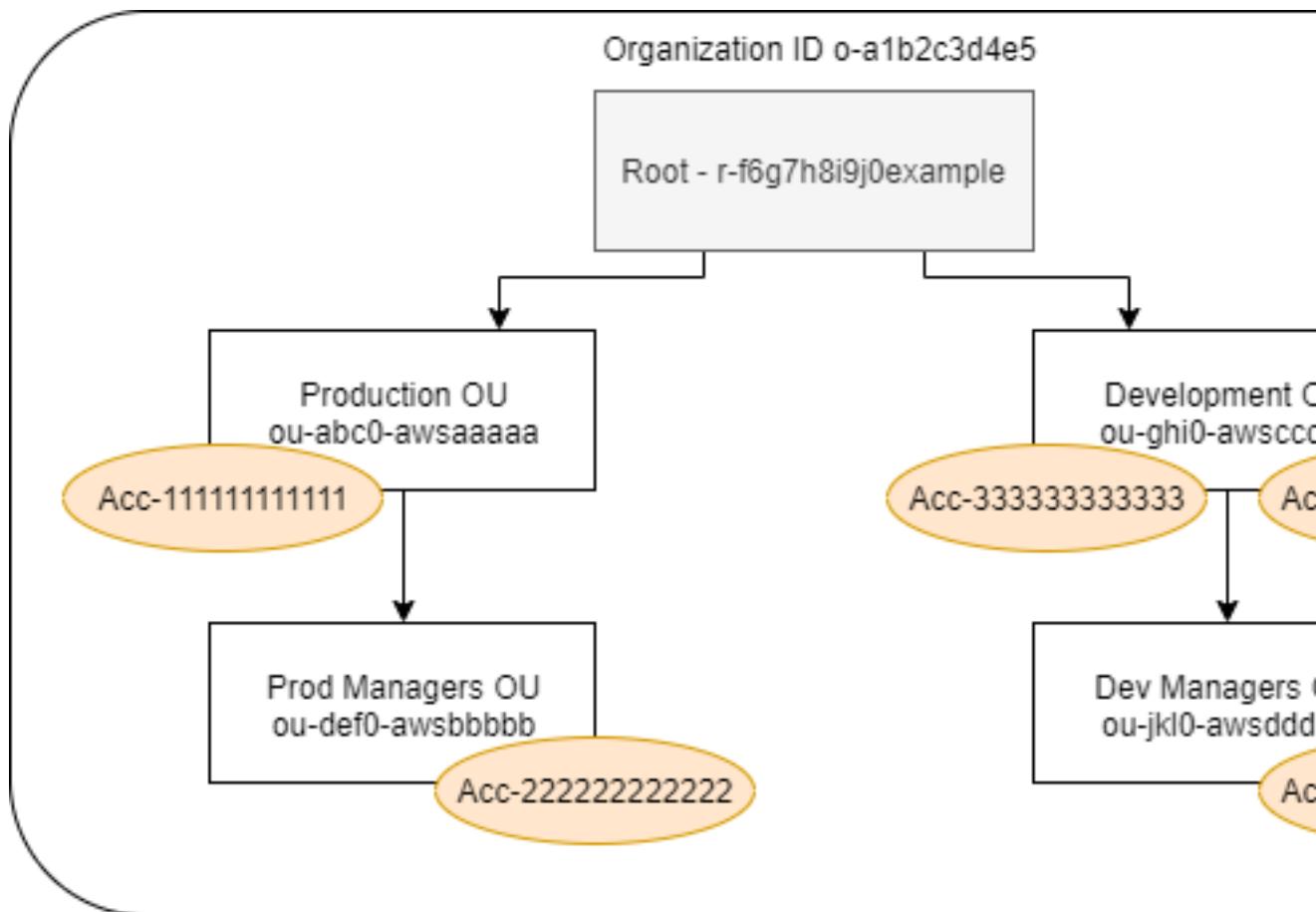
When you generate a programmatic report for a policy, you must specify an Organizations entity. This report includes a list of services that are allowed by the specified SCP. For each service, it includes the most recent account activity in the entity or entity's children that are granted permission 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" \(p. 617\)](#).

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-awsaaaaaa/1111111111111111/`

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 \(p. 622\)](#).
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 \(p. 622\)](#).
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 \(p. 622\)](#).
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 \(p. 617\)](#).
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 organizations-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 \(p. 617\)](#).
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 OrganizationsPolicyId 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 \(p. 616\)](#).

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" \(p. 617\)](#).

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 \(p. 1035\)](#) 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 \(p. 622\)](#) 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 \(p. 86\)](#)
- **Groups** – [Deleting a group \(p. 182\)](#)
- **Roles** – [Deleting a role \(p. 396\)](#)
- **Policies** – [Deleting a managed policy \(this also detaches the policy from identities\) \(p. 613\)](#)

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 \(p. 1035\)](#), 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 \(p. 609\)](#)
- **Policies** – [Converting an inline policy to a managed policy and then deleting it \(p. 500\)](#)
- **Policies** – [Adding an explicit deny to an existing policy \(p. 1316\)](#)
- **Policies** – [Detaching a managed policy from an identity \(user, group, or role\) \(p. 601\)](#)
- **Entities** – [Set a permissions boundary to control the maximum permissions that an entity \(user or role\) can have \(p. 598\)](#)
- **Groups** – [Removing users from a group \(p. 179\)](#)

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.

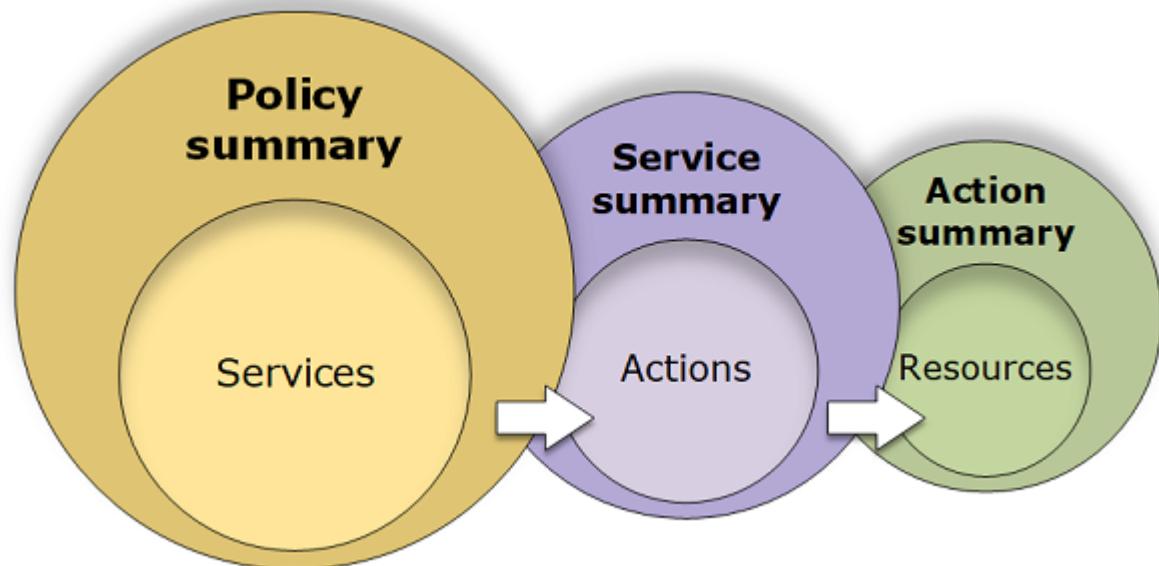
John signs into the IAM console using his management account credentials. In the IAM console, he locates the Organizations data for the Development OU. He reviews the **Service access report** table and sees two AWS services that have not been accessed in more than his preferred period of 180 days. He 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 \(p. 635\)](#), the [service summary \(p. 645\)](#), and the [action summary \(p. 649\)](#). 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.

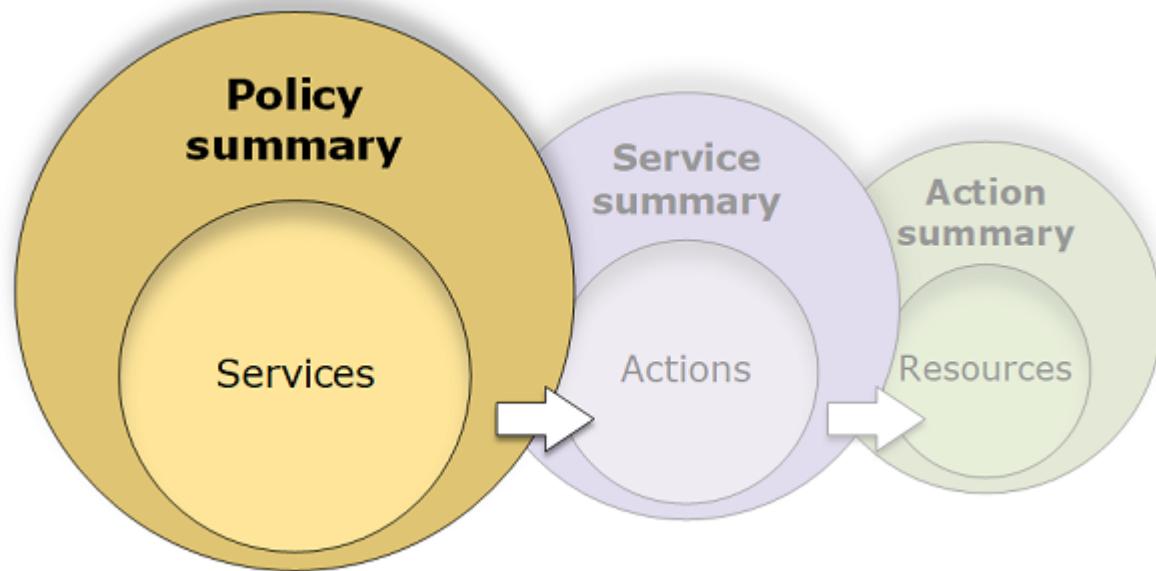
You can use the information in the policy summaries to understand the permissions that are allowed or denied by your policy. Policy summaries can help you [troubleshoot \(p. 1180\)](#) and fix policies that are not providing the permissions that you expect.

Topics

- [Policy summary \(list of services\) \(p. 635\)](#)
- [Service summary \(list of actions\) \(p. 645\)](#)
- [Action summary \(list of resources\) \(p. 649\)](#)
- [Examples of policy summaries \(p. 652\)](#)

Policy summary (list of services)

Policies are summarized in three tables: the policy summary, the [service summary \(p. 645\)](#), and the [action summary \(p. 649\)](#). 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 by choosing the policy name on the **Permissions** tab on the user details page. You can view the summaries for any policies that are attached to a role by choosing the policy name on the **Permissions** tab on the role details 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 \(p. 1184\)](#) 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 **Policy details** 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" \(p. 609\)](#). You cannot edit AWS managed policies.

To edit a policy for your policy summary using the Visual option

1. Open the policy summary as explained in the previous procedures.
2. Choose **Edit**.

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** option 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 \(p. 1181\)](#).
4. Edit your policy and choose **Next** 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 changes** to save your changes.

To edit a policy for your policy summary with the JSON option

1. Open the policy summary as explained in the previous procedures.
2. You can use the **Summary** and **JSON** buttons 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** and then choose the **JSON** option to edit the JSON policy document.

Note

You can switch between the **Visual** and **JSON** editor options any time. However, if you make changes or choose **Next** in the **Visual** editor option, IAM might restructure your policy to optimize it for the visual editor. For more information, see [Policy restructuring \(p. 1181\)](#).

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. Resolve any security warnings, errors, or general warnings generated during [policy validation \(p. 588\)](#), and then choose **Next**. If you still see a problem, choose **Previous** to return to the editing screen.
5. Choose **Save changes** to save your changes.

Understanding the elements of a policy summary

In the following example of a policy details page, 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.

Policy details

Type	Customer managed	Creation time
		September 13, 2023

1

Permissions

Entities attached

Tags

Policy versions

Access Advisor

2

 This policy defines some actions, resources, or conditions that do not provide permissions. To go

Permissions defined in this policy Info

Permissions defined in this policy document specify which actions are allowed or denied. To define permissions for an

 Search

4

Explicit deny (1 of 338 services)

5

Service

Access level

Resource

S3

Limited: List, Permissions management, Read, Write, Tagging

Multiple

Allow (3 of 338 services)

Service

Access level

Resource

Billing Console

Full: Read Limited: Write

All resources

CodeDeploy

Limited: List, Read, Write, Tagging

DeploymentGroupName like | All, region | string west-2

EC2

Limited: Read

All resources

In the preceding image, the policy summary is visible from within the **Policies** page:

1. The **Permissions** tab includes the permissions defined in the policy.
2. 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" \(p. 1186\)](#).
3. Use the **Summary** and **JSON** buttons to toggle between the policy summary and the JSON policy document.
4. Use the **Search** box to reduce the list of services and find a specific service.
5. The expanded view shows additional details of the **SummaryAllElements** policy.

The following policy summary table image shows the expanded **SummaryAllElements** policy on the policy details page.

Explicit deny (1 of 338 services) A		
Service B	Access level C	Resource D
S3	Limited: List, Permissions management, Read, Write, Tagging	Multiple
Allow (3 of 338 services)		
Service	Access level	Resource
Billing Console	Full: Read Limited: Write	All resources
CodeDeploy	Limited: List, Read, Write, Tagging	DeploymentGroupName like All, region string west-2
EC2	Limited: Read	All resources

In the preceding image, the policy summary is visible from within the **Policies** page:

- A. 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 a Deny statement for the Amazon S3 service and Allow statements for the Billing, CodeDeploy, and Amazon EC2 services.
- B. **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 \(list of actions\) \(p. 645\)](#). In this example, permissions are defined for the Amazon S3, Billing, CodeDeploy, and Amazon EC2 services.

C. Access level – This column tells whether the actions in each access level (List, Read, Write, Permission Management, and Tagging) 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 \(p. 643\)](#).

- **Full access** – This entry indicates that the service has access to all actions within all four of the access levels available for the service.
- 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 access level classifications (List, Read, Write, Permission Management, and Tagging):

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.

D. 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 DeploymentGroupName CodeDeploy resource. Depending on the information that the service provides to IAM, you might see an ARN or you might see the defined resource type.

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 \(p. 589\)](#).

E. 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. To view each of the multiple conditions for the policy, choose **JSON** to view the policy document.

F. Show remaining services – Toggle this button 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 \(p. 1306\)](#).

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 services** services on the **SummaryAllElements** policy details page with the possible warnings.

Explicit deny (1 of 338 services)		
Service	Access level	Resource
S3	Limited: List, Permissions management, Read, Write, Tagging	c Multiple! ⚠ One or more resources do not have an applicable resource.
Allow (3 of 338 services)		
Service	Access level	Resource
Billing Console	Full: Read Limited: Write	All resources
CodeCommit	None	d ⚠ No resources are defined.
CodeDeploy	Limited: List, Read, Write, Tagging	e DeploymentGroupName like All, region string west-2 ⚠ One or more resources do not have an applicable resource.
EC2	Limited: Read	All resources
S3	None	None! ⚠ One or more resources do not have an applicable resource.

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 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" \(p. 642\)](#). 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.  **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.
- e. **DeploymentGroupName | string like | All, region | string like | us-west-2 |**  **One or more actions do not have an applicable resource.** – The service has a defined action, and at least one more action that does not have a supporting resource.
- f. **None |**  **One or more conditions do not have an applicable action.** – The service has at least one 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": [
        "billing:Get*",
        "payments>List*",
        "payments:Update*",
        "account:Get*",
        "account>List*",
        "cur:GetUsage*"
      ],
      "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": [
    "codedploy:*",
    "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",
    "s3GetObject",
    "s3DeleteObject",
    "s3PutObject",
    "s3PutObjectAcl"
],
"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 \(p. 634\)](#). Access level summaries indicate whether the actions in each access level (List, Read, Tagging, 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 \(p. 652\)](#).

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. 1186) .

As [previously mentioned \(p. 640\)](#), **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 \(p. 1185\)](#).

Access level summaries that include limited (partial) access to actions are grouped using the AWS access level classifications **List**, **Read**, **Tagging**, **Write**, or **Permissions management**.

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.
- **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.
- **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.

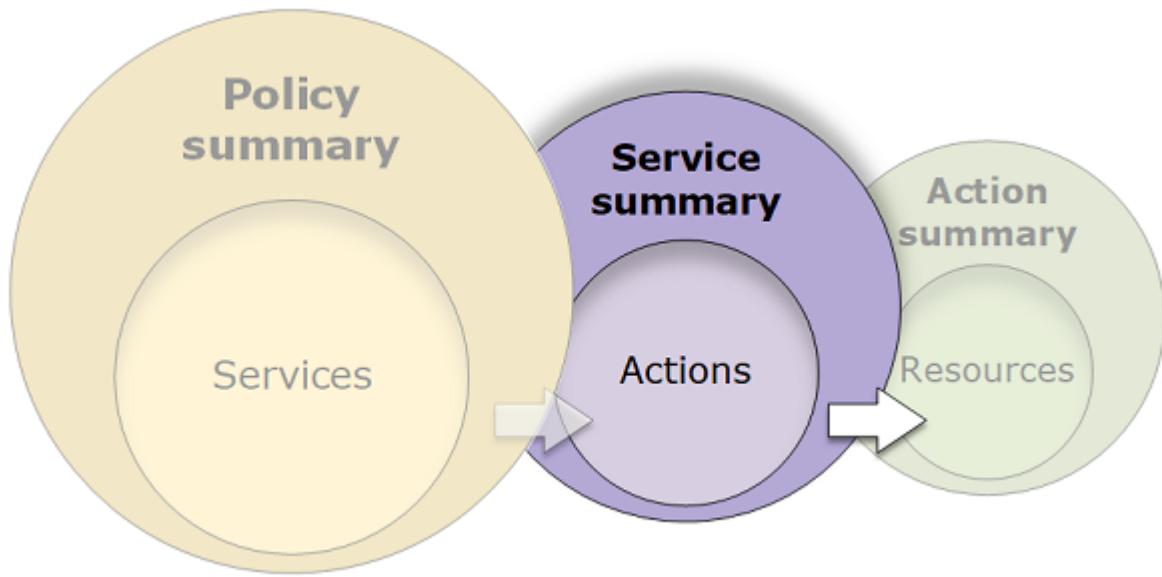
Tip

To improve the security of your AWS account, restrict or regularly monitor policies that include the **Permissions management** access level classification.

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 \(p. 635\)](#), the service summary, and the [action summary \(p. 649\)](#). 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.

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 **Policy details** 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, choose the name of the policy that you want to view.

If you are on the **Users** page and choose to view the service summary for a policy that is attached to that user, you are redirected to the **Policies** page. You can view service summaries only on the **Policies** page.

6. Choose **Summary**. 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, choose the name of the policy that you want to view.

If you are on the **Roles** page and choose to view the service summary for a policy that is attached to that user, you are redirected to the **Policies** page. You can view service summaries only on the **Policies** page.

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 a policy summary. The actions for this service are grouped by access level. For example, 35 **Read** actions are defined out of the total 52 **Read** actions available for the service.

Permissions Entities attached Tags Policy versions Access Advisor

i 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](#)

Permissions defined in this policy Info		Edit Summary 2 JSON	
Permissions defined in this policy document specify which actions are allowed or denied. To define permissions for an IAM identity (user, user group, or role), attach a policy to it.			
4 Search 3	5 Actions in S3 (82 of 128) 6 Show remaining 46 actions		
Action	Resource	Request condition	
DescribeJob (No access) 9	⚠ This action does not have an applicable resource.	None	
DescribeMultiRegionAccessPointOperation (No access)	⚠ This action does not have an applicable resource.	None	
GetAccelerateConfiguration 11	BucketName string like customer	None	
GetAccessPoint (No access)	⚠ This action does not have an applicable resource.	None	
GetAccessPointConfigurationForObjectLambda (No access)	⚠ This action does not have an applicable resource.	None	
GetAccessPointForObjectLambda (No access)	⚠ This action does not have an applicable resource.	None	
GetAccessPointPolicy (No access)	⚠ This action does not have an applicable resource.	None	
GetAccessPointPolicyForObjectLambda (No access)	⚠ This action does not have an applicable resource.	None	
GetAccessPointPolicyStatus (No access)	⚠ This action does not have an applicable resource.	None	
GetAccessPointPolicyStatusForObjectLambda (No access)	⚠ This action does not have an applicable resource.	None	
GetAccountPublicAccessBlock (No access)	⚠ This action does not have an applicable resource.	None	
GetAnalyticsConfiguration	BucketName string like customer	None	
GetBucketAcl	BucketName string like customer	None	

The service summary page for a managed policy includes the following information:

1. 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" \(p. 1186\)](#).
2. 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.)
3. To view the summary for a specific action, type keywords into the **Search** box to reduce the list of available actions.

4. Next to the **Services** back arrow appears the name of the service (in this case **S3**). The service summary for this service includes the list of allowed or denied actions that are defined in the policy. If the service appears under **(Explicit deny)** on the **Permissions** tab, then the actions listed in the service summary table are explicitly denied. If the service appears under **Allow** on the **Permissions** tab, then the actions listed in the service summary table are allowed.
5. **Action** – This column lists the actions that are defined within the policy and provides the resources and conditions for each action. If the policy grants or denies permissions to the action, then the action name links to the [action summary \(p. 649\)](#) table. The table groups these actions into at least one or up to five sections, depending on the level of access that the policy allows or denies. The sections are **List**, **Read**, **Write**, **Permission Management**, and **Tagging**. The count indicates the number of recognized actions that provide permissions within each access level. The total is the number of known actions for the service. In this example, 35 actions provide permissions out of 52 total known Amazon S3 **Read** actions. To view the access level classification that is assigned to each action in a service, see [Actions, Resources, and Condition Keys for AWS Services](#).
6. **Show remaining actions** – Toggle this button to expand or hide the table to include actions that are known but do not provide permissions for this service. Toggling the button also displays warnings for any elements that do not provide permissions.
7. **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 Amazon 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

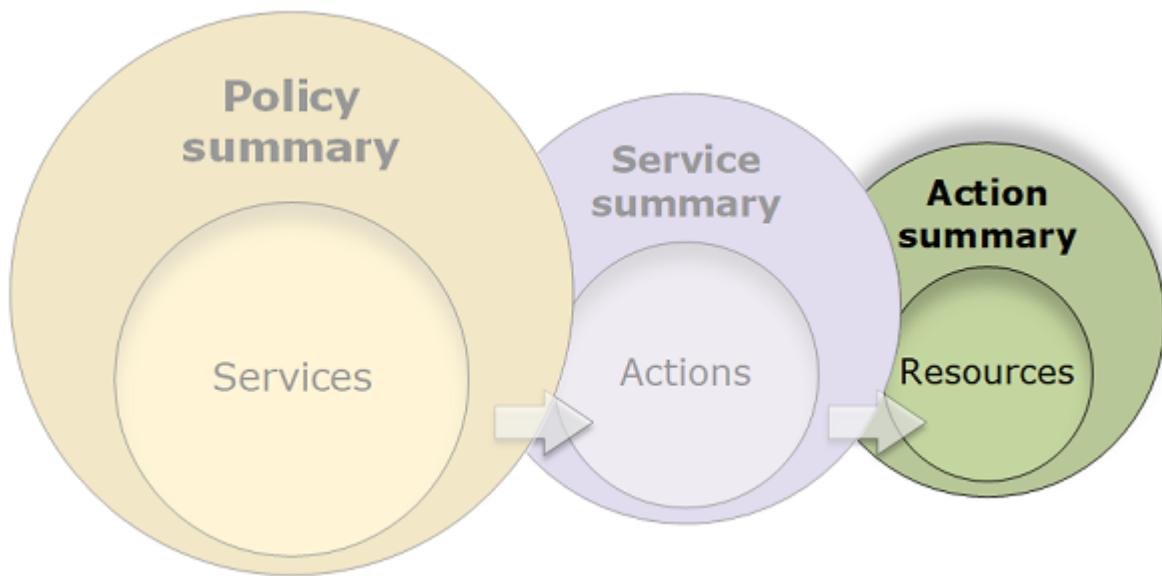
This column can include a resource from a different service. If the policy statement that includes the resource does not include both actions and resources from the same service, then your policy includes mismatched resources. IAM does not warn you about mismatched resources when you create a policy, 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 \(p. 589\)](#).

8. **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.
 9. **(No access)** – This policy includes an action that does not provide permissions.
- 10 **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.

11 Actions that provide permissions include a link to the action summary.

Action summary (list of resources)

Policies are summarized in three tables: the [policy summary \(p. 635\)](#), the [service summary \(p. 645\)](#), 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 managed policies, any policy that is attached to a user, and any policy that is attached to a role on 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 **Policy details** 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, choose the name of the policy that you want to view.

If you are on the **Users** page and choose to view the service summary for a policy that is attached to that user, you are redirected to the **Policies** page. You can view service summaries only on the **Policies** page.

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, choose the name of the policy that you want to view.

If you are on the **Roles** page and choose to view the service summary for a policy that is attached to that user, you are redirected to the **Policies** page. You can view service summaries only on the **Policies** page.

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 \(list of actions\) \(p. 645\)](#)). For this action, the policy defines multiple conditions on a single resource.

The screenshot shows the 'Permissions defined in this policy' section of the IAM console. At the top, there's an 'Info' link, a 'Search' bar with a magnifying glass icon (labeled 2), and tabs for 'Edit', 'Summary' (which has a red notification badge with the number 1), and 'JSON'. Below these are buttons for 'Actions' (labeled 3) and 'Resource' (labeled 4). The 'Actions' button is highlighted with a red circle. The 'Resource' button is also labeled 4. To the right of the 'Actions' button are buttons for 'Region' (labeled 5) and 'Account' (labeled 6), both with red circles. Further to the right is a 'Request condition' button (labeled 7) with a red circle. Below these buttons, the policy details are listed: 'BucketName | string like | customer, ObjectPath | string like | All', 'Region | All regions', 'Account | All accounts', and 'Request condition | s3:x-amz-acl = public-read'.

The action summary page includes the following information:

1. 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.)
2. To view the summary for a specific resource, type keywords into the **Search** box to reduce the list of available resources.

3. Next to the **Actions** back arrow appears the name of the service and action in the format `action name action in service` (in this case **PutObject action in S3**). The action summary for this service includes the list of resources that are defined in the policy.
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 in the current account.
 - **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 \(p. 635\)](#), the [service summaries \(p. 645\)](#), and the [action summaries \(p. 649\)](#) 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:

- i** This policy defines some actions, resources, or conditions that do not provide permissions. To grant details, choose **Show remaining**. [Learn more](#)

Permissions defined in this policy [Info](#)

Permissions defined in this policy document specify which actions are allowed or denied. To define permissions for an

[Edit](#)

Summary

[JSON](#)

 [Search](#)

Explicit deny (1 of 371 services)

Service	▲	Access level	▼	Resource
S3		Limited: List, Permissions management, Read, Write, Tagging		Multiple

Allow (1 of 371 services)

Service	▲	Access level	▼	Resource
S3		Full access		All resources

DenyCustomerBucket S3 (Explicit deny) Service Summary:

[< Services](#) Actions in S3 (82 of 130)

Read (35 of 53)

Action	Resource
GetAccelerateConfiguration	BucketName string like
GetAnalyticsConfiguration	BucketName string like
GetBucketAcl	BucketName string like
GetBucketCORS	BucketName string like
GetBucketLocation	BucketName string like
GetBucketLogging	BucketName string like
GetBucketNotification	BucketName string like
GetBucketObjectLockConfiguration	BucketName string like
GetBucketOwnershipControls	BucketName string like
GetBucketPolicy	BucketName string like
GetBucketPolicyStatus	BucketName string like
GetBucketPublicAccessBlock	BucketName string like
GetBucketRequestPayment	BucketName string like
GetBucketTagging	BucketName string like
GetBucketVersioning	BucketName string like
GetBucketWebsite	BucketName string like

GetObject (Read) Action Summary:

< Actions GetObject action in S3

Resource	Region	Account	Requ
BucketName string like customer, ObjectPath string like All	-	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:

Allow (1 of 370 services)

Service	▲	Access level	▼	Resource
DynamoDB		Limited: Read, Write		region string like us- TableName string like myDynamoTable

DynamoDbRowCognitoID DynamoDB (Allow) Service Summary:

< Services Actions in DynamoDB (4 of 65)

Read (1 of 26)

Action	Resource
GetItem	region string like us-west-1 string like myDynamoTable

Write (3 of 33)

Action	Resource
DeleteItem	region string like us-west-1 string like myDynamoTable
PutItem	region string like us-west-1 string like myDynamoTable
UpdateItem	region string like us-west-1 string like myDynamoTable

GetItem (List) Action Summary:

< Actions GetItem action in DynamoDB

Resource	Region	Account	Request ID
region string like us-west-1 , TableName string like myDynamoTable	us-west-1	123456789012	dynamotest

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:

Allow (1 of 370 services)

Service	▲ Access level	▼ Resource
S3	Limited: Permissions management, Write	Multiple

MultipleResourceCondition S3 (Allow) Service Summary:

< Services Actions in S3 (2 of 130)

Write (1 of 47)

Action	Resource
PutObject	Multiple

Permission Management (1 of 15)

Action	Resource
PutObjectAcl	Multiple

PutObject (Write) Action Summary:

< Actions PutObject action in S3

Resource	Region	Account	Requester
Multiple	-	All accounts	Multiple

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:

Allow (2 of 370 services)

Service	Access level	Resource
EC2	Limited: Read	All resources
S3	Limited: List	BucketName string like developer

EC2_Troubleshoot S3 (Allow) Service Summary:

[< Services](#) Actions in S3 (1 of 130)

List (1 of 7)

Action	Resource
ListBucket	BucketName string like

ListBucket (List) Action Summary:

[< Actions](#) ListBucket action in S3

Resource	Region	Account	Request ID
BucketName string like developer	-	All accounts	None

Policy 5: 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:

Allow (3 of 370 services)

Service	▲	Access level	▼	Resource
CodeBuild		Full: Permissions management Limited: List, Read, Write		region string like us-
CodeCommit		Full: Tagging Limited: List, Read, Write		ResourceSpecifier str MyDemoRepo, region like us-east-2
CodeDeploy		Full: Tagging Limited: List, Read, Write		Multiple

CodeBuild_CodeCommit_CodeDeploy CodeBuild (Allow) Service Summary:

< Services Actions in CodeBuild (24 of 53)

Read (4 of 9)

Action	Resource
BatchGetBuildBatches	region string like us-eas
BatchGetBuilds	region string like us-eas
BatchGetProjects	region string like us-eas
GetResourcePolicy	region string like us-eas

Write (16 of 28)

Action	Resource
BatchDeleteBuilds	region string like us-eas
CreateProject	region string like us-eas
CreateWebhook	region string like us-eas
DeleteBuildBatch	region string like us-eas
DeleteProject	region string like us-eas
DeleteWebhook	region string like us-eas
InvalidateProjectCache	region string like us-eas
RetryBuild	region string like us-eas
RetryBuildBatch	region string like us-eas
StartBuild	region string like us-eas
StartBuildBatch	region string like us-eas
StopBuild	region string like us-eas
StopBuildBatch	region string like us-eas
UpdateProject	region string like us-eas

CodeBuild_CodeCommit_CodeDeploy StartBuild (Write) Action Summary:

< Actions StartBuild action in CodeBuild

Resource	Region	Account	Request ID
region string like us-east-2	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" \(p. 485\)](#).

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 \(p. 1298\)](#).

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* with your AWS account ID):

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": "iam:*AccessKey*",
        "Resource": "arn:aws:iam::account-id: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 \(p. 665\)](#).

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 \(p. 188\)](#).

Note

Some services support resource-based policies as described in [Identity-based policies and resource-based policies \(p. 511\)](#) (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 \(p. 255\)](#).

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 \(p. 279\)](#).

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 \(p. 529\)](#).

Topics

- [Allow a user to list the account's groups, users, policies, and more for reporting purposes \(p. 665\)](#)
- [Allow a user to manage a group's membership \(p. 665\)](#)
- [Allow a user to manage IAM users \(p. 666\)](#)
- [Allow users to set account password policy \(p. 667\)](#)
- [Allow users to generate and retrieve IAM credential reports \(p. 667\)](#)
- [Allow all IAM actions \(admin access\) \(p. 667\)](#)

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 \(p. 565\)](#).

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 \(p. 562\)](#).

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 \(p. 1213\)](#).

```
{
    "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 the [password policy \(p. 94\)](#) of your AWS account. To view the example policy, see [IAM: Allows setting the account password requirements programmatically and in the console \(p. 566\)](#).

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 \(p. 164\)](#). To view the example policy, see [IAM: Generate and retrieve IAM credential reports \(p. 562\)](#).

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": "*"  
        }  
    ]  
}
```

Code examples for IAM using AWS SDKs

The following code examples show how to use IAM with an AWS software development kit (SDK).

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Code examples

- [Code examples for IAM using AWS SDKs \(p. 670\)](#)
 - [Actions for IAM using AWS SDKs \(p. 677\)](#)
 - [Add an IAM user to a group using an AWS SDK \(p. 679\)](#)
 - [Attach an IAM policy to a role using an AWS SDK \(p. 679\)](#)
 - [Attach an IAM policy to a user using an AWS SDK \(p. 690\)](#)
 - [Attach an inline policy to an IAM role using an AWS SDK \(p. 691\)](#)
 - [Create an IAM SAML provider using an AWS SDK \(p. 693\)](#)
 - [Create an IAM group using an AWS SDK \(p. 694\)](#)
 - [Create an IAM policy using an AWS SDK \(p. 695\)](#)
 - [Create an IAM policy version using an AWS SDK \(p. 705\)](#)
 - [Create an IAM role using an AWS SDK \(p. 706\)](#)
 - [Create an IAM service-linked role using an AWS SDK \(p. 715\)](#)
 - [Create an IAM user using an AWS SDK \(p. 719\)](#)
 - [Create an IAM access key using an AWS SDK \(p. 727\)](#)
 - [Create an alias for an IAM account using an AWS SDK \(p. 735\)](#)
 - [Create an inline IAM policy for a group using an AWS SDK \(p. 738\)](#)
 - [Create an inline IAM policy for a user using an AWS SDK \(p. 739\)](#)
 - [Delete an IAM SAML provider using an AWS SDK \(p. 741\)](#)
 - [Delete an IAM group using an AWS SDK \(p. 742\)](#)
 - [Delete an IAM group policy using an AWS SDK \(p. 743\)](#)
 - [Delete an IAM policy using an AWS SDK \(p. 744\)](#)
 - [Delete an IAM role using an AWS SDK \(p. 750\)](#)
 - [Delete an IAM role policy using an AWS SDK \(p. 755\)](#)
 - [Delete an IAM server certificate using an AWS SDK \(p. 756\)](#)
 - [Delete an IAM service-linked role using an AWS SDK \(p. 758\)](#)
 - [Delete an IAM user using an AWS SDK \(p. 760\)](#)
 - [Delete an IAM access key using an AWS SDK \(p. 768\)](#)
 - [Delete an IAM account alias using an AWS SDK \(p. 776\)](#)
 - [Delete an inline IAM policy from a user using an AWS SDK \(p. 779\)](#)
 - [Detach an IAM policy from a role using an AWS SDK \(p. 782\)](#)
 - [Detach an IAM policy from a user using an AWS SDK \(p. 789\)](#)
 - [Generate a credential report from IAM using an AWS SDK \(p. 791\)](#)

- [Get a credential report from IAM using an AWS SDK \(p. 791\)](#)
- [Get a detailed IAM authorization report for your account using an AWS SDK \(p. 792\)](#)
- [Get an IAM policy using an AWS SDK \(p. 793\)](#)
- [Get an IAM policy version using an AWS SDK \(p. 798\)](#)
- [Get an IAM role using an AWS SDK \(p. 799\)](#)
- [Get an IAM server certificate using an AWS SDK \(p. 803\)](#)
- [Get an IAM service-linked role's deletion status using an AWS SDK \(p. 804\)](#)
- [Get a summary of account usage from IAM using an AWS SDK \(p. 805\)](#)
- [Get an IAM user using an AWS SDK \(p. 806\)](#)
- [Get data about the last use of an IAM access key using an AWS SDK \(p. 808\)](#)
- [Get the IAM account password policy using an AWS SDK \(p. 811\)](#)
- [List SAML providers for IAM using an AWS SDK \(p. 814\)](#)
- [List a user's IAM access keys using an AWS SDK \(p. 817\)](#)
- [List IAM account aliases using an AWS SDK \(p. 824\)](#)
- [List IAM groups using an AWS SDK \(p. 828\)](#)
- [List inline policies for an IAM role using an AWS SDK \(p. 832\)](#)
- [List inline IAM policies for a user using an AWS SDK \(p. 837\)](#)
- [List IAM policies using an AWS SDK \(p. 837\)](#)
- [List policies attached to an IAM role using an AWS SDK \(p. 844\)](#)
- [List IAM roles using an AWS SDK \(p. 848\)](#)
- [List IAM server certificates using an AWS SDK \(p. 853\)](#)
- [List IAM users using an AWS SDK \(p. 855\)](#)
- [Remove an IAM user from a group using an AWS SDK \(p. 864\)](#)
- [Update an IAM server certificate using an AWS SDK \(p. 865\)](#)
- [Update an IAM user using an AWS SDK \(p. 866\)](#)
- [Update an IAM access key using an AWS SDK \(p. 870\)](#)
- [Upload an IAM server certificate using an AWS SDK \(p. 873\)](#)
- [Scenarios for IAM using AWS SDKs \(p. 874\)](#)
 - [Create an IAM group and add a user to the group using an AWS SDK \(p. 874\)](#)
 - [Create an IAM user and assume a role with AWS STS using an AWS SDK \(p. 891\)](#)
 - [Create read-only and read-write IAM users using an AWS SDK \(p. 973\)](#)
 - [Manage IAM access keys using an AWS SDK \(p. 980\)](#)
 - [Manage IAM policies using an AWS SDK \(p. 982\)](#)
 - [Manage IAM roles using an AWS SDK \(p. 986\)](#)
 - [Manage your IAM account using an AWS SDK \(p. 989\)](#)
 - [Roll back an IAM policy version using an AWS SDK \(p. 993\)](#)
 - [Work with the IAM Policy Builder API using an AWS SDK \(p. 994\)](#)
- [Code examples for AWS STS using AWS SDKs \(p. 997\)](#)
 - [Actions for AWS STS using AWS SDKs \(p. 998\)](#)
 - [Assume a role with AWS STS using an AWS SDK \(p. 998\)](#)
 - [Get a session token with AWS STS using an AWS SDK \(p. 1007\)](#)
- [Scenarios for AWS STS using AWS SDKs \(p. 1008\)](#)

 - [Assume an IAM role that requires an MFA token with AWS STS using an AWS SDK \(p. 1008\)](#)
 - [Construct a URL with AWS STS for federated users using an AWS SDK \(p. 1012\)](#)

- [Get a session token that requires an MFA token with AWS STS using an AWS SDK \(p. 1015\)](#)

Code examples for IAM using AWS SDKs

The following code examples show how to use IAM with an AWS software development kit (SDK).

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Get started

Hello IAM

The following code examples show how to get started using IAM.

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
namespace IAMActions;

public class HelloIAM
{
    static async Task Main(string[] args)
    {
        // Getting started with AWS Identity and Access Management (IAM). List
        // the policies for the account.
        var iamClient = new AmazonIdentityManagementServiceClient();

        var listPoliciesPaginator = iamClient.Paginator.ListPolicies(new
ListPoliciesRequest());
        var policies = new List<ManagedPolicy>();

        await foreach (var response in listPoliciesPaginator.Responses)
        {
            policies.AddRange(response.Policies);
        }

        Console.WriteLine("Here are the policies defined for your account:\n");
        policies.ForEach(policy =>
        {
            Console.WriteLine($"Created:
{policy.CreateDate}\t{policy.PolicyName}\t{policy.Description}");
        });
    }
}
```

- For API details, see [ListPolicies](#) in *AWS SDK for .NET API Reference*.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Code for the CMakeLists.txt CMake file.

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS iam)

# Set this project's name.
project("hello_iam")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed libraries
    # for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
        "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD)
    # Copy relevant AWS SDK for C++ libraries into the current binary directory for
    # running and debugging.

    # set(BIN_SUB_DIR "/Debug") # if you are building from the command line you may
    # need to uncomment this
    # and set the proper subdirectory to the executables' location.

    AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
        ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
    hello_iam.cpp)

target_link_libraries(${PROJECT_NAME}
    ${AWSSDK_LINK_LIBRARIES})
```

Code for the iam.cpp source file.

```
#include <aws/core/Aws.h>
#include <aws/iam/IAMClient.h>
```

```
#include <aws/iam/model/ListPoliciesRequest.h>
#include <iostream>
#include <iomanip>

/*
 * A "Hello IAM" starter application which initializes an AWS Identity and Access
Management (IAM) client
 * and lists the IAM policies.
 *
 * main function
 *
 * Usage: 'hello_iam'
 */
int main(int argc, char **argv) {
    Aws::SDKOptions options;
    // Optionally change the log level for debugging.
//    options.loggingOptions.logLevel = Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.
    int result = 0;
    {
        const Aws::String DATE_FORMAT("%Y-%m-%d");
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::IAM::IAMClient iamClient(clientConfig);
        Aws::IAM::Model::ListPoliciesRequest request;

        bool done = false;
        bool header = false;
        while (!done) {
            auto outcome = iamClient.ListPolicies(request);
            if (!outcome.IsSuccess()) {
                std::cerr << "Failed to list iam policies: " <<
                    outcome.GetError().GetMessage() << std::endl;
                result = 1;
                break;
            }

            if (!header) {
                std::cout << std::left << std::setw(55) << "Name" <<
                    std::setw(30) << "ID" << std::setw(80) << "Arn" <<
                    std::setw(64) << "Description" << std::setw(12) <<
                    "CreateDate" << std::endl;
                header = true;
            }

            const auto &policies = outcome.GetResult().GetPolicies();
            for (const auto &policy: policies) {
                std::cout << std::left << std::setw(55) <<
                    policy.GetPolicyName() << std::setw(30) <<
                    policy.GetPolicyId() << std::setw(80) << policy.GetArn()
<<
                    std::setw(64) << policy.GetDescription() << std::setw(12)
<<
                    policy.GetCreateDate().ToGmtString(DATE_FORMAT.c_str())
<<
                    std::endl;
            }

            if (outcome.GetResult().GetIsTruncated()) {
                request.SetMarker(outcome.GetResult().GetMarker());
            } else {
                done = true;
            }
        }
    }
}
```

```
        }
    }

Aws::ShutdownAPI(options); // Should only be called once.
return result;
}
```

- For API details, see [ListPolicies](#) in *AWS SDK for C++ API Reference*.

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
package main

import (
    "context"
    "fmt"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/iam"
)

// main uses the AWS SDK for Go (v2) to create an AWS Identity and Access
// Management (IAM)
// client and list up to 10 policies in your account.
// This example uses the default settings specified in your shared credentials
// and config files.
func main() {
    sdkConfig, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
        fmt.Println("Couldn't load default configuration. Have you set up your AWS
account?")
        fmt.Println(err)
        return
    }
    iamClient := iam.NewFromConfig(sdkConfig)
    const maxPols = 10
    fmt.Printf("Let's list up to %v policies for your account.\n", maxPols)
    result, err := iamClient.ListPolicies(context.TODO(), &iam.ListPoliciesInput{
        MaxItems: aws.Int32(maxPols),
    })
    if err != nil {
        fmt.Printf("Couldn't list policies for your account. Here's why: %v\n", err)
        return
    }
    if len(result.Policies) == 0 {
        fmt.Println("You don't have any policies!")
    } else {
        for _, policy := range result.Policies {
            fmt.Printf("\t%v\n", *policy.PolicyName)
        }
    }
}
```

- For API details, see [ListPolicies](#) in *AWS SDK for Go API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import { IAMClient, paginateListPolicies } from "@aws-sdk/client-iam";

const client = new IAMClient({});

export const listLocalPolicies = async () => {
    /**
     * In v3, the clients expose paginateOperationName APIs that are written using
     * async generators so that you can use async iterators in a for await..of loop.
     * https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/index.html#paginators
     */
    const paginator = paginateListPolicies(
        { client, pageSize: 10 },
        // List only customer managed policies.
        { Scope: "Local" }
    );

    console.log("IAM policies defined in your account:");
    let policyCount = 0;
    for await (const page of paginator) {
        if (page.Policies) {
            page.Policies.forEach((p) => {
                console.log(`[${p.PolicyName}]`);
                policyCount++;
            });
        }
    }
    console.log(`Found ${policyCount} policies.`);
};
```

- For API details, see [ListPolicies](#) in *AWS SDK for JavaScript API Reference*.

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

From src/bin/hello.rs.

```
use aws_sdk_iam::error::SdkError;
```

```
use aws_sdk_iam::operation::list_policies::ListPoliciesError;
use clap::Parser;

const PATH_PREFIX_HELP: &str = "The path prefix for filtering the results.";

#[derive(Debug, clap::Parser)]
#[command(about)]
struct HelloScenarioArgs {
    #[arg(long, default_value="/", help=PATH_PREFIX_HELP)]
    pub path_prefix: String,
}

#[tokio::main]
async fn main() -> Result<(), SdkError<ListPoliciesError>> {
    let sdk_config = aws_config::load_from_env().await;
    let client = aws_sdk_iam::Client::new(&sdk_config);

    let args = HelloScenarioArgs::parse();

    iam_service::list_policies(client, args.path_prefix).await?;

    Ok(())
}
```

From src/iam-service-lib.rs.

```
pub async fn list_policies(
    client: iamClient,
    path_prefix: String,
) -> Result<Vec<String>, SdkError<ListPoliciesError>> {
    let mut list_policies = client
        .list_policies()
        .path_prefix(path_prefix)
        .scope(PolicyScopeType::Local)
        .into_paginator()
        .send();

    let mut v = Vec::new();

    while let Some(list_policies_output) = list_policies.next().await {
        match list_policies_output {
            Ok(list_policies) => {
                if let Some(policies) = list_policies.policies() {
                    for policy in policies {
                        let policy_name = policy
                            .policy_name()
                            .unwrap_or("Missing policy name.")
                            .to_string();
                        println!("{}", policy_name);
                        v.push(policy_name);
                    }
                }
            }
            Err(err) => return Err(err),
        }
    }

    Ok(v)
}
```

- For API details, see [ListPolicies](#) in AWS SDK for Rust API reference.

Code examples

- [Actions for IAM using AWS SDKs \(p. 677\)](#)
 - [Add an IAM user to a group using an AWS SDK \(p. 679\)](#)
 - [Attach an IAM policy to a role using an AWS SDK \(p. 679\)](#)
 - [Attach an IAM policy to a user using an AWS SDK \(p. 690\)](#)
 - [Attach an inline policy to an IAM role using an AWS SDK \(p. 691\)](#)
 - [Create an IAM SAML provider using an AWS SDK \(p. 693\)](#)
 - [Create an IAM group using an AWS SDK \(p. 694\)](#)
 - [Create an IAM policy using an AWS SDK \(p. 695\)](#)
 - [Create an IAM policy version using an AWS SDK \(p. 705\)](#)
 - [Create an IAM role using an AWS SDK \(p. 706\)](#)
 - [Create an IAM service-linked role using an AWS SDK \(p. 715\)](#)
 - [Create an IAM user using an AWS SDK \(p. 719\)](#)
 - [Create an IAM access key using an AWS SDK \(p. 727\)](#)
 - [Create an alias for an IAM account using an AWS SDK \(p. 735\)](#)
 - [Create an inline IAM policy for a group using an AWS SDK \(p. 738\)](#)
 - [Create an inline IAM policy for a user using an AWS SDK \(p. 739\)](#)
 - [Delete an IAM SAML provider using an AWS SDK \(p. 741\)](#)
 - [Delete an IAM group using an AWS SDK \(p. 742\)](#)
 - [Delete an IAM group policy using an AWS SDK \(p. 743\)](#)
 - [Delete an IAM policy using an AWS SDK \(p. 744\)](#)
 - [Delete an IAM role using an AWS SDK \(p. 750\)](#)
 - [Delete an IAM role policy using an AWS SDK \(p. 755\)](#)
 - [Delete an IAM server certificate using an AWS SDK \(p. 756\)](#)
 - [Delete an IAM service-linked role using an AWS SDK \(p. 758\)](#)
 - [Delete an IAM user using an AWS SDK \(p. 760\)](#)
 - [Delete an IAM access key using an AWS SDK \(p. 768\)](#)
 - [Delete an IAM account alias using an AWS SDK \(p. 776\)](#)
 - [Delete an inline IAM policy from a user using an AWS SDK \(p. 779\)](#)
 - [Detach an IAM policy from a role using an AWS SDK \(p. 782\)](#)
 - [Detach an IAM policy from a user using an AWS SDK \(p. 789\)](#)
 - [Generate a credential report from IAM using an AWS SDK \(p. 791\)](#)
 - [Get a credential report from IAM using an AWS SDK \(p. 791\)](#)
 - [Get a detailed IAM authorization report for your account using an AWS SDK \(p. 792\)](#)
 - [Get an IAM policy using an AWS SDK \(p. 793\)](#)
 - [Get an IAM policy version using an AWS SDK \(p. 798\)](#)
 - [Get an IAM role using an AWS SDK \(p. 799\)](#)
 - [Get an IAM server certificate using an AWS SDK \(p. 803\)](#)
 - [Get an IAM service-linked role's deletion status using an AWS SDK \(p. 804\)](#)
 - [Get a summary of account usage from IAM using an AWS SDK \(p. 805\)](#)
 - [Get an IAM user using an AWS SDK \(p. 806\)](#)
 - [Get data about the last use of an IAM access key using an AWS SDK \(p. 808\)](#)
 - [Get the IAM account password policy using an AWS SDK \(p. 811\)](#)
 - [List SAML providers for IAM using an AWS SDK \(p. 814\)](#)
 - [List a user's IAM access keys using an AWS SDK \(p. 817\)](#)
 - [List IAM account aliases using an AWS SDK \(p. 824\)](#)

- [List IAM groups using an AWS SDK \(p. 828\)](#)
- [List inline policies for an IAM role using an AWS SDK \(p. 832\)](#)
- [List inline IAM policies for a user using an AWS SDK \(p. 837\)](#)
- [List IAM policies using an AWS SDK \(p. 837\)](#)
- [List policies attached to an IAM role using an AWS SDK \(p. 844\)](#)
- [List IAM roles using an AWS SDK \(p. 848\)](#)
- [List IAM server certificates using an AWS SDK \(p. 853\)](#)
- [List IAM users using an AWS SDK \(p. 855\)](#)
- [Remove an IAM user from a group using an AWS SDK \(p. 864\)](#)
- [Update an IAM server certificate using an AWS SDK \(p. 865\)](#)
- [Update an IAM user using an AWS SDK \(p. 866\)](#)
- [Update an IAM access key using an AWS SDK \(p. 870\)](#)
- [Upload an IAM server certificate using an AWS SDK \(p. 873\)](#)
- [Scenarios for IAM using AWS SDKs \(p. 874\)](#)
 - [Create an IAM group and add a user to the group using an AWS SDK \(p. 874\)](#)
 - [Create an IAM user and assume a role with AWS STS using an AWS SDK \(p. 891\)](#)
 - [Create read-only and read-write IAM users using an AWS SDK \(p. 973\)](#)
 - [Manage IAM access keys using an AWS SDK \(p. 980\)](#)
 - [Manage IAM policies using an AWS SDK \(p. 982\)](#)
 - [Manage IAM roles using an AWS SDK \(p. 986\)](#)
 - [Manage your IAM account using an AWS SDK \(p. 989\)](#)
 - [Roll back an IAM policy version using an AWS SDK \(p. 993\)](#)
 - [Work with the IAM Policy Builder API using an AWS SDK \(p. 994\)](#)

Actions for IAM using AWS SDKs

The following code examples demonstrate how to perform individual IAM actions with AWS SDKs. These excerpts call the IAM API and are code excerpts from larger programs that must be run in context. Each example includes a link to GitHub, where you can find instructions for setting up and running the code.

The following examples include only the most commonly used actions. For a complete list, see the [AWS Identity and Access Management \(IAM\) API Reference](#).

Examples

- [Add an IAM user to a group using an AWS SDK \(p. 679\)](#)
- [Attach an IAM policy to a role using an AWS SDK \(p. 679\)](#)
- [Attach an IAM policy to a user using an AWS SDK \(p. 690\)](#)
- [Attach an inline policy to an IAM role using an AWS SDK \(p. 691\)](#)
- [Create an IAM SAML provider using an AWS SDK \(p. 693\)](#)
- [Create an IAM group using an AWS SDK \(p. 694\)](#)
- [Create an IAM policy using an AWS SDK \(p. 695\)](#)
- [Create an IAM policy version using an AWS SDK \(p. 705\)](#)
- [Create an IAM role using an AWS SDK \(p. 706\)](#)
- [Create an IAM service-linked role using an AWS SDK \(p. 715\)](#)
- [Create an IAM user using an AWS SDK \(p. 719\)](#)
- [Create an IAM access key using an AWS SDK \(p. 727\)](#)

- [Create an alias for an IAM account using an AWS SDK \(p. 735\)](#)
- [Create an inline IAM policy for a group using an AWS SDK \(p. 738\)](#)
- [Create an inline IAM policy for a user using an AWS SDK \(p. 739\)](#)
- [Delete an IAM SAML provider using an AWS SDK \(p. 741\)](#)
- [Delete an IAM group using an AWS SDK \(p. 742\)](#)
- [Delete an IAM group policy using an AWS SDK \(p. 743\)](#)
- [Delete an IAM policy using an AWS SDK \(p. 744\)](#)
- [Delete an IAM role using an AWS SDK \(p. 750\)](#)
- [Delete an IAM role policy using an AWS SDK \(p. 755\)](#)
- [Delete an IAM server certificate using an AWS SDK \(p. 756\)](#)
- [Delete an IAM service-linked role using an AWS SDK \(p. 758\)](#)
- [Delete an IAM user using an AWS SDK \(p. 760\)](#)
- [Delete an IAM access key using an AWS SDK \(p. 768\)](#)
- [Delete an IAM account alias using an AWS SDK \(p. 776\)](#)
- [Delete an inline IAM policy from a user using an AWS SDK \(p. 779\)](#)
- [Detach an IAM policy from a role using an AWS SDK \(p. 782\)](#)
- [Detach an IAM policy from a user using an AWS SDK \(p. 789\)](#)
- [Generate a credential report from IAM using an AWS SDK \(p. 791\)](#)
- [Get a credential report from IAM using an AWS SDK \(p. 791\)](#)
- [Get a detailed IAM authorization report for your account using an AWS SDK \(p. 792\)](#)
- [Get an IAM policy using an AWS SDK \(p. 793\)](#)
- [Get an IAM policy version using an AWS SDK \(p. 798\)](#)
- [Get an IAM role using an AWS SDK \(p. 799\)](#)
- [Get an IAM server certificate using an AWS SDK \(p. 803\)](#)
- [Get an IAM service-linked role's deletion status using an AWS SDK \(p. 804\)](#)
- [Get a summary of account usage from IAM using an AWS SDK \(p. 805\)](#)
- [Get an IAM user using an AWS SDK \(p. 806\)](#)
- [Get data about the last use of an IAM access key using an AWS SDK \(p. 808\)](#)
- [Get the IAM account password policy using an AWS SDK \(p. 811\)](#)
- [List SAML providers for IAM using an AWS SDK \(p. 814\)](#)
- [List a user's IAM access keys using an AWS SDK \(p. 817\)](#)
- [List IAM account aliases using an AWS SDK \(p. 824\)](#)
- [List IAM groups using an AWS SDK \(p. 828\)](#)
- [List inline policies for an IAM role using an AWS SDK \(p. 832\)](#)
- [List inline IAM policies for a user using an AWS SDK \(p. 837\)](#)
- [List IAM policies using an AWS SDK \(p. 837\)](#)
- [List policies attached to an IAM role using an AWS SDK \(p. 844\)](#)
- [List IAM roles using an AWS SDK \(p. 848\)](#)
- [List IAM server certificates using an AWS SDK \(p. 853\)](#)
- [List IAM users using an AWS SDK \(p. 855\)](#)
- [Remove an IAM user from a group using an AWS SDK \(p. 864\)](#)
- [Update an IAM server certificate using an AWS SDK \(p. 865\)](#)

- [Update an IAM user using an AWS SDK \(p. 866\)](#)
- [Update an IAM access key using an AWS SDK \(p. 870\)](#)
- [Upload an IAM server certificate using an AWS SDK \(p. 873\)](#)

Add an IAM user to a group using an AWS SDK

The following code example shows how to add a user to an IAM group.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Create a group and add a user \(p. 874\)](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
///<summary>
/// Add an existing IAM user to an existing IAM group.
///</summary>
///<param name="userName">The username of the user to add.</param>
///<param name="groupName">The name of the group to add the user to.</param>
///<returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> AddUserToGroupAsync(string userName, string groupName)
{
    var response = await _IAMService.AddUserToGroupAsync(new
AddUserToGroupRequest
{
    GroupName = groupName,
    UserName = userName,
});

    return response.HttpStatusCode == HttpStatusCode.OK;
}
```

- For API details, see [AddUserToGroup](#) in *AWS SDK for .NET API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Attach an IAM policy to a role using an AWS SDK

The following code examples show how to attach an IAM policy to a role.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- [Create a group and add a user \(p. 874\)](#)

- [Create a user and assume a role \(p. 891\)](#)
- [Manage roles \(p. 986\)](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Attach an IAM policy to a role.
/// </summary>
/// <param name="policyArn">The policy to attach.</param>
/// <param name="roleName">The role that the policy will be attached to.</param>
public async Task<bool> AttachRolePolicyAsync(string policyArn, string roleName)
{
    var response = await _IAMService.AttachRolePolicyAsync(new
AttachRolePolicyRequest
{
    PolicyArn = policyArn,
    RoleName = roleName,
});

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- For API details, see [AttachRolePolicy](#) in *AWS SDK for .NET API Reference*.

Bash

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
    printf "%s\n" "$*" 2>&1
}

#####
# function iam_attach_role_policy
#
# This function attaches an IAM policy to a role.
#
# Parameters:
#     -n role_name -- The name of the IAM role.
```

```

#           -p policy_ARN -- The IAM policy document ARN..
#
# Returns:
#     0 - If successful.
#     1 - If it fails.
#####
function iam_attach_role_policy() {
    local role_name policy_arn response
    local option OPTARG # Required to use getopts command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_attach_role_policy"
        echo "Attaches an AWS Identity and Access Management (IAM) policy to an IAM"
        echo "role."
        echo "  -n role_name  The name of the IAM role."
        echo "  -p policy_ARN -- The IAM policy document ARN."
        echo ""
    }

    # Retrieve the calling parameters.
    while getopts "n:p:h" option; do
        case "${option}" in
            n) role_name="${OPTARG}" ;;
            p) policy_arn="${OPTARG}" ;;
            h)
                usage
                return 0
                ;;
            \?)
                echo "Invalid parameter"
                usage
                return 1
                ;;
        esac
    done
    export OPTIND=1

    if [[ -z "$role_name" ]]; then
        errecho "ERROR: You must provide a role name with the -n parameter."
        usage
        return 1
    fi

    if [[ -z "$policy_arn" ]]; then
        errecho "ERROR: You must provide a policy ARN with the -p parameter."
        usage
        return 1
    fi

    response=$(aws iam attach-role-policy \
        --role-name "$role_name" \
        --policy-arn "$policy_arn")

    local error_code=${?}

    if [[ $error_code -ne 0 ]]; then
        aws_cli_error_log $error_code
        errecho "ERROR: AWS reports attach-role-policy operation failed.\n$response"
        return 1
    fi

    echo "$response"

    return 0
}

```

- For API details, see [AttachRolePolicy](#) in *AWS CLI Command Reference*.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
bool AwsDoc::IAM::attachRolePolicy(const Aws::String &roleName,
                                    const Aws::String &policyArn,
                                    const Aws::Client::ClientConfiguration
&clientConfig) {
    Aws::IAM::IAMClient iam(clientConfig);

    Aws::IAM::Model::ListAttachedRolePoliciesRequest list_request;
    list_request.SetRoleName(roleName);

    bool done = false;
    while (!done) {
        auto list_outcome = iam.ListAttachedRolePolicies(list_request);
        if (!list_outcome.IsSuccess()) {
            std::cerr << "Failed to list attached policies of role " <<
                roleName << ":" << list_outcome.GetError().GetMessage() <<
                std::endl;
            return false;
        }

        const auto &policies = list_outcome.GetResult().GetAttachedPolicies();
        if (std::any_of(policies.cbegin(), policies.cend(),
                      [=](const Aws::IAM::Model::AttachedPolicy &policy) {
                          return policy.GetPolicyArn() == policyArn;
                      })) {
            std::cout << "Policy " << policyArn <<
                " is already attached to role " << roleName << std::endl;
            return true;
        }

        done = !list_outcome.GetResult().GetIsTruncated();
        list_request.SetMarker(list_outcome.GetResult().GetMarker());
    }

    Aws::IAM::Model::AttachRolePolicyRequest request;
    request.SetRoleName(roleName);
    request.SetPolicyArn(policyArn);

    Aws::IAM::Model::AttachRolePolicyOutcome outcome =
iam.AttachRolePolicy(request);
    if (!outcome.IsSuccess()) {
        std::cerr << "Failed to attach policy " << policyArn << " to role " <<
            roleName << ":" << outcome.GetError().GetMessage() << std::endl;
    }
    else {
        std::cout << "Successfully attached policy " << policyArn << " to role " <<
            roleName << std::endl;
    }

    return outcome.IsSuccess();
}
```

- For API details, see [AttachRolePolicy](#) in *AWS SDK for C++ API Reference*.

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
// It contains an IAM service client that is used to perform role actions.
type RoleWrapper struct {
    IamClient *iam.Client
}

// AttachRolePolicy attaches a policy to a role.
func (wrapper RoleWrapper) AttachRolePolicy(policyArn string, roleName string) error {
    _, err := wrapper.IamClient.AttachRolePolicy(context.TODO(),
        &iam.AttachRolePolicyInput{
            PolicyArn: aws.String(policyArn),
            RoleName:  aws.String(roleName),
        })
    if err != nil {
        log.Printf("Couldn't attach policy %v to role %v. Here's why: %v\n", policyArn,
            roleName, err)
    }
    return err
}
```

- For API details, see [AttachRolePolicy](#) in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public static void attachIAMRolePolicy(IamClient iam, String roleName, String
policyArn ) {

    try {
        ListAttachedRolePoliciesRequest request =
ListAttachedRolePoliciesRequest.builder()
            .roleName(roleName)
            .build();
```

```

ListAttachedRolePoliciesResponse response =
iam.listAttachedRolePolicies(request);
List<AttachedPolicy> attachedPolicies = response.attachedPolicies();

// Ensure that the policy is not attached to this role
String polArn = "";
for (AttachedPolicy policy: attachedPolicies) {
    polArn = policy.policyArn();
    if (polArn.compareTo(policyArn)==0) {
        System.out.println(roleName + " policy is already attached to
this role.");
        return;
    }
}

AttachRolePolicyRequest attachRequest =
AttachRolePolicyRequest.builder()
    .roleName(roleName)
    .policyArn(policyArn)
    .build();

iam.attachRolePolicy(attachRequest);

System.out.println("Successfully attached policy " + policyArn +
    " to role " + roleName);

} catch (IamException e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}
System.out.println("Done");
}

```

- For API details, see [AttachRolePolicy](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Attach the policy.

```

import { AttachRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} policyArn
 * @param {string} roleName
 */
export const attachRolePolicy = (policyArn, roleName) => {
    const command = new AttachRolePolicyCommand({
        PolicyArn: policyArn,
        RoleName: roleName,
    });

    return client.send(command);
};

```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [AttachRolePolicy](#) in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// Load the AWS SDK for Node.js
var AWS = require('aws-sdk');
// Set the region
AWS.config.update({region: 'REGION'});

// Create the IAM service object
var iam = new AWS.IAM({apiVersion: '2010-05-08'});

var paramsRoleList = [
  RoleName: process.argv[2]
];

iam.listAttachedRolePolicies(paramsRoleList, function(err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    var myRolePolicies = data.AttachedPolicies;
    myRolePolicies.forEach(function (val, index, array) {
      if (myRolePolicies[index].PolicyName === 'AmazonDynamoDBFullAccess') {
        console.log("AmazonDynamoDBFullAccess is already attached to this role.")
        process.exit();
      }
    });
    var params = {
      PolicyArn: 'arn:aws:iam::aws:policy/AmazonDynamoDBFullAccess',
      RoleName: process.argv[2]
    };
    iam.attachRolePolicy(params, function(err, data) {
      if (err) {
        console.log("Unable to attach policy to role", err);
      } else {
        console.log("Role attached successfully");
      }
    });
  }
});
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [AttachRolePolicy](#) in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

Note

This is prerelease documentation for a feature in preview release. It is subject to change.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
suspend fun attachIAMRolePolicy(roleNameVal: String, policyArnVal: String) {

    val request = ListAttachedRolePoliciesRequest {
        roleName = roleNameVal
    }

    IamClient { region = "AWS_GLOBAL" }.use { iamClient ->
        val response = iamClient.listAttachedRolePolicies(request)
        val attachedPolicies = response.attachedPolicies

        // Ensure that the policy is not attached to this role.
        val checkStatus: Int
        if (attachedPolicies != null) {
            checkStatus = checkList(attachedPolicies, policyArnVal)
            if (checkStatus == -1)
                return
        }

        val policyRequest = AttachRolePolicyRequest {
            roleName = roleNameVal
            policyArn = policyArnVal
        }
        iamClient.attachRolePolicy(policyRequest)
        println("Successfully attached policy $policyArnVal to role $roleNameVal")
    }
}

fun checkList(attachedPolicies: List<AttachedPolicy>, policyArnVal: String): Int {
    for (policy in attachedPolicies) {
        val polArn = policy.policyArn.toString()

        if (polArn.compareTo(policyArnVal) == 0) {
            println("The policy is already attached to this role.")
            return -1
        }
    }
    return 0
}
```

- For API details, see [AttachRolePolicy](#) in *AWS SDK for Kotlin API reference*.

PHP

SDK for PHP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
$uuid = uniqid();
$service = new IAMService();

$assumeRolePolicyDocument = "{"
    \\"Version\\": \\"2012-10-17\\",
    \\"Statement\\": [{
```

```

        \\"Effect\\": \\\"Allow\\",
        \\"Principal\\": {\\\"AWS\\": \\\"{$user['Arn']}\\\"},
        \\"Action\\": \\\"sts:AssumeRole\\"
    }]
};

$assumeRoleRole = $service->createRole("iam_demo_role_$uuid",
    $assumeRolePolicyDocument);
echo "Created role: {$assumeRoleRole['RoleName']}\\n";

$listAllBucketsPolicyDocument = "{
    \\"Version\\": \\\"2012-10-17\\",
    \\"Statement\\": [{{
        \\"Effect\\": \\\"Allow\\",
        \\"Action\\": \\\"s3>ListAllMyBuckets\\",
        \\"Resource\\": \\\"arn:aws:s3:::*\\"
    }}];
};

$listAllBucketsPolicy = $service->createPolicy("iam_demo_policy_$uuid",
    $listAllBucketsPolicyDocument);
echo "Created policy: {$listAllBucketsPolicy['PolicyName']}\\n";

$service->attachRolePolicy($assumeRoleRole['RoleName'],
    $listAllBucketsPolicy['Arn']);

    public function attachRolePolicy($roleName, $policyArn)
{
    return $this->customWaiter(function () use ($roleName, $policyArn) {
        $this->iamClient->attachRolePolicy([
            'PolicyArn' => $policyArn,
            'RoleName' => $roleName,
        ]);
    });
}

```

- For API details, see [AttachRolePolicy](#) in *AWS SDK for PHP API Reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Attach a policy to a role using the Boto3 Policy object.

```

def attach_to_role(role_name, policy_arn):
    """
    Attaches a policy to a role.

    :param role_name: The name of the role. **Note** this is the name, not the ARN.
    :param policy_arn: The ARN of the policy.
    """
    try:
        iam.Policy(policy_arn).attach_role(RoleName=role_name)
        logger.info("Attached policy %s to role %s.", policy_arn, role_name)
    except ClientError:
        logger.exception("Couldn't attach policy %s to role %s.", policy_arn,
                        role_name)
        raise

```

Attach a policy to a role using the Boto3 Role object.

```
def attach_policy(role_name, policy_arn):
    """
    Attaches a policy to a role.

    :param role_name: The name of the role. **Note** this is the name, not the ARN.
    :param policy_arn: The ARN of the policy.
    """
    try:
        iam.Role(role_name).attach_policy(PolicyArn=policy_arn)
        logger.info("Attached policy %s to role %s.", policy_arn, role_name)
    except ClientError:
        logger.exception("Couldn't attach policy %s to role %s.", policy_arn, role_name)
        raise
```

- For API details, see [AttachRolePolicy](#) in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
# Creates a policy that grants permission to list S3 buckets in the account, and
# then attaches the policy to a role.
#
# @param policy_name [String] The name to give the policy.
# @param role [Aws::IAM::Role] The role that the policy is attached to.
# @return [Aws::IAM::Policy] The newly created policy.
def create_and_attach_role_policy(policy_name, role)
    policy = @iam_resource.create_policy(
        policy_name: policy_name,
        policy_document: [
            {
                Version: "2012-10-17",
                Statement: [
                    {
                        Effect: "Allow",
                        Action: "s3>ListAllMyBuckets",
                        Resource: "arn:aws:s3:::*"
                    }
                ].to_json
            }
        ]
    )
    role.attach_policy(policy_arn: policy.arn)
    puts("Created policy #{policy.policy_name} and attached it to role
#{role.name}.")
    rescue Aws::Errors::ServiceError => e
        puts("Couldn't create a policy and attach it to role #{role.name}. Here's why:
")
        puts("\t#{e.code}: #{e.message}")
        raise
    else
        policy
    end
```

- For API details, see [AttachRolePolicy](#) in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
pub async fn attach_role_policy(
    client: &iamClient,
    role: &Role,
    policy: &Policy,
) -> Result<AttachRolePolicyOutput, SdkError<AttachRolePolicyError>> {
    client
        .attach_role_policy()
        .role_name(role.role_name.as_ref().unwrap())
        .policy_arn(policy.arn.as_ref().unwrap())
        .send()
        .await
}
```

- For API details, see [AttachRolePolicy](#) in *AWS SDK for Rust API reference*.

Swift

SDK for Swift

Note

This is prerelease documentation for an SDK in preview release. It is subject to change.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public func attachRolePolicy(role: String, policyArn: String) async throws {
    let input = AttachRolePolicyInput(
        policyArn: policyArn,
        roleName: role
    )
    do {
        _ = try await client.attachRolePolicy(input: input)
    } catch {
        throw error
    }
}
```

- For API details, see [AttachRolePolicy](#) in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Attach an IAM policy to a user using an AWS SDK

The following code examples show how to attach an IAM policy to a user.

Warning

To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as [AWS IAM Identity Center \(successor to AWS Single Sign-On\)](#).

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Create read-only and read-write users \(p. 973\)](#)

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
def attach_policy(user_name, policy_arn):
    """
    Attaches a policy to a user.

    :param user_name: The name of the user.
    :param policy_arn: The Amazon Resource Name (ARN) of the policy.
    """
    try:
        iam.User(user_name).attach_policy(PolicyArn=policy_arn)
        logger.info("Attached policy %s to user %s.", policy_arn, user_name)
    except ClientError:
        logger.exception("Couldn't attach policy %s to user %s.", policy_arn,
                         user_name)
        raise
```

- For API details, see [AttachUserPolicy](#) in *AWS SDK for Python (Boto3) API Reference*.

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
pub async fn attach_user_policy(
    client: &iamClient,
    user_name: &str,
    policy_arn: &str,
) -> Result<(), iamError> {
    client
        .attach_user_policy()
```

```
.user_name(user_name)
.policy_arn(policy_arn)
.send()
.await?;

    Ok(())
}
```

- For API details, see [AttachUserPolicy](#) in *AWS SDK for Rust API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Attach an inline policy to an IAM role using an AWS SDK

The following code examples show how to attach an inline policy to an IAM role.

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Update the inline policy document embedded in a role.
/// </summary>
/// <param name="policyName">The name of the policy to embed.</param>
/// <param name="roleName">The name of the role to update.</param>
/// <param name="policyDocument">The policy document that defines the role.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> PutRolePolicyAsync(string policyName, string roleName,
string policyDocument)
{
    var request = new PutRolePolicyRequest
    {
        PolicyName = policyName,
        RoleName = roleName,
        PolicyDocument = policyDocument
    };

    var response = await _IAMService.PutRolePolicyAsync(request);
    return response.HttpStatusCode == HttpStatusCode.OK;
}
```

- For API details, see [PutRolePolicy](#) in *AWS SDK for .NET API Reference*.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```

bool AwsDoc::IAM::putRolePolicy(
    const Aws::String &roleName,
    const Aws::String &policyName,
    const Aws::String &policyDocument,
    const Aws::Client::ClientConfiguration &clientConfig) {
    Aws::IAM::IAMClient iamClient(clientConfig);
    Aws::IAM::Model::PutRolePolicyRequest request;

    request.SetRoleName(roleName);
    request.SetPolicyName(policyName);
    request.SetPolicyDocument(policyDocument);

    Aws::IAM::Model::PutRolePolicyOutcome outcome =
        iamClient.PutRolePolicy(request);
    if (!outcome.IsSuccess()) {
        std::cerr << "Error putting policy on role. " <<
            outcome.GetError().GetMessage() << std::endl;
    }
    else {
        std::cout << "Successfully put the role policy." << std::endl;
    }

    return outcome.IsSuccess();
}

```

- For API details, see [PutRolePolicy](#) in *AWS SDK for C++ API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```

import { PutRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const examplePolicyDocument = JSON.stringify({
    Version: "2012-10-17",
    Statement: [
        {
            Sid: "VisualEditor0",
            Effect: "Allow",
            Action: [
                "s3>ListBucketMultipartUploads",
                "s3>ListBucketVersions",
                "s3>ListBucket",
                "s3>ListMultipartUploadParts",
            ],
            Resource: "arn:aws:s3:::some-test-bucket",
        },
        {
            Sid: "VisualEditor1",
            Effect: "Allow",
            Action: [
                "s3>ListStorageLensConfigurations",
                "s3>ListAccessPointsForObjectLambda",
                "s3>ListAllMyBuckets",
                "s3>ListAccessPoints",
                "s3>ListJobs",
            ],
        },
    ],
});

```

```

        "s3>ListMultiRegionAccessPoints",
    ],
    Resource: "*",
},
],
});

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 * @param {string} policyName
 * @param {string} policyDocument
 */
export const putRolePolicy = async (roleName, policyName, policyDocument) => {
    const command = new PutRolePolicyCommand({
        RoleName: roleName,
        PolicyName: policyName,
        PolicyDocument: policyDocument,
    });

    const response = await client.send(command);
    console.log(response);
    return response;
};

```

- For API details, see [PutRolePolicy](#) in *AWS SDK for JavaScript API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Create an IAM SAML provider using an AWS SDK

The following code example shows how to create an AWS Identity and Access Management (IAM) SAML provider.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```

import { CreateSAMLProviderCommand, IAMClient } from "@aws-sdk/client-iam";
import { readFileSync } from "fs";
import * as path from "path";
import { dirnameFromMetaUrl } from "libs/utils/util-fs.js";

const client = new IAMClient({});

/**
 * This sample document was generated using Auth0.
 * For more information on generating this document,
 * see https://docs.aws.amazon.com/IAM/latest/UserGuide/
 * id_roles_providers_create_saml.html#samlstep1.
 */
const sampleMetadataDocument = readFileSync(
    path.join(

```

```

        dirnameFromMetaUrl(import.meta.url),
        "../../../../../resources/sample_files/sample_saml_metadata.xml"
    );
}

/**
 *
 * @param {*} providerName
 * @returns
 */
export const createSAMLProvider = async (providerName) => {
    const command = new CreateSAMLProviderCommand({
        Name: providerName,
        SAMLMetadataDocument: sampleMetadataDocument.toString(),
    });

    const response = await client.send(command);
    console.log(response);
    return response;
};

```

- For API details, see [CreateSAMLProvider](#) in *AWS SDK for JavaScript API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Create an IAM group using an AWS SDK

The following code examples show how to create an IAM group.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Create a group and add a user \(p. 874\)](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```

///<summary>
/// Create an IAM group.
///</summary>
///<param name="groupName">The name to give the IAM group.</param>
///<returns>The IAM group that was created.</returns>
public async Task<Group> CreateGroupAsync(string groupName)
{
    var response = await _IAMService.CreateGroupAsync(new CreateGroupRequest
{ GroupName = groupName });
    return response.Group;
}

```

- For API details, see [CreateGroup](#) in *AWS SDK for .NET API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import { CreateGroupCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} groupName
 */
export const createGroup = async (groupName) => {
  const command = new CreateGroupCommand({ GroupName: groupName });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- For API details, see [CreateGroup](#) in *AWS SDK for JavaScript API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Create an IAM policy using an AWS SDK

The following code examples show how to create an IAM policy.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- [Create a group and add a user \(p. 874\)](#)
- [Create a user and assume a role \(p. 891\)](#)
- [Create read-only and read-write users \(p. 973\)](#)
- [Manage policies \(p. 982\)](#)
- [Work with the IAM Policy Builder API \(p. 994\)](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Create an IAM policy.
/// </summary>
```

```
/// <param name="policyName">The name to give the new IAM policy.</param>
/// <param name="policyDocument">The policy document for the new policy.</param>
/// <returns>The new IAM policy object.</returns>
public async Task<ManagedPolicy> CreatePolicyAsync(string policyName, string
policyDocument)
{
    var response = await _IAMService.CreatePolicyAsync(new CreatePolicyRequest
    {
        PolicyDocument = policyDocument,
        PolicyName = policyName,
    });

    return response.Policy;
}
```

- For API details, see [CreatePolicy](#) in *AWS SDK for .NET API Reference*.

Bash

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
    printf "%s\n" "$*" 2>&1
}

#####
# function iam_create_policy
#
# This function creates an IAM policy.
#
# Parameters:
#     -n policy_name -- The name of the IAM policy.
#     -p policy_json -- The policy document.
#
# Returns:
#     0 - If successful.
#     1 - If it fails.
#####
function iam_create_policy() {
    local policy_name policy_document response
    local option OPTARG # Required to use getopts command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_create_policy"
        echo "Creates an AWS Identity and Access Management (IAM) policy."
        echo " -n policy_name The name of the IAM policy."
        echo " -p policy_json -- The policy document."
        echo ""
    }
}
```

```

# Retrieve the calling parameters.
while getopts "n:p:h" option; do
    case "${option}" in
        n) policy_name="${OPTARG}" ;;
        p) policy_document="${OPTARG}" ;;
        h)
            usage
            return 0
            ;;
        \?) echo "Invalid parameter"
            usage
            return 1
            ;;
    esac
done
export OPTIND=1

if [[ -z "$policy_name" ]]; then
    errecho "ERROR: You must provide a policy name with the -n parameter."
    usage
    return 1
fi

if [[ -z "$policy_document" ]]; then
    errecho "ERROR: You must provide a policy document with the -p parameter."
    usage
    return 1
fi

response=$(aws iam create-policy \
    --policy-name "$policy_name" \
    --policy-document "$policy_document" \
    --output text \
    --query Policy.Arn)

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports create-policy operation failed.\n$response"
    return 1
fi

echo "$response"
}

```

- For API details, see [CreatePolicy](#) in *AWS CLI Command Reference*.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```

Aws::String AwsDoc::IAM::createPolicy(const Aws::String &policyName,
                                       const Aws::String &rsrcArn,
                                       const Aws::Client::ClientConfiguration
                                       &clientConfig) {
    Aws::IAM::IAMClient iam(clientConfig);

```

```

Aws::IAM::Model::CreatePolicyRequest request;
request.SetPolicyName(policyName);
request.SetPolicyDocument(BuildSamplePolicyDocument(rsrcArn));

Aws::IAM::Model::CreatePolicyOutcome outcome = iam.CreatePolicy(request);
Aws::String result;
if (!outcome.IsSuccess()) {
    std::cerr << "Error creating policy " << policyName << ": " <<
        outcome.GetError().GetMessage() << std::endl;
}
else {
    result = outcome.GetResult().GetPolicy().GetArn();
    std::cout << "Successfully created policy " << policyName <<
        std::endl;
}

return result;
}

Aws::String AwsDoc::IAM::BuildSamplePolicyDocument(const Aws::String &rsrc_arn) {
    std::stringstream stringStream;
    stringStream << "{"
        << "    \"Version\": \"2012-10-17\","
        << "    \"Statement\": ["
        << "        {"
        << "            \"Effect\": \"Allow\","
        << "            \"Action\": \"logs>CreateLogGroup\","
        << "            \"Resource\": \""
        << rsrc_arn
        << "\""
        << "        },"
        << "        {"
        << "            \"Effect\": \"Allow\","
        << "            \"Action\": ["
        << "                \"dynamodb>DeleteItem\","
        << "                \"dynamodb>GetItem\","
        << "                \"dynamodb>PutItem\","
        << "                \"dynamodb>Scan\","
        << "                \"dynamodb>UpdateItem\""
        << "            ],"
        << "            \"Resource\": \""
        << rsrc_arn
        << "\""
        << "        }"
        << "    ]"
        << "}";
    }

    return stringStream.str();
}

```

- For API details, see [CreatePolicy](#) in *AWS SDK for C++ API Reference*.

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// PolicyWrapper encapsulates AWS Identity and Access Management (IAM) policy
// actions
// used in the examples.
// It contains an IAM service client that is used to perform policy actions.
type PolicyWrapper struct {
    IamClient *iam.Client
}

// CreatePolicy creates a policy that grants a list of actions to the specified
// resource.
// PolicyDocument shows how to work with a policy document as a data structure and
// serialize it to JSON by using Go's JSON marshaler.
func (wrapper PolicyWrapper) CreatePolicy(policyName string, actions []string,
    resourceArn string) (*types.Policy, error) {
    var policy *types.Policy
    policyDoc := PolicyDocument{
        Version: "2012-10-17",
        Statement: []PolicyStatement{
            Effect: "Allow",
            Action: actions,
            Resource: aws.String(resourceArn),
        },
    }
    policyBytes, err := json.Marshal(policyDoc)
    if err != nil {
        log.Printf("Couldn't create policy document for %v. Here's why: %v\n",
            resourceArn, err)
        return nil, err
    }
    result, err := wrapper.IamClient.CreatePolicy(context.TODO(),
        &iam.CreatePolicyInput{
            PolicyDocument: aws.String(string(policyBytes)),
            PolicyName:     aws.String(policyName),
        })
    if err != nil {
        log.Printf("Couldn't create policy %v. Here's why: %v\n", policyName, err)
    } else {
        policy = result.Policy
    }
    return policy, err
}
```

- For API details, see [CreatePolicy](#) in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public static String createIAMPolicy(IamClient iam, String policyName ) {

    try {
        // Create an IamWaiter object.
        IamWaiter iamWaiter = iam.waiter();

        CreatePolicyRequest request = CreatePolicyRequest.builder()
```

```
.policyName(policyName)
.policyDocument(PolicyDocument)
.build();

CreatePolicyResponse response = iam.createPolicy(request);

// Wait until the policy is created.
GetPolicyRequest polRequest = GetPolicyRequest.builder()
.policyArn(response.policy().arn())
.build();

WaiterResponse<GetPolicyResponse> waitUntilPolicyExists =
iamWaiter.waitUntilPolicyExists(polRequest);

waitUntilPolicyExists.matched().response().ifPresent(System.out::println);
return response.policy().arn();

} catch (IamException e) {
System.err.println(e.awsErrorDetails().errorMessage());
System.exit(1);
}
return "" ;
}
```

- For API details, see [CreatePolicy](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Create the policy.

```
import { CreatePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} policyName
 */
export const createPolicy = (policyName) => {
  const command = new CreatePolicyCommand({
    PolicyDocument: JSON.stringify({
      Version: "2012-10-17",
      Statement: [
        {
          Effect: "Allow",
          Action: "*",
          Resource: "*",
        },
      ],
    }),
    PolicyName: policyName,
  });

  return client.send(command);
};
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [CreatePolicy](#) in [AWS SDK for JavaScript API Reference](#).

SDK for JavaScript (v2)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// Load the AWS SDK for Node.js
var AWS = require('aws-sdk');
// Set the region
AWS.config.update({region: 'REGION'});

// Create the IAM service object
var iam = new AWS.IAM({apiVersion: '2010-05-08'});

var myManagedPolicy = {
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "logs>CreateLogGroup",
            "Resource": "RESOURCE_ARN"
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb>DeleteItem",
                "dynamodb>GetItem",
                "dynamodb>PutItem",
                "dynamodb>Scan",
                "dynamodb>UpdateItem"
            ],
            "Resource": "RESOURCE_ARN"
        }
    ]
};

var params = {
    PolicyDocument: JSON.stringify(myManagedPolicy),
    PolicyName: 'myDynamoDBPolicy',
};

iam.createPolicy(params, function(err, data) {
    if (err) {
        console.log("Error", err);
    } else {
        console.log("Success", data);
    }
});
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [CreatePolicy](#) in [AWS SDK for JavaScript API Reference](#).

Kotlin

SDK for Kotlin

Note

This is prerelease documentation for a feature in preview release. It is subject to change.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
suspend fun createIAMPolicy(policyNameVal: String?): String {

    val policyDocumentVal = "{" +
        "  \"Version\": \"2012-10-17\", " +
        "  \"Statement\": [" +
        "    {" +
        "      \"Effect\": \"Allow\", " +
        "      \"Action\": [" +
        "        \"dynamodb>DeleteItem\", " +
        "        \"dynamodb>GetItem\", " +
        "        \"dynamodb>PutItem\", " +
        "        \"dynamodb>Scan\", " +
        "        \"dynamodb>UpdateItem\" " +
        "      ], " +
        "      \"Resource\": \"*\\" +
        "    }" +
        "  ]" +
    "}"

    val request = CreatePolicyRequest {
        policyName = policyNameVal
        policyDocument = policyDocumentVal
    }

    IamClient { region = "AWS_GLOBAL" }.use { iamClient ->
        val response = iamClient.createPolicy(request)
        return response.policy?.arn.toString()
    }
}
```

- For API details, see [CreatePolicy](#) in *AWS SDK for Kotlin API reference*.

PHP

SDK for PHP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
$uuid = uniqid();
$service = new IAMService();

$listAllBucketsPolicyDocument = "{$
    \"Version\": \"2012-10-17\",
    \"Statement\": [
        {
            \"Effect\": \"Allow\",
            \"Action\": \"s3>ListAllMyBuckets\",
            \"Resource\": \"arn:aws:s3:::*\"
        }
    ]
}";
$listAllBucketsPolicy = $service->createPolicy("iam_demo_policy_$uuid",
    $listAllBucketsPolicyDocument);
echo "Created policy: {$listAllBucketsPolicy['PolicyName']}\n";

public function createPolicy(string $policyName, string $policyDocument)
```

```

    {
        $result = $this->customWaiter(function () use ($policyName,
$policyDocument) {
            return $this->iamClient->createPolicy([
                'PolicyName' => $policyName,
                'PolicyDocument' => $policyDocument,
            ]);
        });
        return $result['Policy'];
    }
}

```

- For API details, see [CreatePolicy](#) in *AWS SDK for PHP API Reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```

def create_policy(name, description, actions, resource_arn):
    """
    Creates a policy that contains a single statement.

    :param name: The name of the policy to create.
    :param description: The description of the policy.
    :param actions: The actions allowed by the policy. These typically take the
                    form of service:action, such as s3:PutObject.
    :param resource_arn: The Amazon Resource Name (ARN) of the resource this policy
                        applies to. This ARN can contain wildcards, such as
                        'arn:aws:s3:::my-bucket/*' to allow actions on all objects
                        in the bucket named 'my-bucket'.
    :return: The newly created policy.
    """

    policy_doc = {
        "Version": "2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Action": actions,
                "Resource": resource_arn
            }
        ]
    }
    try:
        policy = iam.create_policy(
            PolicyName=name, Description=description,
            PolicyDocument=json.dumps(policy_doc))
        logger.info("Created policy %s.", policy.arn)
    except ClientError:
        logger.exception("Couldn't create policy %s.", name)
        raise
    else:
        return policy
}

```

- For API details, see [CreatePolicy](#) in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
# Creates a policy that grants permission to list S3 buckets in the account, and
# then attaches the policy to a role.
#
# @param policy_name [String] The name to give the policy.
# @param role [Aws::IAM::Role] The role that the policy is attached to.
# @return [Aws::IAM::Policy] The newly created policy.
def create_and_attach_role_policy(policy_name, role)
    policy = @iam_resource.create_policy(
        policy_name: policy_name,
        policy_document: {
            Version: "2012-10-17",
            Statement: [
                {
                    Effect: "Allow",
                    Action: "s3>ListAllMyBuckets",
                    Resource: "arn:aws:s3:::*"
                }
            ].to_json
        }.to_json
    )
    role.attach_policy(policy_arn: policy.arn)
    puts("Created policy #{policy.policy_name} and attached it to role #{role.name}.")
    rescue Aws::Errors::ServiceError => e
        puts("Couldn't create a policy and attach it to role #{role.name}. Here's why:")
    end
    else
        policy
    end
```

- For API details, see [CreatePolicy](#) in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
pub async fn create_policy(
    client: &iamClient,
    policy_name: &str,
    policy_document: &str,
) -> Result<Policy, iamError> {
    let policy = client
        .create_policy()
        .policy_name(policy_name)
        .policy_document(policy_document)
```

```
        .send()
        .await?;
    Ok(policy.policy.unwrap())
}
```

- For API details, see [CreatePolicy](#) in *AWS SDK for Rust API reference*.

Swift

SDK for Swift

Note

This is prerelease documentation for an SDK in preview release. It is subject to change.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public func createPolicy(name: String, policyDocument: String) async throws ->
IAMClientTypes.Policy {
    let input = CreatePolicyInput(
        policyDocument: policyDocument,
        policyName: name
    )
    do {
        let output = try await iamClient.createPolicy(input: input)
        guard let policy = output.policy else {
            throw ServiceHandlerError.noSuchPolicy
        }
        return policy
    } catch {
        throw error
    }
}
```

- For API details, see [CreatePolicy](#) in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Create an IAM policy version using an AWS SDK

The following code example shows how to create an IAM policy version.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Manage policies \(p. 982\)](#)

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
def create_policy_version(policy_arn, actions, resource_arn, set_as_default):
    """
    Creates a policy version. Policies can have up to five versions. The default
    version is the one that is used for all resources that reference the policy.

    :param policy_arn: The ARN of the policy.
    :param actions: The actions to allow in the policy version.
    :param resource_arn: The ARN of the resource this policy version applies to.
    :param set_as_default: When True, this policy version is set as the default
                          version for the policy. Otherwise, the default
                          is not changed.
    :return: The newly created policy version.
    """

    policy_doc = {
        'Version': '2012-10-17',
        'Statement': [
            {
                'Effect': 'Allow',
                'Action': actions,
                'Resource': resource_arn
            }
        ]
    }
    try:
        policy = iam.Policy(policy_arn)
        policy_version = policy.create_version(
            PolicyDocument=json.dumps(policy_doc), SetAsDefault=set_as_default)
        logger.info(
            "Created policy version %s for policy %s.",
            policy_version.version_id, policy_version.arn)
    except ClientError:
        logger.exception("Couldn't create a policy version for %s.", policy_arn)
        raise
    else:
        return policy_version
```

- For API details, see [CreatePolicyVersion](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Create an IAM role using an AWS SDK

The following code examples show how to create an IAM role.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- [Create a group and add a user \(p. 874\)](#)
- [Create a user and assume a role \(p. 891\)](#)
- [Manage roles \(p. 986\)](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Create a new IAM role.
/// </summary>
/// <param name="roleName">The name of the IAM role.</param>
/// <param name="rolePolicyDocument">The name of the IAM policy document
/// for the new role.</param>
/// <returns>The Amazon Resource Name (ARN) of the role.</returns>
public async Task<string> CreateRoleAsync(string roleName, string
rolePolicyDocument)
{
    var request = new CreateRoleRequest
    {
        RoleName = roleName,
        AssumeRolePolicyDocument = rolePolicyDocument,
    };

    var response = await _IAMService.CreateRoleAsync(request);
    return response.Role.Arn;
}
```

- For API details, see [CreateRole](#) in *AWS SDK for .NET API Reference*.

Bash

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
    printf "%s\n" "$*" 2>&1
}

#####
# function iam_create_role
#
# This function creates an IAM role.
#
# Parameters:
#     -n role_name -- The name of the IAM role.
#     -p policy_json -- The assume role policy document.
#
# Returns:
#     The ARN of the role.
#     And:
#         0 - If successful.
```

```

#      1 - If it fails.
#####
function iam_create_role() {
    local role_name policy_document response
    local option OPTARG # Required to use getopts command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_create_user_access_key"
        echo "Creates an AWS Identity and Access Management (IAM) role."
        echo " -n role_name   The name of the IAM role."
        echo " -p policy_json -- The assume role policy document."
        echo ""
    }

    # Retrieve the calling parameters.
    while getopts "n:p:h" option; do
        case "${option}" in
            n) role_name="${OPTARG}" ;;
            p) policy_document="${OPTARG}" ;;
            h)
                usage
                return 0
                ;;
            \?)
                echo "Invalid parameter"
                usage
                return 1
                ;;
        esac
    done
    export OPTIND=1

    if [[ -z "$role_name" ]]; then
        errecho "ERROR: You must provide a role name with the -n parameter."
        usage
        return 1
    fi

    if [[ -z "$policy_document" ]]; then
        errecho "ERROR: You must provide a policy document with the -p parameter."
        usage
        return 1
    fi

    response=$(aws iam create-role \
        --role-name "$role_name" \
        --assume-role-policy-document "$policy_document" \
        --output text \
        --query Role.Arn)

    local error_code=${?}

    if [[ $error_code -ne 0 ]]; then
        aws_cli_error_log $error_code
        errecho "ERROR: AWS reports create-role operation failed.\n$response"
        return 1
    fi

    echo "$response"

    return 0
}

```

- For API details, see [CreateRole](#) in *AWS CLI Command Reference*.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
bool AwsDoc::IAM::createIamRole(
    const Aws::String &roleName,
    const Aws::String &policy,
    const Aws::Client::ClientConfiguration &clientConfig) {
    Aws::IAM::IAMClient client(clientConfig);
    Aws::IAM::Model::CreateRoleRequest request;

    request.SetRoleName(roleName);
    request.SetAssumeRolePolicyDocument(policy);

    Aws::IAM::Model::CreateRoleOutcome outcome = client.CreateRole(request);
    if (!outcome.IsSuccess()) {
        std::cerr << "Error creating role. " <<
            outcome.GetError().GetMessage() << std::endl;
    }
    else {
        const Aws::IAM::Model::Role iamRole = outcome.GetResult().GetRole();
        std::cout << "Created role " << iamRole.GetRoleName() << "\n";
        std::cout << "ID: " << iamRole.GetRoleId() << "\n";
        std::cout << "ARN: " << iamRole.GetArn() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- For API details, see [CreateRole](#) in *AWS SDK for C++ API Reference*.

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
// It contains an IAM service client that is used to perform role actions.
type RoleWrapper struct {
    IamClient *iam.Client
}

// CreateRole creates a role that trusts a specified user. The trusted user can
// assume
// the role to acquire its permissions.
// PolicyDocument shows how to work with a policy document as a data structure and
```

```
// serialize it to JSON by using Go's JSON marshaler.
func (wrapper RoleWrapper) CreateRole(roleName string, trustedUserArn string)
(*types.Role, error) {
var role *types.Role
trustPolicy := PolicyDocument{
Version: "2012-10-17",
Statement: []PolicyStatement{
Effect: "Allow",
Principal: map[string]string{"AWS": trustedUserArn},
Action: []string{"sts:AssumeRole"},
}},
}
policyBytes, err := json.Marshal(trustPolicy)
if err != nil {
log.Printf("Couldn't create trust policy for %v. Here's why: %v\n",
trustedUserArn, err)
return nil, err
}
result, err := wrapper.IamClient.CreateRole(context.TODO(), &iam.CreateRoleInput{
AssumeRolePolicyDocument: aws.String(string(policyBytes)),
RoleName: aws.String(roleName),
})
if err != nil {
log.Printf("Couldn't create role %v. Here's why: %v\n", roleName, err)
} else {
role = result.Role
}
return role, err
}
```

- For API details, see [CreateRole](#) in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public static String createIAMRole(IamClient iam, String rolename, String
fileLocation ) throws Exception {

try {
JSONObject jsonObject = (JSONObject) readJsonSimpleDemo(fileLocation);
CreateRoleRequest request = CreateRoleRequest.builder()
.roleName(rolename)
.assumeRolePolicyDocument(jsonObject.toJSONString())
.description("Created using the AWS SDK for Java")
.build();

CreateRoleResponse response = iam.createRole(request);
System.out.println("The ARN of the role is "+response.role().arn());

} catch (IamException e) {
System.err.println(e.awsErrorDetails().errorMessage());
System.exit(1);
}
return "";
}
```

```
public static Object readJsonSimpleDemo(String filename) throws Exception {
    FileReader reader = new FileReader(filename);
    JSONParser jsonParser = new JSONParser();
    return jsonParser.parse(reader);
}
```

- For API details, see [CreateRole](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Create the role.

```
import { CreateRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 */
export const createRole = (roleName) => {
    const command = new CreateRoleCommand({
        AssumeRolePolicyDocument: JSON.stringify({
            Version: "2012-10-17",
            Statement: [
                {
                    Effect: "Allow",
                    Principal: {
                        Service: "lambda.amazonaws.com",
                    },
                    Action: "sts:AssumeRole",
                },
            ],
        }),
        RoleName: roleName,
    });

    return client.send(command);
};
```

- For API details, see [CreateRole](#) in *AWS SDK for JavaScript API Reference*.

PHP

SDK for PHP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
$uuid = uniqid();
$service = new IAMService();
```

```
$assumeRolePolicyDocument = "{
    \"Version\": \"2012-10-17\",
    \"Statement\": [
        {
            \"Effect\": \"Allow\",
            \"Principal\": {\"AWS\": \"{$user['Arn']}\"},
            \"Action\": \"sts:AssumeRole\"
        }
    ];
$assumeRoleRole = $service->createRole("iam_demo_role_$uuid",
    $assumeRolePolicyDocument);
echo "Created role: {$assumeRoleRole['RoleName']}\n";

/**
 * @param string $roleName
 * @param string $rolePolicyDocument
 * @return array
 * @throws AwsException
 */
public function createRole(string $roleName, string $rolePolicyDocument)
{
    $result = $this->customWaiter(function () use ($roleName,
$rolePolicyDocument) {
        return $this->iامClient->createRole([
            'AssumeRolePolicyDocument' => $rolePolicyDocument,
            'RoleName' => $roleName,
        ]);
    });
    return $result['Role'];
}
```

- For API details, see [CreateRole](#) in *AWS SDK for PHP API Reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
def create_role(role_name, allowed_services):
    """
    Creates a role that lets a list of specified services assume the role.

    :param role_name: The name of the role.
    :param allowed_services: The services that can assume the role.
    :return: The newly created role.
    """
    trust_policy = {
        'Version': '2012-10-17',
        'Statement': [
            {
                'Effect': 'Allow',
                'Principal': {'Service': service},
                'Action': 'sts:AssumeRole'
            } for service in allowed_services
        ]
    }

    try:
        role = iam.create_role(
```

```
    RoleName=role_name,
    AssumeRolePolicyDocument=json.dumps(trust_policy))
    logger.info("Created role %s.", role.name)
except ClientError:
    logger.exception("Couldn't create role %s.", role_name)
    raise
else:
    return role
```

- For API details, see [CreateRole](#) in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
# Creates a role that can be assumed by a user.
#
# @param role_name [String] The name to give the role.
# @param user [Aws::IAM::User] The user who is granted permission to assume the
#   role.
# @return [Aws::IAM::Role] The newly created role.
def create_role(role_name, user)
  role = @iam_resource.create_role(
    role_name: role_name,
    assume_role_policy_document: {
      Version: "2012-10-17",
      Statement: [
        {
          Effect: "Allow",
          Principal: {'AWS': user.arn},
          Action: "sts:AssumeRole"
        }
      ].to_json)
  puts("Created role #{role.name}.")
rescue Aws::Errors::ServiceError => e
  puts("Couldn't create a role for the demo. Here's why: ")
  puts("\t#{e.code}: #{e.message}")
  raise
else
  role
end
```

- For API details, see [CreateRole](#) in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
pub async fn create_role(
    client: &iamClient,
    role_name: &str,
    role_policy_document: &str,
) -> Result<Role, iamError> {
    let response: CreateRoleOutput = loop {
        if let Ok(response) = client
            .create_role()
            .role_name(role_name)
            .assume_role_policy_document(role_policy_document)
            .send()
            .await
        {
            break response;
        }
    };

    Ok(response.role.unwrap())
}
```

- For API details, see [CreateRole](#) in *AWS SDK for Rust API reference*.

Swift

SDK for Swift

Note

This is prerelease documentation for an SDK in preview release. It is subject to change.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public func createRole(name: String, policyDocument: String) async throws -> String {
    let input = CreateRoleInput(
        assumeRolePolicyDocument: policyDocument,
        roleName: name
    )
    do {
        let output = try await client.createRole(input: input)
        guard let role = output.role else {
            throw ServiceHandlerError.noSuchRole
        }
        guard let id = role.roleId else {
            throw ServiceHandlerError.noSuchRole
        }
        return id
    } catch {
        throw error
    }
}
```

- For API details, see [CreateRole](#) in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Create an IAM service-linked role using an AWS SDK

The following code examples show how to create an IAM service-linked role.

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Create an IAM service-linked role.
/// </summary>
/// <param name="serviceName">The name of the AWS Service.</param>
/// <param name="description">A description of the IAM service-linked role.</param>
/// <returns>The IAM role that was created.</returns>
public async Task<Role> CreateServiceLinkedRoleAsync(string serviceName, string
description)
{
    var request = new CreateServiceLinkedRoleRequest
    {
        AWSServiceName = serviceName,
        Description = description
    };

    var response = await _IAMService.CreateServiceLinkedRoleAsync(request);
    return response.Role;
}
```

- For API details, see [CreateServiceLinkedRole](#) in *AWS SDK for .NET API Reference*.

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
// It contains an IAM service client that is used to perform role actions.
type RoleWrapper struct {
    IamClient *iam.Client
}

// CreateServiceLinkedRole creates a service-linked role that is owned by the
// specified service.
func (wrapper RoleWrapper) CreateServiceLinkedRole(serviceName string, description
string) (*types.Role, error) {
    var role *types.Role
```

```
result, err := wrapper.IamClient.CreateServiceLinkedRole(context.TODO(),
&iam.CreateServiceLinkedRoleInput{
    AWSServiceName: aws.String(serviceName),
    Description:   aws.String(description),
})
if err != nil {
    log.Printf("Couldn't create service-linked role %v. Here's why: %v\n",
    serviceName, err)
} else {
    role = result.Role
}
return role, err
}
```

- For API details, see [CreateServiceLinkedRole](#) in *AWS SDK for Go API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Create a service-linked role.

```
import { CreateServiceLinkedRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} serviceName
 */
export const createServiceLinkedRole = async (serviceName) => {
    const command = new CreateServiceLinkedRoleCommand({
        // For a list of AWS services that support service-linked roles,
        // see https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_aws-services-that-work-with-iam.html.
        //
        // For a list of AWS service endpoints, see https://docs.aws.amazon.com/general/latest/gr/aws-service-information.html.
        AWSServiceName: serviceName,
    });

    const response = await client.send(command);
    console.log(response);
    return response;
};
```

- For API details, see [CreateServiceLinkedRole](#) in *AWS SDK for JavaScript API Reference*.

PHP

SDK for PHP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
$uuid = uniqid();
$service = new IAMService();

    public function createServiceLinkedRole($awsServiceName, $customSuffix = "", $description = "") {
        $createServiceLinkedRoleArguments = ['AWSServiceName' => $awsServiceName];
        if ($customSuffix) {
            $createServiceLinkedRoleArguments['CustomSuffix'] = $customSuffix;
        }
        if ($description) {
            $createServiceLinkedRoleArguments['Description'] = $description;
        }
        return $this->iamClient->createServiceLinkedRole($createServiceLinkedRoleArguments);
    }
}
```

- For API details, see [CreateServiceLinkedRole](#) in *AWS SDK for PHP API Reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
def create_service_linked_role(service_name, description):
    """
    Creates a service-linked role.

    :param service_name: The name of the service that owns the role.
    :param description: A description to give the role.
    :return: The newly created role.
    """
    try:
        response = iam.meta.client.create_service_linked_role(
            AWSServiceName=service_name, Description=description)
        role = iam.Role(response['Role']['RoleName'])
        logger.info("Created service-linked role %s.", role.name)
    except ClientError:
        logger.exception("Couldn't create service-linked role for %s.",
                         service_name)
        raise
    else:
        return role
```

- For API details, see [CreateServiceLinkedRole](#) in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
# Creates a service-linked role.  
#  
# @param service_name [String] The name of the service that owns the role.  
# @param description [String] A description to give the role.  
# @return [Aws::IAM::Role] The newly created role.  
def create_service_linked_role(service_name, description)  
    response = @iam_resource.client.create_service_linked_role(  
        aws_service_name: service_name, description: description)  
    role = @iam_resource.role(response.role.role_name)  
    puts("Created service-linked role #{role.name}.")  
rescue Aws::Errors::ServiceError => e  
    puts("Couldn't create service-linked role for #{service_name}. Here's why:")  
    puts("\t#{e.code}: #{e.message}")  
    raise  
else  
    role  
end
```

- For API details, see [CreateServiceLinkedRole](#) in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
pub async fn create_service_linked_role(  
    client: &iامClient,  
    aws_service_name: String,  
    custom_suffix: Option<String>,  
    description: Option<String>,  
) -> Result<CreateServiceLinkedRoleOutput, SdkError<CreateServiceLinkedRoleError>>  
{  
    let response = client  
        .create_service_linked_role()  
        .aws_service_name(aws_service_name)  
        .set_custom_suffix(custom_suffix)  
        .set_description(description)  
        .send()  
        .await?  
  
    Ok(response)  
}
```

- For API details, see [CreateServiceLinkedRole](#) in *AWS SDK for Rust API reference*.

Swift

SDK for Swift

Note

This is prerelease documentation for an SDK in preview release. It is subject to change.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public func createServiceLinkedRole(service: String, suffix: String? = nil,  
description: String?)  
    async throws -> IAMClientTypes.Role {  
    let input = CreateServiceLinkedRoleInput(  
        awsServiceName: service,  
        customSuffix: suffix,  
        description: description  
    )  
    do {  
        let output = try await client.createServiceLinkedRole(input: input)  
        guard let role = output.role else {  
            throw ServiceHandlerError.noSuchRole  
        }  
        return role  
    } catch {  
        throw error  
    }  
}
```

- For API details, see [CreateServiceLinkedRole](#) in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Create an IAM user using an AWS SDK

The following code examples show how to create an IAM user.

Warning

To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as [AWS IAM Identity Center \(successor to AWS Single Sign-On\)](#).

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- [Create a group and add a user \(p. 874\)](#)
- [Create a user and assume a role \(p. 891\)](#)
- [Create read-only and read-write users \(p. 973\)](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
```

```
/// Create an IAM user.
/// </summary>
/// <param name="userName">The username for the new IAM user.</param>
/// <returns>The IAM user that was created.</returns>
public async Task<User> CreateUserAsync(string userName)
{
    var response = await _IAMService.CreateUserAsync(new CreateUserRequest
{ UserName = userName });
    return response.User;
}
```

- For API details, see [CreateUser](#) in *AWS SDK for .NET API Reference*.

Bash

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
#####
function iecho() {
    if [[ $VERBOSE == true ]]; then
        echo "$@"
    fi
}

#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
    printf "%s\n" "$*" 2>&1
}

#####
# function iam_create_user
#
# This function creates the specified IAM user, unless
# it already exists.
#
# Parameters:
#     -u user_name -- The name of the user to create.
#
# Returns:
#     The ARN of the user.
#     And:
#         0 - If successful.
#         1 - If it fails.
#####
function iam_create_user() {
    local user_name response
    local option OPTARG # Required to use getopt command in a function.
```

```

# bashsupport disable=BP5008
function usage() {
    echo "function iam_create_user"
    echo "Creates an AWS Identity and Access Management (IAM) user. You must supply"
    echo "a username:"
    echo "    -u user_name      The name of the user. It must be unique within the"
    echo "account."
    echo ""
}

# Retrieve the calling parameters.
while getopts "u:h" option; do
    case "${option}" in
        u) user_name="${OPTARG}" ;;
        h)
            usage
            return 0
            ;;
        \?)
            echo "Invalid parameter"
            usage
            return 1
            ;;
    esac
done
export OPTIND=1

if [[ -z "$user_name" ]]; then
    errecho "ERROR: You must provide a username with the -u parameter."
    usage
    return 1
fi

iecho "Parameters:\n"
iecho "    User name: $user_name"
iecho ""

# If the user already exists, we don't want to try to create it.
if (iam_user_exists "$user_name"); then
    errecho "ERROR: A user with that name already exists in the account."
    return 1
fi

response=$(aws iam create-user --user-name "$user_name" \
    --output text \
    --query 'User.Arn')

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports create-user operation failed.$response"
    return 1
fi

echo "$response"

return 0
}

```

- For API details, see [CreateUser](#) in *AWS CLI Command Reference*.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
Aws::IAM::IAMClient iam(clientConfig);

Aws::IAM::Model::CreateUserRequest create_request;
create_request.SetUserName(userName);

auto create_outcome = iam.CreateUser(create_request);
if (!create_outcome.IsSuccess()) {
    std::cerr << "Error creating IAM user " << userName << ":" <<
        create_outcome.GetError().GetMessage() << std::endl;
}
else {
    std::cout << "Successfully created IAM user " << userName << std::endl;
}

return create_outcome.IsSuccess();
```

- For API details, see [CreateUser](#) in *AWS SDK for C++ API Reference*.

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// UserWrapper encapsulates user actions used in the examples.
// It contains an IAM service client that is used to perform user actions.
type UserWrapper struct {
    IamClient *iam.Client
}

// CreateUser creates a new user with the specified name.
func (wrapper UserWrapper) CreateUser(userName string) (*types.User, error) {
    var user *types.User
    result, err := wrapper.IamClient.CreateUser(context.TODO(), &iam.CreateUserInput{
        UserName: aws.String(userName),
    })
    if err != nil {
        log.Printf("Couldn't create user %v. Here's why: %v\n", userName, err)
    } else {
        user = result.User
    }
    return user, err
}
```

- For API details, see [CreateUser](#) in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public static String createIAMUser(IamClient iam, String username) {  
  
    try {  
        // Create an IamWaiter object  
        IamWaiter iamWaiter = iam.waiter();  
  
        CreateUserRequest request = CreateUserRequest.builder()  
            .userName(username)  
            .build();  
  
        CreateUserResponse response = iam.createUser(request);  
  
        // Wait until the user is created  
        GetUserRequest userRequest = GetUserRequest.builder()  
            .userName(response.user().userName())  
            .build();  
  
        WaiterResponse< GetUserResponse> waitUntilUserExists =  
            iamWaiter.waitUntilUserExists(userRequest);  
  
        waitUntilUserExists.matched().response().ifPresent(System.out::println);  
        return response.user().userName();  
  
    } catch (IamException e) {  
        System.err.println(e.awsErrorDetails().errorMessage());  
        System.exit(1);  
    }  
    return "";  
}
```

- For API details, see [CreateUser](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Create the user.

```
import { CreateUserCommand, IAMClient } from "@aws-sdk/client-iam";  
  
const client = new IAMClient({});  
  
/**  
 *  
 * @param {string} name
```

```
/*
export const createUser = (name) => {
  const command = new CreateUserCommand({ UserName: name });
  return client.send(command);
};
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [CreateUser](#) in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// Load the AWS SDK for Node.js
var AWS = require('aws-sdk');
// Set the region
AWS.config.update({region: 'REGION'});

// Create the IAM service object
var iam = new AWS.IAM({apiVersion: '2010-05-08'});

var params = {
  UserName: process.argv[2]
};

iam.getUser(params, function(err, data) {
  if (err && err.code === 'NoSuchEntity') {
    iam.createUser(params, function(err, data) {
      if (err) {
        console.log("Error", err);
      } else {
        console.log("Success", data);
      }
    });
  } else {
    console.log("User " + process.argv[2] + " already exists", data.UserId);
  }
});
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [CreateUser](#) in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

Note

This is prerelease documentation for a feature in preview release. It is subject to change.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
suspend fun createIAMUser(usernameVal: String?): String? {
```

```
    val request = CreateUserRequest {
        userName = usernameVal
    }

    IamClient { region = "AWS_GLOBAL" }.use { iamClient ->
        val response = iamClient.createUser(request)
        return response.user?.userName
    }
}
```

- For API details, see [CreateUser](#) in *AWS SDK for Kotlin API reference*.

PHP

SDK for PHP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
$uuid = uniqid();
 getService();

$user = $service->createUser("iam_demo_user_$uuid");
echo "Created user with the arn: {$user['Arn']}\n";

/**
 * @param string $name
 * @return array
 * @throws AwsException
 */
public function createUser(string $name): array
{
    $result = $this->iamClient->createUser([
        'UserName' => $name,
    ]);

    return $result['User'];
}
```

- For API details, see [CreateUser](#) in *AWS SDK for PHP API Reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
def create_user(user_name):
    """
    Creates a user. By default, a user has no permissions or access keys.

```

```
:param user_name: The name of the user.  
:return: The newly created user.  
"""  
try:  
    user = iam.create_user(UserName=user_name)  
    logger.info("Created user %s.", user.name)  
except ClientError:  
    logger.exception("Couldn't create user %s.", user_name)  
    raise  
else:  
    return user
```

- For API details, see [CreateUser](#) in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
# Creates a user.  
#  
# @param user_name [String] The name to give the user.  
# @return [Aws::IAM::User] The newly created user.  
def create_user(user_name)  
    user = @iam_resource.create_user(user_name: user_name)  
    puts("Created demo user named #{user.name}.")  
rescue Aws::Errors::ServiceError => e  
    puts("Tried and failed to create demo user.")  
    puts("\t#{e.code}: #{e.message}")  
    puts("\nCan't continue the demo without a user!")  
    raise  
else  
    user  
end
```

- For API details, see [CreateUser](#) in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
pub async fn create_user(client: &iamClient, user_name: &str) -> Result<User, iamError> {  
    let response = client.create_user().user_name(user_name).send().await?;
```

```
        Ok(response.user.unwrap())
    }
```

- For API details, see [CreateUser](#) in *AWS SDK for Rust API reference*.

Swift

SDK for Swift

Note

This is prerelease documentation for an SDK in preview release. It is subject to change.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public func createUser(name: String) async throws -> String {
    let input = CreateUserInput(
        userName: name
    )
    do {
        let output = try await client.createUser(input: input)
        guard let user = output.user else {
            throw ServiceHandlerError.noSuchUser
        }
        guard let id = user.userId else {
            throw ServiceHandlerError.noSuchUser
        }
        return id
    } catch {
        throw error
    }
}
```

- For API details, see [CreateUser](#) in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Create an IAM access key using an AWS SDK

The following code examples show how to create an IAM access key.

Warning

To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as [AWS IAM Identity Center \(successor to AWS Single Sign-On\)](#).

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- [Create a group and add a user \(p. 874\)](#)
- [Create a user and assume a role \(p. 891\)](#)
- [Create read-only and read-write users \(p. 973\)](#)

- [Manage access keys \(p. 980\)](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Create an IAM access key for a user.
/// </summary>
/// <param name="userName">The username for which to create the IAM access
/// key.</param>
/// <returns>The AccessKey.</returns>
public async Task<AccessKey> CreateAccessKeyAsync(string userName)
{
    var response = await _IAMService.CreateAccessKeyAsync(new
CreateAccessKeyRequest
    {
        UserName = userName,
    });

    return response.AccessKey;
}
```

- For API details, see [CreateAccessKey](#) in *AWS SDK for .NET API Reference*.

Bash

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
    printf "%s\n" "$*" 2>&1
}

#####
# function iam_create_user_access_key
#
# This function creates an IAM access key for the specified user.
#
# Parameters:
#     -u user_name -- The name of the IAM user.
#     [-f file_name] -- The optional file name for the access key output.
#
# Returns:
#     [access_key_id access_key_secret]
```

```

#      And:
#          0 - If successful.
#          1 - If it fails.
#####
function iam_create_user_access_key() {
    local user_name file_name response
    local option OPTARG # Required to use getopts command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_create_user_access_key"
        echo "Creates an AWS Identity and Access Management (IAM) key pair."
        echo " -u user_name   The name of the IAM user."
        echo " [-f file_name]  Optional file name for the access key output."
        echo ""
    }

    # Retrieve the calling parameters.
    while getopts "u:f:h" option; do
        case "${option}" in
            u) user_name="${OPTARG}" ;;
            f) file_name="${OPTARG}" ;;
            h)
                usage
                return 0
                ;;
            \?)
                echo "Invalid parameter"
                usage
                return 1
                ;;
        esac
    done
    export OPTIND=1

    if [[ -z "$user_name" ]]; then
        errecho "ERROR: You must provide a username with the -u parameter."
        usage
        return 1
    fi

    response=$(aws iam create-access-key \
        --user-name "$user_name" \
        --output text)

    local error_code=${?}

    if [[ $error_code -ne 0 ]]; then
        aws_cli_error_log $error_code
        errecho "ERROR: AWS reports create-access-key operation failed.$response"
        return 1
    fi

    if [[ -n "$file_name" ]]; then
        echo "$response" >"$file_name"
    fi

    local key_id key_secret
    # shellcheck disable=SC2086
    key_id=$(echo $response | cut -f 2 -d ' ')
    # shellcheck disable=SC2086
    key_secret=$(echo $response | cut -f 4 -d ' ')

    echo "$key_id $key_secret"

    return 0
}

```

```
}
```

- For API details, see [CreateAccessKey](#) in *AWS CLI Command Reference*.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
Aws::String AwsDoc::IAM::createAccessKey(const Aws::String &userName,
                                         const Aws::Client::ClientConfiguration
&clientConfig) {
    Aws::IAM::IAMClient iam(clientConfig);

    Aws::IAM::Model::CreateAccessKeyRequest request;
    request.SetUserName(userName);

    Aws::String result;
    Aws::IAM::Model::CreateAccessKeyOutcome outcome = iam.CreateAccessKey(request);
    if (!outcome.IsSuccess()) {
        std::cerr << "Error creating access key for IAM user " << userName
              << ":" << outcome.GetError().GetMessage() << std::endl;
    }
    else {
        const auto &accessKey = outcome.GetResult().GetAccessKey();
        std::cout << "Successfully created access key for IAM user " <<
            userName << std::endl << " aws_access_key_id = " <<
            accessKey.GetAccessKeyId() << std::endl <<
            " aws_secret_access_key = " << accessKey.GetSecretAccessKey() <<
            std::endl;
        result = accessKey.GetAccessKeyId();
    }
    return result;
}
```

- For API details, see [CreateAccessKey](#) in *AWS SDK for C++ API Reference*.

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// UserWrapper encapsulates user actions used in the examples.
// It contains an IAM service client that is used to perform user actions.
type UserWrapper struct {
    IamClient *iam.Client
}
```

```
// CreateAccessKeyPair creates an access key for a user. The returned access key
// contains
// the ID and secret credentials needed to use the key.
func (wrapper UserWrapper) CreateAccessKeyPair(userName string) (*types.AccessKey,
error) {
var key *types.AccessKey
result, err := wrapper.IamClient.CreateAccessKey(context.TODO(),
&iam.CreateAccessKeyInput{
UserName: aws.String(userName)})
if err != nil {
log.Printf("Couldn't create access key pair for user %v. Here's why: %v\n",
userName, err)
} else {
key = result.AccessKey
}
return key, err
}
```

- For API details, see [CreateAccessKey](#) in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public static String createIAMAccessKey(IamClient iam, String user) {
    try {
        CreateAccessKeyRequest request = CreateAccessKeyRequest.builder()
            .userName(user)
            .build();

        CreateAccessKeyResponse response = iam.createAccessKey(request);
        return response.accessKey().accessKeyId();

    } catch (IamException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
    return "";
}
```

- For API details, see [CreateAccessKey](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Create the access key.

```
import { CreateAccessKeyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} userName
 */
export const createAccessKey = (userName) => {
  const command = new CreateAccessKeyCommand({ UserName: userName });
  return client.send(command);
};
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [CreateAccessKey](#) in [AWS SDK for JavaScript API Reference](#).

SDK for JavaScript (v2)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// Load the AWS SDK for Node.js
var AWS = require('aws-sdk');
// Set the region
AWS.config.update({region: 'REGION'});

// Create the IAM service object
var iam = new AWS.IAM({apiVersion: '2010-05-08'});

iam.createAccessKey({UserName: 'IAM_USER_NAME'}, function(err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.AccessKey);
  }
});
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [CreateAccessKey](#) in [AWS SDK for JavaScript API Reference](#).

Kotlin

SDK for Kotlin

Note

This is prerelease documentation for a feature in preview release. It is subject to change.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
suspend fun createIAMAccessKey(user: String?): String {

    val request = CreateAccessKeyRequest {
        userName = user
    }
```

```
        }

        IamClient { region = "AWS_GLOBAL" }.use { iamClient ->
            val response = iamClient.createAccessKey(request)
            return response.accessKey?.accessKeyId.toString()
        }
    }
```

- For API details, see [CreateAccessKey](#) in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
def create_key(user_name):
    """
    Creates an access key for the specified user. Each user can have a
    maximum of two keys.

    :param user_name: The name of the user.
    :return: The created access key.
    """
    try:
        key_pair = iam.User(user_name).create_access_key_pair()
        logger.info(
            "Created access key pair for %s. Key ID is %s.",
            key_pair.user_name, key_pair.id)
    except ClientError:
        logger.exception("Couldn't create access key pair for %s.", user_name)
        raise
    else:
        return key_pair
```

- For API details, see [CreateAccessKey](#) in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
# Creates an access key for a user.
#
# @param user [Aws::IAM::User] The user that owns the key.
# @return [Aws::IAM::AccessKeyPair] The newly created access key.
def create_access_key_pair(user)
    user_key = user.create_access_key_pair
    puts("Created access key pair for user.")
rescue Aws::Errors::ServiceError => e
    puts("Couldn't create access keys for user #{user.name}.")
    puts("\t#{e.code}: #{e.message}")
```

```
    raise
else
  user_key
end
```

- For API details, see [CreateAccessKey](#) in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
pub async fn create_access_key(client: &iamClient, user_name: &str) ->
Result<AccessKey, iamError> {
let mut tries: i32 = 0;
let max_tries: i32 = 10;

let response: Result<CreateAccessKeyOutput, SdkError<CreateAccessKeyError>> =
loop {
    match client.create_access_key().user_name(user_name).send().await {
        Ok(inner_response) => {
            break Ok(inner_response);
        }
        Err(e) => {
            tries += 1;
            if tries > max_tries {
                break Err(e);
            }
            sleep(Duration::from_secs(2)).await;
        }
    }
};

Ok(response.unwrap().access_key.unwrap())
}
```

- For API details, see [CreateAccessKey](#) in *AWS SDK for Rust API reference*.

Swift

SDK for Swift

Note

This is prerelease documentation for an SDK in preview release. It is subject to change.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public func createAccessKey(userName: String) async throws ->
IAMClientTypes.AccessKey {
```

```
let input = CreateAccessKeyInput(  
    userName: userName  
)  
do {  
    let output = try await iamClient.createAccessKey(input: input)  
    guard let accessKey = output.accessKey else {  
        throw ServiceHandlerError.keyError  
    }  
    return accessKey  
} catch {  
    throw error  
}  
}
```

- For API details, see [CreateAccessKey](#) in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Create an alias for an IAM account using an AWS SDK

The following code examples show how to create an alias for an IAM account.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Manage your account \(p. 989\)](#)

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
bool AwsDoc::IAM::createAccountAlias(const Aws::String &aliasName,  
                                      const Aws::Client::ClientConfiguration  
&clientConfig) {  
    Aws::IAM::IAMClient iam(clientConfig);  
    Aws::IAM::Model::CreateAccountAliasRequest request;  
    request.SetAccountAlias(aliasName);  
  
    Aws::IAM::Model::CreateAccountAliasOutcome outcome = iam.CreateAccountAlias(  
        request);  
    if (!outcome.IsSuccess()) {  
        std::cerr << "Error creating account alias " << aliasName << ":"  
              << outcome.GetError().GetMessage() << std::endl;  
    }  
    else {  
        std::cout << "Successfully created account alias " << aliasName <<  
              std::endl;  
    }  
  
    return outcome.IsSuccess();  
}
```

- For API details, see [CreateAccountAlias](#) in *AWS SDK for C++ API Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public static void createIAMAccountAlias(IamClient iam, String alias) {  
  
    try {  
        CreateAccountAliasRequest request = CreateAccountAliasRequest.builder()  
            .accountAlias(alias)  
            .build();  
  
        iam.createAccountAlias(request);  
        System.out.println("Successfully created account alias: " + alias);  
  
    } catch (IamException e) {  
        System.err.println(e.awsErrorDetails().errorMessage());  
        System.exit(1);  
    }  
}
```

- For API details, see [CreateAccountAlias](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Create the account alias.

```
import { CreateAccountAliasCommand, IAMClient } from "@aws-sdk/client-iam";  
  
const client = new IAMClient({});  
  
/**  
 *  
 * @param {string} alias - A unique name for the account alias.  
 * @returns  
 */  
export const createAccountAlias = (alias) => {  
    const command = new CreateAccountAliasCommand({  
        AccountAlias: alias,  
    });  
  
    return client.send(command);  
};
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [CreateAccountAlias](#) in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// Load the AWS SDK for Node.js
var AWS = require('aws-sdk');
// Set the region
AWS.config.update({region: 'REGION'});

// Create the IAM service object
var iam = new AWS.IAM({apiVersion: '2010-05-08'});

iam.createAccountAlias({AccountAlias: process.argv[2]}, function(err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [CreateAccountAlias](#) in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

Note

This is prerelease documentation for a feature in preview release. It is subject to change.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
suspend fun createIAMAccountAlias(alias: String) {

    val request = CreateAccountAliasRequest {
        accountAlias = alias
    }

    IamClient { region = "AWS_GLOBAL" }.use { iamClient ->
        iamClient.createAccountAlias(request)
        println("Successfully created account alias named $alias")
    }
}
```

- For API details, see [CreateAccountAlias](#) in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
def create_alias(alias):
    """
    Creates an alias for the current account. The alias can be used in place of the
    account ID in the sign-in URL. An account can have only one alias. When a new
    alias is created, it replaces any existing alias.

    :param alias: The alias to assign to the account.
    """

    try:
        iam.create_account_alias(AccountAlias=alias)
        logger.info("Created an alias '%s' for your account.", alias)
    except ClientError:
        logger.exception("Couldn't create alias '%s' for your account.", alias)
        raise
```

- For API details, see [CreateAccountAlias](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Create an inline IAM policy for a group using an AWS SDK

The following code example shows how to create an inline IAM policy for a group.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Create a group and add a user \(p. 874\)](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Add or update an inline policy document that is embedded in an IAM group.
/// </summary>
/// <param name="groupName">The name of the IAM group.</param>
/// <param name="policyName">The name of the IAM policy.</param>
/// <param name="policyDocument">The policy document defining the IAM policy.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> PutGroupPolicyAsync(string groupName, string
policyName, string policyDocument)
{
    var request = new PutGroupPolicyRequest
    {
        GroupName = groupName,
        PolicyName = policyName,
        PolicyDocument = policyDocument
    };
}
```

```
        var response = await _IAMService.PutGroupPolicyAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- For API details, see [PutGroupPolicy](#) in *AWS SDK for .NET API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Create an inline IAM policy for a user using an AWS SDK

The following code examples show how to create an inline IAM policy for a user.

Warning

To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as [AWS IAM Identity Center \(successor to AWS Single Sign-On\)](#).

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Create a user and assume a role \(p. 891\)](#)

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// UserWrapper encapsulates user actions used in the examples.
// It contains an IAM service client that is used to perform user actions.
type UserWrapper struct {
    IamClient *iam.Client
}

// CreateUserPolicy adds an inline policy to a user. This example creates a policy
// that
// grants a list of actions on a specified role.
// PolicyDocument shows how to work with a policy document as a data structure and
// serialize it to JSON by using Go's JSON marshaler.
func (wrapper UserWrapper) CreateUserPolicy(userName string, policyName string,
    actions []string,
    roleArn string) error {
    policyDoc := PolicyDocument{
        Version: "2012-10-17",
        Statement: []PolicyStatement{{
            Effect: "Allow",
            Action: actions,
            Resource: aws.String(roleArn),
        }},
    }
}
```

```
policyBytes, err := json.Marshal(policyDoc)
if err != nil {
    log.Printf("Couldn't create policy document for %v. Here's why: %v\n", roleArn,
err)
    return err
}
_, err = wrapper.IamClient.PutUserPolicy(context.TODO(), &iam.PutUserPolicyInput{
    PolicyDocument: aws.String(string(policyBytes)),
    PolicyName:     aws.String(policyName),
    UserName:       aws.String(userName),
})
if err != nil {
    log.Printf("Couldn't create policy for user %v. Here's why: %v\n", userName, err)
}
return err
}
```

- For API details, see [PutUserPolicy](#) in *AWS SDK for Go API Reference*.

Ruby

SDK for Ruby

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
# Creates an inline policy for a user that lets the user assume a role.
#
# @param policy_name [String] The name to give the policy.
# @param user [Aws::IAM::User] The user that owns the policy.
# @param role [Aws::IAM::Role] The role that can be assumed.
# @return [Aws::IAM::UserPolicy] The newly created policy.
def create_user_policy(policy_name, user, role)
    policy = user.create_policy(
        policy_name: policy_name,
        policy_document: [
            Version: "2012-10-17",
            Statement: [
                Effect: "Allow",
                Action: "sts:AssumeRole",
                Resource: role.arn
            ]
        ].to_json)
    puts("Created an inline policy for #{user.name} that lets the user assume role
#{role.name}.")
    rescue Aws::Errors::ServiceError => e
        puts("Couldn't create an inline policy for user #{user.name}. Here's why: ")
        puts("\t#{e.code}: #{e.message}")
        raise
    else
        policy
    end

```

- For API details, see [PutUserPolicy](#) in *AWS SDK for Ruby API Reference*.

Swift

SDK for Swift

Note

This is prerelease documentation for an SDK in preview release. It is subject to change.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
func putUserPolicy(policyDocument: String, policyName: String, user: IAMClientTypes.User) async throws {
    let input = PutUserPolicyInput(
        policyDocument: policyDocument,
        policyName: policyName,
        userName: user.userName
    )
    do {
        _ = try await iamClient.putUserPolicy(input: input)
    } catch {
        throw error
    }
}
```

- For API details, see [PutUserPolicy](#) in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Delete an IAM SAML provider using an AWS SDK

The following code example shows how to delete an AWS Identity and Access Management (IAM) SAML provider.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import { DeleteSAMLProviderCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} providerArn
 * @returns
 */
export const deleteSAMLProvider = async (providerArn) => {
    const command = new DeleteSAMLProviderCommand({
        SAMLProviderArn: providerArn,
    });

    const response = await client.send(command);
```

```
    console.log(response);
    return response;
};
```

- For API details, see [DeleteSAMLProvider](#) in *AWS SDK for JavaScript API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Delete an IAM group using an AWS SDK

The following code examples show how to delete an IAM group.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Create a group and add a user \(p. 874\)](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Delete an IAM group.
/// </summary>
/// <param name="groupName">The name of the IAM group to delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteGroupAsync(string groupName)
{
    var response = await _IAMService.DeleteGroupAsync(new DeleteGroupRequest
{ GroupName = groupName });
    return response.StatusCode == HttpStatusCode.OK;
}
```

- For API details, see [DeleteGroup](#) in *AWS SDK for .NET API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import { DeleteGroupCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
```

```
*  
* @param {string} groupName  
*/  
export const deleteGroup = async (groupName) => {  
    const command = new DeleteGroupCommand({  
        GroupName: groupName,  
    });  
  
    const response = await client.send(command);  
    console.log(response);  
    return response;  
};
```

- For API details, see [DeleteGroup](#) in *AWS SDK for JavaScript API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Delete an IAM group policy using an AWS SDK

The following code example shows how to delete an IAM group policy.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Create a group and add a user \(p. 874\)](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>  
/// Delete an IAM policy associated with an IAM group.  
/// </summary>  
/// <param name="groupName">The name of the IAM group associated with the  
/// policy.</param>  
/// <param name="policyName">The name of the policy to delete.</param>  
/// <returns>A Boolean value indicating the success of the action.</returns>  
public async Task<bool> DeleteGroupPolicyAsync(string groupName, string  
policyName)  
{  
    var request = new DeleteGroupPolicyRequest()  
    {  
        GroupName = groupName,  
        PolicyName = policyName,  
    };  
  
    var response = await _IAMService.DeleteGroupPolicyAsync(request);  
    return response.HttpStatusCode == System.Net HttpStatusCode.OK;  
}
```

- For API details, see [DeleteGroupPolicy](#) in *AWS SDK for .NET API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Delete an IAM policy using an AWS SDK

The following code examples show how to delete an IAM policy.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- [Create a user and assume a role \(p. 891\)](#)
- [Create read-only and read-write users \(p. 973\)](#)
- [Manage policies \(p. 982\)](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Delete an IAM policy.
/// </summary>
/// <param name="policyArn">The Amazon Resource Name (ARN) of the policy to
/// delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeletePolicyAsync(string policyArn)
{
    var response = await _IAMService.DeletePolicyAsync(new DeletePolicyRequest
{ PolicyArn = policyArn });
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- For API details, see [DeletePolicy](#) in *AWS SDK for .NET API Reference*.

Bash

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
#####
function iecho() {
    if [[ $VERBOSE == true ]]; then
        echo "$@"
    fi
}
```

```

}

#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
    printf "%s\n" "$*" 2>&1
}

#####
# function iam_delete_policy
#
# This function deletes an IAM policy.
#
# Parameters:
#     -n policy_arn -- The name of the IAM policy arn.
#
# Returns:
#     0 - If successful.
#     1 - If it fails.
#####
function iam_delete_policy() {
    local policy_arn response
    local option OPTARG # Required to use getopt command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_delete_policy"
        echo "Deletes an AWS Identity and Access Management (IAM) policy"
        echo " -n policy_arn -- The name of the IAM policy arn."
        echo ""
    }

    # Retrieve the calling parameters.
    while getopt "n:h" option; do
        case "${option}" in
            n) policy_arn="${OPTARG}" ;;
            h)
                usage
                return 0
                ;;
            \?)
                echo "Invalid parameter"
                usage
                return 1
                ;;
        esac
    done
    export OPTIND=1

    if [[ -z "$policy_arn" ]]; then
        errecho "ERROR: You must provide a policy arn with the -n parameter."
        usage
        return 1
    fi

    iecho "Parameters:\n"
    iecho "    Policy arn: $policy_arn"
    iecho ""

    response=$(aws iam delete-policy \
        --policy-arn "$policy_arn")

    local error_code=${?}
}

```

```
if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-policy operation failed.\n$response"
    return 1
fi

iecho "delete-policy response:$response"
iecho

return 0
}
```

- For API details, see [DeletePolicy](#) in *AWS CLI Command Reference*.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
bool AwsDoc::IAM::deletePolicy(const Aws::String &policyArn,
                                const Aws::Client::ClientConfiguration
&clientConfig) {
    Aws::IAM::IAMClient iam(clientConfig);
    Aws::IAM::Model::DeletePolicyRequest request;
    request.SetPolicyArn(policyArn);

    auto outcome = iam.DeletePolicy(request);
    if (!outcome.IsSuccess()) {
        std::cerr << "Error deleting policy with arn " << policyArn << ":" 
              << outcome.GetError().GetMessage() << std::endl;
    }
    else {
        std::cout << "Successfully deleted policy with arn " << policyArn
              << std::endl;
    }

    return outcome.IsSuccess();
}
```

- For API details, see [DeletePolicy](#) in *AWS SDK for C++ API Reference*.

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// PolicyWrapper encapsulates AWS Identity and Access Management (IAM) policy
// actions
// used in the examples.
// It contains an IAM service client that is used to perform policy actions.
```

```
type PolicyWrapper struct {
    IamClient *iam.Client
}

// DeletePolicy deletes a policy.
func (wrapper PolicyWrapper) DeletePolicy(policyArn string) error {
    _, err := wrapper.IamClient.DeletePolicy(context.TODO(), &iam.DeletePolicyInput{
        PolicyArn: aws.String(policyArn),
    })
    if err != nil {
        log.Printf("Couldn't delete policy %v. Here's why: %v\n", policyArn, err)
    }
    return err
}
```

- For API details, see [DeletePolicy](#) in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public static void deleteIAMPolicy(IamClient iam, String policyARN) {

    try {
        DeletePolicyRequest request = DeletePolicyRequest.builder()
            .policyArn(policyARN)
            .build();

        iam.deletePolicy(request);
        System.out.println("Successfully deleted the policy");

    } catch (IamException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
    System.out.println("Done");
}
```

- For API details, see [DeletePolicy](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Delete the policy.

```
import { DeletePolicyCommand, IAMClient } from "@aws-sdk/client-iam";
```

```
const client = new IAMClient({});

/**
 *
 * @param {string} policyArn
 */
export const deletePolicy = (policyArn) => {
  const command = new DeletePolicyCommand({ PolicyArn: policyArn });
  return client.send(command);
};
```

- For API details, see [DeletePolicy](#) in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

Note

This is prerelease documentation for a feature in preview release. It is subject to change.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
suspend fun deleteIAMPolicy(policyARNVal: String?) {

    val request = DeletePolicyRequest {
        policyArn = policyARNVal
    }

    IamClient { region = "AWS_GLOBAL" }.use { iamClient ->
        iamClient.deletePolicy(request)
        println("Successfully deleted $policyARNVal")
    }
}
```

- For API details, see [DeletePolicy](#) in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
def delete_policy(policy_arn):
    """
    Deletes a policy.

    :param policy_arn: The ARN of the policy to delete.
    """
    try:
        iam.Policy(policy_arn).delete()
        logger.info("Deleted policy %s.", policy_arn)
```

```
except ClientError:  
    logger.exception("Couldn't delete policy %s.", policy_arn)  
    raise
```

- For API details, see [DeletePolicy](#) in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
# Deletes a role. If the role has policies attached, they are detached and  
# deleted before the role is deleted.  
#  
# @param role [Aws::IAM::Role] The role to delete.  
def delete_role(role)  
    role.attached_policies.each do |policy|  
        name = policy.policy_name  
        policy.detach_role(role_name: role.name)  
        policy.delete  
        puts("Deleted policy #{name}.")  
    end  
    name = role.name  
    role.delete  
    puts("Deleted role #{name}.")  
rescue Aws::Errors::ServiceError => e  
    puts("Couldn't detach policies and delete role #{role.name}. Here's why:")  
    puts("t#{e.code}: #{e.message}")  
    raise  
end
```

- For API details, see [DeletePolicy](#) in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
pub async fn delete_policy(client: &iamClient, policy: Policy) -> Result<(),  
    iamError> {  
    client  
        .delete_policy()  
        .policy_arn(policy.arn.unwrap())  
        .send()  
        .await?;  
    Ok(())  
}
```

- For API details, see [DeletePolicy](#) in *AWS SDK for Rust API reference*.

Swift

SDK for Swift

Note

This is prerelease documentation for an SDK in preview release. It is subject to change.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public func deletePolicy(policy: IAMClientTypes.Policy) async throws {
    let input = DeletePolicyInput(
        policyArn: policy.arn
    )
    do {
        _ = try await iamClient.deletePolicy(input: input)
    } catch {
        throw error
    }
}
```

- For API details, see [DeletePolicy](#) in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Delete an IAM role using an AWS SDK

The following code examples show how to delete an IAM role.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- [Create a user and assume a role \(p. 891\)](#)
- [Manage roles \(p. 986\)](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Delete an IAM role.
/// </summary>
/// <param name="roleName">The name of the IAM role to delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
```

```
public async Task<bool> DeleteRoleAsync(string roleName)
{
    var response = await _IAMService.DeleteRoleAsync(new DeleteRoleRequest
{ RoleName = roleName });
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- For API details, see [DeleteRole](#) in *AWS SDK for .NET API Reference*.

Bash

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
#####
function iecho() {
    if [[ $VERBOSE == true ]]; then
        echo "$@"
    fi
}

#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
    printf "%s\n" "$*" 2>&1
}

#####
# function iam_delete_role
#
# This function deletes an IAM role.
#
# Parameters:
#     -n role_name -- The name of the IAM role.
#
# Returns:
#     0 - If successful.
#     1 - If it fails.
#####
function iam_delete_role() {
    local role_name response
    local option OPTARG # Required to use getopt command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_delete_role"
        echo "Deletes an AWS Identity and Access Management (IAM) role"
        echo "  -n role_name -- The name of the IAM role."
        echo ""
    }
}
```

```

# Retrieve the calling parameters.
while getopts "n:h" option; do
    case "${option}" in
        n) role_name="${OPTARG}" ;;
        h)
            usage
            return 0
            ;;
        \?) echo "Invalid parameter"
            usage
            return 1
            ;;
    esac
done
export OPTIND=1

echo "role_name:$role_name"
if [[ -z "$role_name" ]]; then
    errecho "ERROR: You must provide a role name with the -n parameter."
    usage
    return 1
fi

iecho "Parameters:\n"
iecho "    Role name: $role_name"
iecho ""

response=$(aws iam delete-role \
    --role-name "$role_name")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-role operation failed.\n$response"
    return 1
fi

iecho "delete-role response:$response"
iecho

return 0
}

```

- For API details, see [DeleteRole](#) in *AWS CLI Command Reference*.

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```

// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
// It contains an IAM service client that is used to perform role actions.
type RoleWrapper struct {
    IamClient *iam.Client
}

```

```
}

// DeleteRole deletes a role. All attached policies must be detached before a
// role can be deleted.
func (wrapper RoleWrapper) DeleteRole(roleName string) error {
    _, err := wrapper.IamClient.DeleteRole(context.TODO(), &iam.DeleteRoleInput{
        RoleName: aws.String(roleName),
    })
    if err != nil {
        log.Printf("Couldn't delete role %v. Here's why: %v\n", roleName, err)
    }
    return err
}
```

- For API details, see [DeleteRole](#) in *AWS SDK for Go API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Delete the role.

```
import { DeleteRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 */
export const deleteRole = (roleName) => {
    const command = new DeleteRoleCommand({ RoleName: roleName });
    return client.send(command);
};
```

- For API details, see [DeleteRole](#) in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
def delete_role(role_name):
    """
    Deletes a role.

    :param role_name: The name of the role to delete.
    """
```

```
try:
    iam.Role(role_name).delete()
    logger.info("Deleted role %s.", role_name)
except ClientError:
    logger.exception("Couldn't delete role %s.", role_name)
    raise
```

- For API details, see [DeleteRole](#) in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
# Deletes a role. If the role has policies attached, they are detached and
# deleted before the role is deleted.
#
# @param role [Aws::IAM::Role] The role to delete.
def delete_role(role)
  role.attached_policies.each do |policy|
    name = policy.policy_name
    policy.detach_role(role_name: role.name)
    policy.delete
    puts("Deleted policy #{name}.")
  end
  name = role.name
  role.delete
  puts("Deleted role #{name}.")
rescue Aws::Errors::ServiceError => e
  puts("Couldn't detach policies and delete role #{role.name}. Here's why:")
  puts("\t#{e.code}: #{e.message}")
  raise
end
```

- For API details, see [DeleteRole](#) in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
pub async fn delete_role(client: &iamClient, role: &Role) -> Result<(), iamError> {
    let role = role.clone();
    while client
        .delete_role()
        .role_name(role.role_name.as_ref().unwrap())
        .send()
```

```
        .await
        .is_err()
    {
        sleep(Duration::from_secs(2)).await;
    }
    Ok(())
}
```

- For API details, see [DeleteRole](#) in *AWS SDK for Rust API reference*.

Swift

SDK for Swift

Note

This is prerelease documentation for an SDK in preview release. It is subject to change.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public func deleteRole(role: IAMClientTypes.Role) async throws {
    let input = DeleteRoleInput(
        roleName: role.roleName
    )
    do {
        _ = try await iamClient.deleteRole(input: input)
    } catch {
        throw error
    }
}
```

- For API details, see [DeleteRole](#) in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Delete an IAM role policy using an AWS SDK

The following code examples show how to delete an IAM role policy.

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Delete an IAM role policy.
/// </summary>
/// <param name="roleName">The name of the IAM role.</param>
/// <param name="policyName">The name of the IAM role policy to delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
```

```
public async Task<bool> DeleteRolePolicyAsync(string roleName, string
policyName)
{
    var response = await _IAMService.DeleteRolePolicyAsync(new
DeleteRolePolicyRequest
{
    PolicyName = policyName,
    RoleName = roleName,
});

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- For API details, see [DeleteRolePolicy](#) in *AWS SDK for .NET API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import { DeleteRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 * @param {string} policyName
 */
export const deleteRolePolicy = (roleName, policyName) => {
    const command = new DeleteRolePolicyCommand({
        RoleName: roleName,
        PolicyName: policyName,
    });
    return client.send(command);
};
```

- For API details, see [DeleteRolePolicy](#) in *AWS SDK for JavaScript API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Delete an IAM server certificate using an AWS SDK

The following code examples show how to delete an IAM server certificate.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```

bool AwsDoc::IAM::deleteServerCertificate(const Aws::String &certificateName,
                                           const Aws::Client::ClientConfiguration
                                         &clientConfig) {
    Aws::IAM::IAMClient iam(clientConfig);
    Aws::IAM::Model::DeleteServerCertificateRequest request;
    request.SetServerCertificateName(certificateName);

    const auto outcome = iam.DeleteServerCertificate(request);
    bool result = true;
    if (!outcome.IsSuccess()) {
        if (outcome.GetError().GetErrorType() !=
            Aws::IAM::IAMErrors::NO_SUCH_ENTITY) {
            std::cerr << "Error deleting server certificate " << certificateName <<
                ":" << outcome.GetError().GetMessage() << std::endl;
            result = false;
        }
        else {
            std::cout << "Certificate '" << certificateName
                  << "' not found." << std::endl;
        }
    }
    else {
        std::cout << "Successfully deleted server certificate " << certificateName
              << std::endl;
    }

    return result;
}

```

- For API details, see [DeleteServerCertificate](#) in *AWS SDK for C++ API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Delete a server certificate.

```

import { DeleteServerCertificateCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} certName
 */
export const deleteServerCertificate = (certName) => {
    const command = new DeleteServerCertificateCommand({
        ServerCertificateName: certName,
    });

    return client.send(command);
};

```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [DeleteServerCertificate](#) in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// Load the AWS SDK for Node.js
var AWS = require('aws-sdk');
// Set the region
AWS.config.update({region: 'REGION'});

// Create the IAM service object
var iam = new AWS.IAM({apiVersion: '2010-05-08'});

iam.deleteServerCertificate({ServerCertificateName: 'CERTIFICATE_NAME'},
  function(err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data);
    }
});
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [DeleteServerCertificate](#) in [AWS SDK for JavaScript API Reference](#).

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Delete an IAM service-linked role using an AWS SDK

The following code examples show how to delete an IAM service-linked role.

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
// It contains an IAM service client that is used to perform role actions.
type RoleWrapper struct {
    IamClient *iam.Client
}

// DeleteServiceLinkedRole deletes a service-linked role.
func (wrapper RoleWrapper) DeleteServiceLinkedRole(roleName string) error {
    _, err := wrapper.IamClient.DeleteServiceLinkedRole(context.TODO(),
        &iam.DeleteServiceLinkedRoleInput{
            RoleName: aws.String(roleName),
        })
}
```

```
if err != nil {
    log.Printf("Couldn't delete service-linked role %v. Here's why: %v\n", roleName,
    err)
}
return err
}
```

- For API details, see [DeleteServiceLinkedRole](#) in *AWS SDK for Go API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import { DeleteServiceLinkedRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 */
export const deleteServiceLinkedRole = (roleName) => {
    const command = new DeleteServiceLinkedRoleCommand({ RoleName: roleName });
    return client.send(command);
};
```

- For API details, see [DeleteServiceLinkedRole](#) in *AWS SDK for JavaScript API Reference*.

Ruby

SDK for Ruby

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
# Deletes a service-linked role from the account.
#
# @param role [Aws::IAM::Role] The role to delete.
def delete_service_linked_role(role)
    response = @iam_resource.client.delete_service_linked_role(role_name:
role.name)
    task_id = response.deletion_task_id
    while true
        response = @iam_resource.client.get_service_linked_role_deletion_status(
            deletion_task_id: task_id)
        status = response.status
        puts("Deletion of #{role.name} #{status}.")
        if %w(SUCCEEDED FAILED).include?(status)
            break
        else
            sleep(3)
        end
    end
end
```

```
    end
rescue Aws::Errors::ServiceError => e
  # If AWS has not yet fully propagated the role, it deletes the role but
  # returns NoSuchEntity.
  if e.code != "NoSuchEntity"
    puts("Couldn't delete #{role.name}. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
end
end
```

- For API details, see [DeleteServiceLinkedRole](#) in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
pub async fn delete_service_linked_role(
    client: &iamClient,
    role_name: &str,
) -> Result<(), iamError> {
    client
        .delete_service_linked_role()
        .role_name(role_name)
        .send()
        .await?;

    Ok(())
}
```

- For API details, see [DeleteServiceLinkedRole](#) in *AWS SDK for Rust API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Delete an IAM user using an AWS SDK

The following code examples show how to delete an IAM user.

Warning

To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as [AWS IAM Identity Center \(successor to AWS Single Sign-On\)](#).

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- [Create a group and add a user \(p. 874\)](#)

- [Create a user and assume a role \(p. 891\)](#)
- [Create read-only and read-write users \(p. 973\)](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Delete an IAM user.
/// </summary>
/// <param name="userName">The username of the IAM user to delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteUserAsync(string userName)
{
    var response = await _IAMService.DeleteUserAsync(new DeleteUserRequest
{ UserName = userName });

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- For API details, see [DeleteUser](#) in *AWS SDK for .NET API Reference*.

Bash

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
#####
function iecho() {
    if [[ $VERBOSE == true ]]; then
        echo "$@"
    fi
}

#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
    printf "%s\n" "$*" 2>&1
}

#####
# function iam_delete_user
```

```

#
# This function deletes the specified IAM user.
#
# Parameters:
#     -u user_name -- The name of the user to create.
#
# Returns:
#     0 - If successful.
#     1 - If it fails.
#####
function iam_delete_user() {
    local user_name response
    local option OPTARG # Required to use getopts command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_delete_user"
        echo "Deletes an AWS Identity and Access Management (IAM) user. You must supply a username."
        echo "  -u user_name      The name of the user."
        echo ""
    }

    # Retrieve the calling parameters.
    while getopts "u:h" option; do
        case "${option}" in
            u)
                user_name="${OPTARG}";;
            h)
                usage
                return 0
                ;;
            \?)
                echo "Invalid parameter"
                usage
                return 1
                ;;
        esac
    done
    export OPTIND=1

    if [[ -z "$user_name" ]]; then
        errecho "ERROR: You must provide a username with the -u parameter."
        usage
        return 1
    fi

    iecho "Parameters:\n"
    iecho "  User name: $user_name"
    iecho ""

    # If the user does not exist, we don't want to try to delete it.
    if (! iam_user_exists "$user_name"); then
        errecho "ERROR: A user with that name does not exist in the account."
        return 1
    fi

    response=$(aws iam delete-user \
        --user-name "$user_name")

    local error_code=${?}

    if [[ $error_code -ne 0 ]]; then
        aws_cli_error_log $error_code
        errecho "ERROR: AWS reports delete-user operation failed.$response"
        return 1
    fi
}

```

```
    iecho "delete-user response:$response"
    iecho

    return 0
}
```

- For API details, see [DeleteUser](#) in *AWS CLI Command Reference*.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
Aws::IAM::IAMClient iam(clientConfig);

Aws::IAM::Model::DeleteUserRequest request;
request.SetUserName(userName);
auto outcome = iam.DeleteUser(request);
if (!outcome.IsSuccess()) {
    std::cerr << "Error deleting IAM user " << userName << ":" <<
        outcome.GetError().GetMessage() << std::endl;;
}
else {
    std::cout << "Successfully deleted IAM user " << userName << std::endl;
}

return outcome.IsSuccess();
```

- For API details, see [DeleteUser](#) in *AWS SDK for C++ API Reference*.

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// UserWrapper encapsulates user actions used in the examples.
// It contains an IAM service client that is used to perform user actions.
type UserWrapper struct {
    IamClient *iam.Client
}

// DeleteUser deletes a user.
func (wrapper UserWrapper) DeleteUser(userName string) error {
    _, err := wrapper.IamClient.DeleteUser(context.TODO(), &iam.DeleteUserInput{
        UserName: aws.String(userName),
})
if err != nil {
```

```
    log.Printf("Couldn't delete user %v. Here's why: %v\n", userName, err)
}
return err
}
```

- For API details, see [DeleteUser](#) in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public static void deleteIAMUser(IamClient iam, String userName) {

    try {
        DeleteUserRequest request = DeleteUserRequest.builder()
            .userName(userName)
            .build();

        iam.deleteUser(request);
        System.out.println("Successfully deleted IAM user " + userName);

    } catch (IamException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

- For API details, see [DeleteUser](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Delete the user.

```
import { DeleteUserCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} name
 */
export const deleteUser = (name) => {
    const command = new DeleteUserCommand({ UserName: name });
    return client.send(command);
};
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [DeleteUser](#) in [AWS SDK for JavaScript API Reference](#).

SDK for JavaScript (v2)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// Load the AWS SDK for Node.js
var AWS = require('aws-sdk');
// Set the region
AWS.config.update({region: 'REGION'});

// Create the IAM service object
var iam = new AWS.IAM({apiVersion: '2010-05-08'});

var params = {
  UserName: process.argv[2]
};

iam.getUser(params, function(err, data) {
  if (err && err.code === 'NoSuchEntity') {
    console.log("User " + process.argv[2] + " does not exist.");
  } else {
    iam.deleteUser(params, function(err, data) {
      if (err) {
        console.log("Error", err);
      } else {
        console.log("Success", data);
      }
    });
  }
});
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [DeleteUser](#) in [AWS SDK for JavaScript API Reference](#).

Kotlin

SDK for Kotlin

Note

This is prerelease documentation for a feature in preview release. It is subject to change.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
suspend fun deleteIAMUser(userNameVal: String) {

    val request = DeleteUserRequest {
        userName = userNameVal
    }

    // To delete a user, ensure that the user's access keys are deleted first.
    IamClient { region = "AWS_GLOBAL" }.use { iamClient ->
        iamClient.deleteUser(request)
        println("Successfully deleted user $userNameVal")
    }
}
```

```
    }  
}
```

- For API details, see [DeleteUser](#) in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
def delete_user(user_name):  
    """  
    Deletes a user. Before a user can be deleted, all associated resources,  
    such as access keys and policies, must be deleted or detached.  
  
    :param user_name: The name of the user.  
    """  
    try:  
        iam.User(user_name).delete()  
        logger.info("Deleted user %s.", user_name)  
    except ClientError:  
        logger.exception("Couldn't delete user %s.", user_name)  
        raise
```

- For API details, see [DeleteUser](#) in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
# Deletes a user. If the user has inline policies or access keys, they are  
# deleted  
# before the user is deleted.  
#  
# @param user [Aws::IAM::User] The user to delete.  
def delete_user(user)  
    user.policies.each do |policy|  
        name = policy.name  
        policy.delete  
        puts("Deleted user policy #{name}.")  
    end  
    user.access_keys.each do |key|  
        key.delete  
        puts("Deleted access key for user #{user.name}.")  
    end  
    name = user.name  
    user.delete  
    puts("Deleted user #{name}.")  
rescue Aws::Errors::ServiceError => e  
    puts("Couldn't detach policies and delete user #{user.name}. Here's why:")
```

```
    puts("\t#{e.code}: #{e.message}")
end
```

- For API details, see [DeleteUser](#) in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
pub async fn delete_user(client: &iamClient, user: &User) -> Result<(), SdkError<DeleteUserError>> {
    let user = user.clone();
    let mut tries: i32 = 0;
    let max_tries: i32 = 10;

    let response: Result<(), SdkError<DeleteUserError>> = loop {
        match client
            .delete_user()
            .user_name(user.user_name.as_ref().unwrap())
            .send()
            .await
        {
            Ok(_) => {
                break Ok(());
            }
            Err(e) => {
                tries += 1;
                if tries > max_tries {
                    break Err(e);
                }
                sleep(Duration::from_secs(2)).await;
            }
        };
    }
    response
}
```

- For API details, see [DeleteUser](#) in *AWS SDK for Rust API reference*.

Swift

SDK for Swift

Note

This is prerelease documentation for an SDK in preview release. It is subject to change.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public func deleteUser(user: IAMClientTypes.User) async throws {
    let input = DeleteUserInput(
        userName: user.userName
    )
    do {
        _ = try await iamClient.deleteUser(input: input)
    } catch {
        throw error
    }
}
```

- For API details, see [DeleteUser](#) in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Delete an IAM access key using an AWS SDK

The following code examples show how to delete an IAM access key.

Warning

To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as [AWS IAM Identity Center \(successor to AWS Single Sign-On\)](#).

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- [Create a group and add a user \(p. 874\)](#)
- [Create a user and assume a role \(p. 891\)](#)
- [Create read-only and read-write users \(p. 973\)](#)
- [Manage access keys \(p. 980\)](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Delete an IAM user's access key.
/// </summary>
/// <param name="accessKeyId">The Id for the IAM access key.</param>
/// <param name="userName">The username of the user that owns the IAM
/// access key.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteAccessKeyAsync(string accessKeyId, string
userName)
{
    var response = await _IAMService.DeleteAccessKeyAsync(new
DeleteAccessKeyRequest
{
    AccessKeyId = accessKeyId,
    UserName = userName,
```

```
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- For API details, see [DeleteAccessKey](#) in *AWS SDK for .NET API Reference*.

Bash

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
    printf "%s\n" "$*" 2>&1
}

#####
# function iam_delete_access_key
#
# This function deletes an IAM access key for the specified IAM user.
#
# Parameters:
#     -u user_name -- The name of the user.
#     -k access_key -- The access key to delete.
#
# Returns:
#     0 - If successful.
#     1 - If it fails.
#####
function iam_delete_access_key() {
    local user_name access_key response
    local option OPTARG # Required to use getopt command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_delete_access_key"
        echo "Deletes an AWS Identity and Access Management (IAM) access key for the"
        echo "specified IAM user"
        echo "  -u user_name      The name of the user."
        echo "  -k access_key     The access key to delete."
        echo ""
    }

    # Retrieve the calling parameters.
    while getopt "u:k:h" option; do
        case "${option}" in
            u) user_name="${OPTARG}" ;;
            k) access_key="${OPTARG}" ;;
            h)
                usage
                return 0
                ;;
            \?)
```

```

        echo "Invalid parameter"
        usage
        return 1
    ;;
esac
done
export OPTIND=1

if [[ -z "$user_name" ]]; then
    errecho "ERROR: You must provide a username with the -u parameter."
    usage
    return 1
fi

if [[ -z "$access_key" ]]; then
    errecho "ERROR: You must provide an access key with the -k parameter."
    usage
    return 1
fi

iecho "Parameters:\n"
iecho "    Username: $user_name"
iecho "    Access key: $access_key"
iecho ""

response=$(aws iam delete-access-key \
    --user-name "$user_name" \
    --access-key-id "$access_key")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-access-key operation failed.\n$response"
    return 1
fi

iecho "delete-access-key response:$response"
iecho

return 0
}

```

- For API details, see [DeleteAccessKey](#) in *AWS CLI Command Reference*.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```

bool AwsDoc::IAM::deleteAccessKey(const Aws::String &userName,
                                   const Aws::String &accessKeyID,
                                   const Aws::Client::ClientConfiguration
&clientConfig) {
    Aws::IAM::IAMClient iam(clientConfig);

    Aws::IAM::Model::DeleteAccessKeyRequest request;
    request.SetUserName(userName);
    request.SetAccessKeyId(accessKeyID);

```

```
auto outcome = iam.DeleteAccessKey(request);

if (!outcome.IsSuccess()) {
    std::cerr << "Error deleting access key " << accessKeyId << " from user "
    << userName << ":" << outcome.GetError().GetMessage() <<
    std::endl;
}
else {
    std::cout << "Successfully deleted access key " << accessKeyId
    << " for IAM user " << userName << std::endl;
}

return outcome.IsSuccess();
}
```

- For API details, see [DeleteAccessKey](#) in *AWS SDK for C++ API Reference*.

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// UserWrapper encapsulates user actions used in the examples.
// It contains an IAM service client that is used to perform user actions.
type UserWrapper struct {
    IamClient *iam.Client
}

// DeleteAccessKey deletes an access key from a user.
func (wrapper UserWrapper) DeleteAccessKey(userName string, keyId string) error {
    _, err := wrapper.IamClient.DeleteAccessKey(context.TODO(),
        &iam.DeleteAccessKeyInput{
            AccessKeyId: aws.String(keyId),
            UserName:    aws.String(userName),
        })
    if err != nil {
        log.Printf("Couldn't delete access key %v. Here's why: %v\n", keyId, err)
    }
    return err
}
```

- For API details, see [DeleteAccessKey](#) in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public static void deleteKey(IamClient iam ,String username, String accessKey ) {  
  
    try {  
        DeleteAccessKeyRequest request = DeleteAccessKeyRequest.builder()  
            .accessKeyId(accessKey)  
            .userName(username)  
            .build();  
  
        iam.deleteAccessKey(request);  
        System.out.println("Successfully deleted access key " + accessKey +  
            " from user " + username);  
  
    } catch (IamException e) {  
        System.err.println(e.awsErrorDetails().errorMessage());  
        System.exit(1);  
    }  
}
```

- For API details, see [DeleteAccessKey](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Delete the access key.

```
import { DeleteAccessKeyCommand, IAMClient } from "@aws-sdk/client-iam";  
  
const client = new IAMClient({});  
  
/**  
 *  
 * @param {string} userName  
 * @param {string} accessKeyId  
 */  
export const deleteAccessKey = (userName, accessKeyId) => {  
    const command = new DeleteAccessKeyCommand({  
        AccessKeyId: accessKeyId,  
        UserName: userName,  
    });  
  
    return client.send(command);  
};
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [DeleteAccessKey](#) in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// Load the AWS SDK for Node.js
```

```
var AWS = require('aws-sdk');
// Set the region
AWS.config.update({region: 'REGION'});

// Create the IAM service object
var iam = new AWS.IAM({apiVersion: '2010-05-08'});

var params = {
    AccessKeyId: 'ACCESS_KEY_ID',
    UserName: 'USER_NAME'
};

iam.deleteAccessKey(params, function(err, data) {
    if (err) {
        console.log("Error", err);
    } else {
        console.log("Success", data);
    }
});
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [DeleteAccessKey](#) in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

Note

This is prerelease documentation for a feature in preview release. It is subject to change.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
suspend fun deleteKey(userNameVal: String, accessKey: String) {

    val request = DeleteAccessKeyRequest {
        accessKeyId = accessKey
        userName = userNameVal
    }

    IamClient { region = "AWS_GLOBAL" }.use { iamClient ->
        iamClient.deleteAccessKey(request)
        println("Successfully deleted access key $accessKey from $userNameVal")
    }
}
```

- For API details, see [DeleteAccessKey](#) in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
def delete_key(user_name, key_id):
    """
    Deletes a user's access key.

    :param user_name: The user that owns the key.
    :param key_id: The ID of the key to delete.
    """

    try:
        key = iam.AccessKey(user_name, key_id)
        key.delete()
        logger.info(
            "Deleted access key %s for %s.", key.id, key.user_name)
    except ClientError:
        logger.exception("Couldn't delete key %s for %s", key_id, user_name)
        raise
```

- For API details, see [DeleteAccessKey](#) in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
# Deletes a user. If the user has inline policies or access keys, they are
deleted
# before the user is deleted.
#
# @param user [Aws::IAM::User] The user to delete.
def delete_user(user)
    user.policies.each do |policy|
        name = policy.name
        policy.delete
        puts("Deleted user policy #{name}.")
    end
    user.access_keys.each do |key|
        key.delete
        puts("Deleted access key for user #{user.name}.")
    end
    name = user.name
    user.delete
    puts("Deleted user #{name}.")
rescue Aws::Errors::ServiceError => e
    puts("Couldn't detach policies and delete user #{user.name}. Here's why:")
    puts("\t#{e.code}: #{e.message}")
end
```

- For API details, see [DeleteAccessKey](#) in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
pub async fn delete_access_key(
    client: &iamClient,
    user: &User,
    key: &AccessKey,
) -> Result<(), iamError> {
    loop {
        match client
            .delete_access_key()
            .user_name(user.user_name.as_ref().unwrap())
            .access_key_id(key.access_key_id.as_ref().unwrap())
            .send()
            .await
        {
            Ok(_) => {
                break;
            }
            Err(e) => {
                println!("Can't delete the access key: {:?}", e);
                sleep(Duration::from_secs(2)).await;
            }
        }
    }
    Ok(())
}
```

- For API details, see [DeleteAccessKey](#) in *AWS SDK for Rust API reference*.

Swift

SDK for Swift

Note

This is prerelease documentation for an SDK in preview release. It is subject to change.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public func deleteAccessKey(user: IAMClientTypes.User? = nil,
                           key: IAMClientTypes.AccessKey) async throws {
    let userName: String?

    if user != nil {
        userName = user!.userName
    } else {
        userName = nil
    }

    let input = DeleteAccessKeyInput(
        accessKeyId: key.accessKeyId,
        userName: userName
    )
    do {
        _ = try await iamClient.deleteAccessKey(input: input)
    } catch {
        throw error
    }
}
```

```
    }
```

- For API details, see [DeleteAccessKey](#) in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Delete an IAM account alias using an AWS SDK

The following code examples show how to delete an IAM account alias.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Manage your account \(p. 989\)](#)

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
bool AwsDoc::IAM::deleteAccountAlias(const Aws::String &accountAlias,
                                      const Aws::Client::ClientConfiguration
&clientConfig) {
    Aws::IAM::IAMClient iam(clientConfig);

    Aws::IAM::Model::DeleteAccountAliasRequest request;
    request.SetAccountAlias(accountAlias);

    const auto outcome = iam.DeleteAccountAlias(request);
    if (!outcome.IsSuccess()) {
        std::cerr << "Error deleting account alias " << accountAlias << ":" 
              << outcome.GetError().GetMessage() << std::endl;
    }
    else {
        std::cout << "Successfully deleted account alias " << accountAlias <<
                      std::endl;
    }

    return outcome.IsSuccess();
}
```

- For API details, see [DeleteAccountAlias](#) in *AWS SDK for C++ API Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public static void deleteIAMAccountAlias(IamClient iam, String alias ) {  
  
    try {  
        DeleteAccountAliasRequest request = DeleteAccountAliasRequest.builder()  
            .accountAlias(alias)  
            .build();  
  
        iam.deleteAccountAlias(request);  
        System.out.println("Successfully deleted account alias " + alias);  
  
    } catch (IamException e) {  
        System.err.println(e.awsErrorDetails().errorMessage());  
        System.exit(1);  
    }  
    System.out.println("Done");  
}
```

- For API details, see [DeleteAccountAlias](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Delete the account alias.

```
import { DeleteAccountAliasCommand, IAMClient } from "@aws-sdk/client-iam";  
  
const client = new IAMClient({});  
  
/**  
 *  
 * @param {string} alias  
 */  
export const deleteAccountAlias = (alias) => {  
    const command = new DeleteAccountAliasCommand({ AccountAlias: alias });  
  
    return client.send(command);  
};
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [DeleteAccountAlias](#) in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// Load the AWS SDK for Node.js  
var AWS = require('aws-sdk');  
// Set the region  
AWS.config.update({region: 'REGION'});  
  
// Create the IAM service object  
var iam = new AWS.IAM({apiVersion: '2010-05-08'});
```

```
iam.deleteAccountAlias({AccountAlias: process.argv[2]}, function(err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [DeleteAccountAlias](#) in [AWS SDK for JavaScript API Reference](#).

Kotlin

SDK for Kotlin

Note

This is prerelease documentation for a feature in preview release. It is subject to change.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
suspend fun deleteIAMAccountAlias(alias: String) {

    val request = DeleteAccountAliasRequest {
        accountAlias = alias
    }

    IamClient { region = "AWS_GLOBAL" }.use { iamClient ->
        iamClient.deleteAccountAlias(request)
        println("Successfully deleted account alias $alias")
    }
}
```

- For API details, see [DeleteAccountAlias](#) in [AWS SDK for Kotlin API reference](#).

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
def delete_alias(alias):
    """
    Removes the alias from the current account.

    :param alias: The alias to remove.
    """
    try:
        iam.meta.client.delete_account_alias(AccountAlias=alias)
        logger.info("Removed alias '%s' from your account.", alias)
    except ClientError:
        logger.exception("Couldn't remove alias '%s' from your account.", alias)
```

```
raise
```

- For API details, see [DeleteAccountAlias](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Delete an inline IAM policy from a user using an AWS SDK

The following code examples show how to delete an inline IAM policy from a user.

Warning

To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as [AWS IAM Identity Center \(successor to AWS Single Sign-On\)](#).

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Create a user and assume a role \(p. 891\)](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Delete an IAM user policy.
/// </summary>
/// <param name="policyName">The name of the IAM policy to delete.</param>
/// <param name="userName">The username of the IAM user.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteUserPolicyAsync(string policyName, string
userName)
{
    var response = await _IAMService.DeleteUserPolicyAsync(new
DeleteUserPolicyRequest { PolicyName = policyName, UserName = userName });

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- For API details, see [DeleteUserPolicy](#) in *AWS SDK for .NET API Reference*.

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// UserWrapper encapsulates user actions used in the examples.  
// It contains an IAM service client that is used to perform user actions.  
type UserWrapper struct {  
    IamClient *iam.Client  
}  
  
// DeleteUserPolicy deletes an inline policy from a user.  
func (wrapper UserWrapper) DeleteUserPolicy(userName string, policyName string)  
error {  
    _, err := wrapper.IamClient.DeleteUserPolicy(context.TODO(),  
&iam.DeleteUserPolicyInput{  
    PolicyName: aws.String(policyName),  
    UserName:   aws.String(userName),  
})  
if err != nil {  
    log.Printf("Couldn't delete policy from user %v. Here's why: %v\n", userName,  
err)  
}  
return err  
}
```

- For API details, see [DeleteUserPolicy](#) in *AWS SDK for Go API Reference*.

Ruby

SDK for Ruby

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
# Deletes a user. If the user has inline policies or access keys, they are  
deleted  
# before the user is deleted.  
#  
# @param user [Aws::IAM::User] The user to delete.  
def delete_user(user)  
    user.policies.each do |policy|  
        name = policy.name  
        policy.delete  
        puts("Deleted user policy #{name}.")  
    end  
    user.access_keys.each do |key|  
        key.delete  
        puts("Deleted access key for user #{user.name}.")  
    end  
    name = user.name  
    user.delete  
    puts("Deleted user #{name}.")  
rescue Aws::Errors::ServiceError => e  
    puts("Couldn't detach policies and delete user #{user.name}. Here's why:")  
    puts("\t#{e.code}: #{e.message}")  
end
```

- For API details, see [DeleteUserPolicy](#) in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
pub async fn delete_user_policy(
    client: &iamClient,
    user: &User,
    policy_name: &str,
) -> Result<(), SdkError<DeleteUserPolicyError>> {
    client
        .delete_user_policy()
        .user_name(user.user_name.as_ref().unwrap())
        .policy_name(policy_name)
        .send()
        .await?;

    Ok(())
}
```

- For API details, see [DeleteUserPolicy](#) in *AWS SDK for Rust API reference*.

Swift

SDK for Swift

Note

This is prerelease documentation for an SDK in preview release. It is subject to change.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
func deleteUserPolicy(user: IAMClientTypes.User, policyName: String) async throws {
    let input = DeleteUserPolicyInput(
        policyName: policyName,
        userName: user.userName
    )
    do {
        _ = try await iamClient.deleteUserPolicy(input: input)
    } catch {
        throw error
    }
}
```

- For API details, see [DeleteUserPolicy](#) in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Detach an IAM policy from a role using an AWS SDK

The following code examples show how to detach an IAM policy from a role.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- [Create a user and assume a role \(p. 891\)](#)
- [Manage roles \(p. 986\)](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Detach an IAM policy from an IAM role.
/// </summary>
/// <param name="policyArn">The Amazon Resource Name (ARN) of the IAM policy.</param>
/// <param name="roleName">The name of the IAM role.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DetachRolePolicyAsync(string policyArn, string
roleName)
{
    var response = await _IAMService.DetachRolePolicyAsync(new
DetachRolePolicyRequest
{
    PolicyArn = policyArn,
    RoleName = roleName,
});

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- For API details, see [DetachRolePolicy](#) in *AWS SDK for .NET API Reference*.

Bash

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
    printf "%s\n" "$*" 2>&1
}
```

```
#####
# function iam_detach_role_policy
#
# This function detaches an IAM policy to a role.
#
# Parameters:
#   -n role_name -- The name of the IAM role.
#   -p policy_ARN -- The IAM policy document ARN..
#
# Returns:
#   0 - If successful.
#   1 - If it fails.
#####
function iam_detach_role_policy() {
    local role_name policy_arn response
    local option OPTARG # Required to use getopts command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_detach_role_policy"
        echo "Detaches an AWS Identity and Access Management (IAM) policy to an IAM role."
        echo "  -n role_name  The name of the IAM role."
        echo "  -p policy_ARN -- The IAM policy document ARN."
        echo ""
    }

    # Retrieve the calling parameters.
    while getopts "n:p:h" option; do
        case "${option}" in
            n) role_name="${OPTARG}" ;;
            p) policy_arn="${OPTARG}" ;;
            h)
                usage
                return 0
                ;;
            \?)
                echo "Invalid parameter"
                usage
                return 1
                ;;
        esac
    done
    export OPTIND=1

    if [[ -z "$role_name" ]]; then
        errecho "ERROR: You must provide a role name with the -n parameter."
        usage
        return 1
    fi

    if [[ -z "$policy_arn" ]]; then
        errecho "ERROR: You must provide a policy ARN with the -p parameter."
        usage
        return 1
    fi

    response=$(aws iam detach-role-policy \
        --role-name "$role_name" \
        --policy-arn "$policy_arn")

    local error_code=${?}

    if [[ $error_code -ne 0 ]]; then
        aws_cli_error_log $error_code
    fi
}
```

```
errecho "ERROR: AWS reports detach-role-policy operation failed.\n$response"
return 1
fi

echo "$response"

return 0
}
```

- For API details, see [DetachRolePolicy](#) in *AWS CLI Command Reference*.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
Aws::IAM::IAMClient iam(clientConfig);

Aws::IAM::Model::DetachRolePolicyRequest detachRequest;
detachRequest.SetRoleName(roleName);
detachRequest.SetPolicyArn(policyArn);

auto detachOutcome = iam.DetachRolePolicy(detachRequest);
if (!detachOutcome.IsSuccess()) {
    std::cerr << "Failed to detach policy " << policyArn << " from role "
        << roleName << ": " << detachOutcome.GetError().GetMessage() <<
        std::endl;
}
else {
    std::cout << "Successfully detached policy " << policyArn << " from role "
        << roleName << std::endl;
}

return detachOutcome.IsSuccess();
```

- For API details, see [DetachRolePolicy](#) in *AWS SDK for C++ API Reference*.

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
// It contains an IAM service client that is used to perform role actions.
type RoleWrapper struct {
    IamClient *iam.Client
}
```

```
// DetachRolePolicy detaches a policy from a role.
func (wrapper RoleWrapper) DetachRolePolicy(roleName string, policyArn string)
    error {
    _, err := wrapper.IamClient.DetachRolePolicy(context.TODO(),
&iam.DetachRolePolicyInput{
    PolicyArn: aws.String(policyArn),
    RoleName:  aws.String(roleName),
})
if err != nil {
    log.Printf("Couldn't detach policy from role %v. Here's why: %v\n", roleName,
err)
}
return err
}
```

- For API details, see [DetachRolePolicy](#) in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public static void detachPolicy(IamClient iam, String roleName, String
policyArn ) {

    try {
        DetachRolePolicyRequest request = DetachRolePolicyRequest.builder()
            .roleName(roleName)
            .policyArn(policyArn)
            .build();

        iam.detachRolePolicy(request);
        System.out.println("Successfully detached policy " + policyArn +
                           " from role " + roleName);

    } catch (IamException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

- For API details, see [DetachRolePolicy](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Detach the policy.

```
import { DetachRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";
```

```
const client = new IAMClient({});

/**
 *
 * @param {string} policyArn
 * @param {string} roleName
 */
export const detachRolePolicy = (policyArn, roleName) => {
    const command = new DetachRolePolicyCommand({
        PolicyArn: policyArn,
        RoleName: roleName,
    });

    return client.send(command);
};
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [DetachRolePolicy](#) in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// Load the AWS SDK for Node.js
var AWS = require('aws-sdk');
// Set the region
AWS.config.update({region: 'REGION'});

// Create the IAM service object
var iam = new AWS.IAM({apiVersion: '2010-05-08'});

var paramsRoleList = {
    RoleName: process.argv[2]
};

iam.listAttachedRolePolicies(paramsRoleList, function(err, data) {
    if (err) {
        console.log("Error", err);
    } else {
        var myRolePolicies = data.AttachedPolicies;
        myRolePolicies.forEach(function (val, index, array) {
            if (myRolePolicies[index].PolicyName === 'AmazonDynamoDBFullAccess') {
                var params = {
                    PolicyArn: 'arn:aws:iam::aws:policy/AmazonDynamoDBFullAccess',
                    RoleName: process.argv[2]
                };
                iam.detachRolePolicy(params, function(err, data) {
                    if (err) {
                        console.log("Unable to detach policy from role", err);
                    } else {
                        console.log("Policy detached from role successfully");
                        process.exit();
                    }
                });
            }
        });
    }
});
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [DetachRolePolicy](#) in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

Note

This is prerelease documentation for a feature in preview release. It is subject to change.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
suspend fun detachPolicy(roleNameVal: String, policyArnVal: String) {  
  
    val request = DetachRolePolicyRequest {  
        roleName = roleNameVal  
        policyArn = policyArnVal  
    }  
  
    IamClient { region = "AWS_GLOBAL" }.use { iamClient ->  
        iamClient.detachRolePolicy(request)  
        println("Successfully detached policy $policyArnVal from role  
$roleNameVal")  
    }  
}
```

- For API details, see [DetachRolePolicy](#) in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Detach a policy from a role using the Boto3 Policy object.

```
def detach_from_role(role_name, policy_arn):  
    """  
    Detaches a policy from a role.  
  
    :param role_name: The name of the role. **Note** this is the name, not the ARN.  
    :param policy_arn: The ARN of the policy.  
    """  
    try:  
        iam.Policy(policy_arn).detach_role(RoleName=role_name)  
        logger.info("Detached policy %s from role %s.", policy_arn, role_name)  
    except ClientError:  
        logger.exception(  
            "Couldn't detach policy %s from role %s.", policy_arn, role_name)  
        raise
```

Detach a policy from a role using the Boto3 Role object.

```
def detach_policy(role_name, policy_arn):
    """
    Detaches a policy from a role.

    :param role_name: The name of the role. **Note** this is the name, not the ARN.
    :param policy_arn: The ARN of the policy.
    """
    try:
        iam.Role(role_name).detach_policy(PolicyArn=policy_arn)
        logger.info("Detached policy %s from role %s.", policy_arn, role_name)
    except ClientError:
        logger.exception(
            "Couldn't detach policy %s from role %s.", policy_arn, role_name)
        raise
```

- For API details, see [DetachRolePolicy](#) in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
# Deletes a role. If the role has policies attached, they are detached and
# deleted before the role is deleted.
#
# @param role [Aws::IAM::Role] The role to delete.
def delete_role(role)
    role.attached_policies.each do |policy|
        name = policy.policy_name
        policy.detach_role(role_name: role.name)
        policy.delete
        puts("Deleted policy #{name}.")
    end
    name = role.name
    role.delete
    puts("Deleted role #{name}.")
rescue Aws::Errors::ServiceError => e
    puts("Couldn't detach policies and delete role #{role.name}. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
end
```

- For API details, see [DetachRolePolicy](#) in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
pub async fn detach_role_policy(  
    client: &iamClient,  
    role_name: &str,  
    policy_arn: &str,  
) -> Result<(), iamError> {  
    client  
        .detach_role_policy()  
        .role_name(role_name)  
        .policy_arn(policy_arn)  
        .send()  
        .await?;  
  
    Ok(())  
}
```

- For API details, see [DetachRolePolicy](#) in *AWS SDK for Rust API reference*.

Swift

SDK for Swift

Note

This is prerelease documentation for an SDK in preview release. It is subject to change.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public func detachRolePolicy(policy: IAMClientTypes.Policy, role:  
    IAMClientTypes.Role) async throws {  
    let input = DetachRolePolicyInput(  
        policyArn: policy.arn,  
        roleName: role.roleName  
    )  
  
    do {  
        _ = try await iamClient.detachRolePolicy(input: input)  
    } catch {  
        throw error  
    }  
}
```

- For API details, see [DetachRolePolicy](#) in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Detach an IAM policy from a user using an AWS SDK

The following code examples show how to detach an IAM policy from a user.

Warning

To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as [AWS IAM Identity Center \(successor to AWS Single Sign-On\)](#).

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Create read-only and read-write users \(p. 973\)](#)

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
def detach_policy(user_name, policy_arn):
    """
    Detaches a policy from a user.

    :param user_name: The name of the user.
    :param policy_arn: The Amazon Resource Name (ARN) of the policy.
    """
    try:
        iam.User(user_name).detach_policy(PolicyArn=policy_arn)
        logger.info("Detached policy %s from user %s.", policy_arn, user_name)
    except ClientError:
        logger.exception(
            "Couldn't detach policy %s from user %s.", policy_arn, user_name)
        raise
```

- For API details, see [DetachUserPolicy](#) in *AWS SDK for Python (Boto3) API Reference*.

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
pub async fn detach_user_policy(
    client: &iamClient,
    user_name: &str,
    policy_arn: &str,
) -> Result<(), iamError> {
    client
        .detach_user_policy()
        .user_name(user_name)
        .policy_arn(policy_arn)
        .send()
        .await?;

    Ok(())
}
```

- For API details, see [DetachUserPolicy](#) in *AWS SDK for Rust API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Generate a credential report from IAM using an AWS SDK

The following code example shows how to generate a credential report from IAM for the current account. After the report is generated, get it by using the GetCredentialReport action.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Manage your account \(p. 989\)](#)

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
def generate_credential_report():
    """
    Starts generation of a credentials report about the current account. After
    calling this function to generate the report, call get_credential_report
    to get the latest report. A new report can be generated a minimum of four hours
    after the last one was generated.
    """
    try:
        response = iam.meta.client.generate_credential_report()
        logger.info("Generating credentials report for your account. "
                    "Current state is %s.", response['State'])
    except ClientError:
        logger.exception("Couldn't generate a credentials report for your
account.")
        raise
    else:
        return response
```

- For API details, see [GenerateCredentialReport](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Get a credential report from IAM using an AWS SDK

The following code example shows how to get the most recently generated credential report from IAM.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Manage your account \(p. 989\)](#)

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
def get_credential_report():
    """
    Gets the most recently generated credentials report about the current account.

    :return: The credentials report.
    """
    try:
        response = iam.meta.client.get_credential_report()
        logger.debug(response['Content'])
    except ClientError:
        logger.exception("Couldn't get credentials report.")
        raise
    else:
        return response['Content']
```

- For API details, see [GetCredentialReport](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Get a detailed IAM authorization report for your account using an AWS SDK

The following code example shows how to get a detailed IAM authorization report for your account.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Manage your account \(p. 989\)](#)

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
def get_authorization_details(response_filter):
    """
    Gets an authorization detail report for the current account.

    :param response_filter: A list of resource types to include in the report, such
                           as users or roles. When not specified, all resources
                           are included.
    :return: The authorization detail report.
    """
    try:
```

```
account_details = iam.meta.client.get_account_authorization_details(
    Filter=response_filter
)
logger.debug(account_details)
except ClientError:
    logger.exception("Couldn't get details for your account.")
    raise
else:
    return account_details
```

- For API details, see [GetAccountAuthorizationDetails in AWS SDK for Python \(Boto3\) API Reference](#).

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Get an IAM policy using an AWS SDK

The following code examples show how to get an IAM policy.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Work with the IAM Policy Builder API \(p. 994\)](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Get information about an IAM policy.
/// </summary>
/// <param name="policyArn">The IAM policy to retrieve information for.</param>
/// <returns>The IAM policy.</returns>
public async Task<ManagedPolicy> GetPolicyAsync(string policyArn)
{
    var response = await _IAMService.GetPolicyAsync(new GetPolicyRequest
    { PolicyArn = policyArn });
    return response.Policy;
}
```

- For API details, see [GetPolicy in AWS SDK for .NET API Reference](#).

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```

bool AwsDoc::IAM::getPolicy(const Aws::String &policyArn,
                             const Aws::Client::ClientConfiguration &clientConfig) {
    Aws::IAM::IAMClient iam(clientConfig);
    Aws::IAM::Model::GetPolicyRequest request;
    request.SetPolicyArn(policyArn);

    auto outcome = iam.GetPolicy(request);
    if (!outcome.IsSuccess()) {
        std::cerr << "Error getting policy " << policyArn << ": " <<
            outcome.GetError().GetMessage() << std::endl;
    }
    else {
        const auto &policy = outcome.GetResult().GetPolicy();
        std::cout << "Name: " << policy.GetPolicyName() << std::endl <<
            "ID: " << policy.GetPolicyId() << std::endl << "Arn: " <<
            policy.GetArn() << std::endl << "Description: " <<
            policy.GetDescription() << std::endl << "CreateDate: " <<

        policy.GetCreateDate().ToGmtString(Aws::Utils::DateFormat::ISO_8601)
            << std::endl;
    }
    return outcome.IsSuccess();
}

```

- For API details, see [GetPolicy](#) in *AWS SDK for C++ API Reference*.

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```

// PolicyWrapper encapsulates AWS Identity and Access Management (IAM) policy
// actions
// used in the examples.
// It contains an IAM service client that is used to perform policy actions.
type PolicyWrapper struct {
    IamClient *iam.Client
}

// GetPolicy gets data about a policy.
func (wrapper PolicyWrapper) GetPolicy(policyArn string) (*types.Policy, error) {
    var policy *types.Policy
    result, err := wrapper.IamClient.GetPolicy(context.TODO(), &iam.GetPolicyInput{
        PolicyArn: aws.String(policyArn),
    })
    if err != nil {
        log.Printf("Couldn't get policy %v. Here's why: %v\n", policyArn, err)
    } else {
        policy = result.Policy
    }
    return policy, err
}

```

- For API details, see [GetPolicy](#) in *AWS SDK for Go API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Get the policy.

```
import { GetPolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} policyArn
 */
export const getPolicy = (policyArn) => {
  const command = new GetPolicyCommand({
    PolicyArn: policyArn,
  });

  return client.send(command);
};
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [GetPolicy](#) in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// Load the AWS SDK for Node.js
var AWS = require('aws-sdk');
// Set the region
AWS.config.update({region: 'REGION'});

// Create the IAM service object
var iam = new AWS.IAM({apiVersion: '2010-05-08'});

var params = {
  PolicyArn: 'arn:aws:iam::aws:policy/AWSLambdaExecute'
};

iam.getPolicy(params, function(err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Policy.Description);
  }
});
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [GetPolicy](#) in [AWS SDK for JavaScript API Reference](#).

Kotlin

SDK for Kotlin

Note

This is prerelease documentation for a feature in preview release. It is subject to change.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
suspend fun getIAMPolicy(policyArnVal: String?) {  
  
    val request = GetPolicyRequest {  
        policyArn = policyArnVal  
    }  
  
    IamClient { region = "AWS_GLOBAL" }.use { iamClient ->  
        val response = iamClient.getPolicy(request)  
        println("Successfully retrieved policy ${response.policy?.policyName}")  
    }  
}
```

- For API details, see [GetPolicy](#) in [AWS SDK for Kotlin API reference](#).

PHP

SDK for PHP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
$uuid = uniqid();  
$service = new IAMService();  
  
public function getPolicy($policyArn)  
{  
    return $this->customWaiter(function () use ($policyArn) {  
        return $this->iamClient->getPolicy(['PolicyArn' => $policyArn]);  
    });  
}
```

- For API details, see [GetPolicy](#) in [AWS SDK for PHP API Reference](#).

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
def get_default_policy_statement(policy_arn):
    """
    Gets the statement of the default version of the specified policy.

    :param policy_arn: The ARN of the policy to look up.
    :return: The statement of the default policy version.
    """
    try:
        policy = iam.Policy(policy_arn)
        # To get an attribute of a policy, the SDK first calls get_policy.
        policy_doc = policy.default_version.document
        policy_statement = policy_doc.get('Statement', None)
        logger.info("Got default policy doc for %s.", policy.policy_name)
        logger.info(policy_doc)
    except ClientError:
        logger.exception("Couldn't get default policy statement for %s.",
                         policy_arn)
        raise
    else:
        return policy_statement
```

- For API details, see [GetPolicy](#) in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
# Gets data about a policy.
#
# @param policy_arn [String] The ARN of the policy to look up.
# @return [Aws::IAM::Policy] The retrieved policy.
def get_policy(policy_arn)
    policy = @iam_resource.policy(policy_arn)
    puts("Got policy '#{policy.policy_name}'. Its ID is: #{policy.policy_id}.")
rescue Aws::Errors::ServiceError => e
    puts("Couldn't get policy '#{policy_arn}'. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
else
    policy
end
```

- For API details, see [GetPolicy](#) in *AWS SDK for Ruby API Reference*.

Swift

SDK for Swift

Note

This is prerelease documentation for an SDK in preview release. It is subject to change.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public func getPolicy(arn: String) async throws -> IAMClientTypes.Policy {  
    let input = GetPolicyInput(  
        policyArn: arn  
    )  
    do {  
        let output = try await client.getPolicy(input: input)  
        guard let policy = output.policy else {  
            throw ServiceHandlerError.noSuchPolicy  
        }  
        return policy  
    } catch {  
        throw error  
    }  
}
```

- For API details, see [GetPolicy](#) in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Get an IAM policy version using an AWS SDK

The following code example shows how to get an IAM policy version.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- [Manage policies \(p. 982\)](#)
- [Work with the IAM Policy Builder API \(p. 994\)](#)

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
def get_default_policy_statement(policy_arn):  
    """  
    Gets the statement of the default version of the specified policy.  
  
    :param policy_arn: The ARN of the policy to look up.  
    :return: The statement of the default policy version.  
    """  
    try:  
        policy = iam.Policy(policy_arn)  
        # To get an attribute of a policy, the SDK first calls get_policy.  
        policy_doc = policy.default_version.document  
        policy_statement = policy_doc.get('Statement', None)  
        logger.info("Got default policy doc for %s.", policy.policy_name)  
        logger.info(policy_doc)  
    except ClientError:  
        logger.exception("Couldn't get default policy statement for %s.",  
                         policy_arn)  
        raise
```

```
    else:  
        return policy_statement
```

- For API details, see [GetPolicyVersion](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Get an IAM role using an AWS SDK

The following code examples show how to get an IAM role.

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>  
/// Get information about an IAM role.  
/// </summary>  
/// <param name="roleName">The name of the IAM role to retrieve information  
/// for.</param>  
/// <returns>The IAM role that was retrieved.</returns>  
public async Task<Role> GetRoleAsync(string roleName)  
{  
    var response = await _IAMService.GetRoleAsync(new GetRoleRequest  
    {  
        RoleName = roleName,  
    });  
  
    return response.Role;  
}
```

- For API details, see [GetRole](#) in *AWS SDK for .NET API Reference*.

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions  
// used in the examples.  
// It contains an IAM service client that is used to perform role actions.  
type RoleWrapper struct {  
    IamClient *iam.Client  
}
```

```
// GetRole gets data about a role.
func (wrapper RoleWrapper) GetRole(roleName string) (*types.Role, error) {
    var role *types.Role
    result, err := wrapper.IamClient.GetRole(context.TODO(),
        &iam.GetRoleInput{RoleName: aws.String(roleName)})
    if err != nil {
        log.Printf("Couldn't get role %v. Here's why: %v\n", roleName, err)
    } else {
        role = result.Role
    }
    return role, err
}
```

- For API details, see [GetRole](#) in *AWS SDK for Go API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Get the role.

```
import { GetRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 */
export const getRole = (roleName) => {
    const command = new GetRoleCommand({
        RoleName: roleName,
    });

    return client.send(command);
};
```

- For API details, see [GetRole](#) in *AWS SDK for JavaScript API Reference*.

PHP

SDK for PHP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
$uuid = uniqid();
$service = new IAMService();

public function getRole($roleName)
{
```

```
        return $this->customWaiter(function () use ($roleName) {
            return $this->iamClient->getRole(['RoleName' => $roleName]);
        });
    }
```

- For API details, see [GetRole](#) in *AWS SDK for PHP API Reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
def get_role(role_name):
    """
    Gets a role by name.

    :param role_name: The name of the role to retrieve.
    :return: The specified role.
    """
    try:
        role = iam.Role(role_name)
        role.load() # calls GetRole to load attributes
        logger.info("Got role with arn %s.", role.arn)
    except ClientError:
        logger.exception("Couldn't get role named %s.", role_name)
        raise
    else:
        return role
```

- For API details, see [GetRole](#) in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
# Gets data about a role.
#
# @param name [String] The name of the role to look up.
# @return [Aws::IAM::Role] The retrieved role.
def get_role(name)
    role = @iam_resource.role(name)
    puts("Got data for role '#{role.name}'. Its ARN is '#{role.arn}'")
rescue Aws::Errors::ServiceError => e
    puts("Couldn't get data for role '#{name}' Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
else
    role
end
```

- For API details, see [GetRole](#) in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
pub async fn get_role(
    client: &iamClient,
    role_name: String,
) -> Result<GetRoleOutput, SdkError<GetRoleError>> {
    let response = client.get_role().role_name(role_name).send().await?;
    Ok(response)
}
```

- For API details, see [GetRole](#) in *AWS SDK for Rust API reference*.

Swift

SDK for Swift

Note

This is prerelease documentation for an SDK in preview release. It is subject to change.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public func getRole(name: String) async throws -> IAMClientTypes.Role {
    let input = GetRoleInput(
        roleName: name
    )
    do {
        let output = try await client.getRole(input: input)
        guard let role = output.role else {
            throw ServiceHandlerError.noSuchRole
        }
        return role
    } catch {
        throw error
    }
}
```

- For API details, see [GetRole](#) in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Get an IAM server certificate using an AWS SDK

The following code examples show how to get an IAM server certificate.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
bool AwsDoc::IAM::getServerCertificate(const Aws::String &certificateName,
                                         const Aws::Client::ClientConfiguration
                                         &clientConfig) {
    Aws::IAM::IAMClient iam(clientConfig);
    Aws::IAM::Model::GetServerCertificateRequest request;
    request.SetServerCertificateName(certificateName);

    auto outcome = iam.GetServerCertificate(request);
    bool result = true;
    if (!outcome.IsSuccess()) {
        if (outcome.GetError().GetErrorType() !=
            Aws::IAM::IAMErrors::NO_SUCH_ENTITY) {
            std::cerr << "Error getting server certificate " << certificateName <<
                ":" << outcome.GetError().GetMessage() << std::endl;
            result = false;
        }
        else {
            std::cout << "Certificate '" << certificateName
                  << "' not found." << std::endl;
        }
    }
    else {
        const auto &certificate = outcome.GetResult().GetServerCertificate();
        std::cout << "Name: " <<

        certificate.GetServerCertificateMetadata().GetServerCertificateName()
            << std::endl << "Body: " << certificate.GetCertificateBody() <<
            std::endl << "Chain: " << certificate.GetCertificateChain() <<
            std::endl;
    }
}

return result;
}
```

- For API details, see [GetServerCertificate](#) in *AWS SDK for C++ API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Get a server certificate.

```
import { GetServerCertificateCommand, IAMClient } from "@aws-sdk/client-iam";
```

```
const client = new IAMClient({});

/**
 *
 * @param {string} certName
 * @returns
 */
export const getServerCertificate = async (certName) => {
  const command = new GetServerCertificateCommand({
    ServerCertificateName: certName,
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [GetServerCertificate](#) in [AWS SDK for JavaScript API Reference](#).

SDK for JavaScript (v2)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// Load the AWS SDK for Node.js
var AWS = require('aws-sdk');
// Set the region
AWS.config.update({region: 'REGION'});

// Create the IAM service object
var iam = new AWS.IAM({apiVersion: '2010-05-08'});

iam.getServerCertificate({ServerCertificateName: 'CERTIFICATE_NAME'}, function(err,
data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [GetServerCertificate](#) in [AWS SDK for JavaScript API Reference](#).

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Get an IAM service-linked role's deletion status using an AWS SDK

The following code example shows how to get an AWS Identity and Access Management (IAM) service-linked role's deletion status.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import { GetServiceLinkedRoleDeletionStatusCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} deletionTaskId
 */
export const getServiceLinkedRoleDeletionStatus = (deletionTaskId) => {
  const command = new GetServiceLinkedRoleDeletionStatusCommand({
    DeletionTaskId: deletionTaskId,
  });

  return client.send(command);
};
```

- For API details, see [GetServiceLinkedRoleDeletionStatus](#) in *AWS SDK for JavaScript API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Get a summary of account usage from IAM using an AWS SDK

The following code example shows how to get a summary of account usage from IAM.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Manage your account \(p. 989\)](#)

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
def get_summary():
    """
    Gets a summary of account usage.

    :return: The summary of account usage.
    """
    try:
        summary = iam.AccountSummary()
```

```
    logger.debug(summary.summary_map)
except ClientError:
    logger.exception("Couldn't get a summary for your account.")
    raise
else:
    return summary.summary_map
```

- For API details, see [GetAccountSummary](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Get an IAM user using an AWS SDK

The following code examples show how to get an IAM user.

Warning

To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as [AWS IAM Identity Center \(successor to AWS Single Sign-On\)](#).

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Get information about an IAM user.
/// </summary>
/// <param name="userName">The username of the user.</param>
/// <returns>An IAM user object.</returns>
public async Task<User> GetUserAsync(string userName)
{
    var response = await _IAMService.GetUserAsync(new GetUserRequest { UserName
= userName });
    return response.User;
}
```

- For API details, see [GetUser](#) in *AWS SDK for .NET API Reference*.

Bash

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function errecho
```

```

#
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
    printf "%s\n" "$*" 2>&1
}

#####
# function iam_user_exists
#
# This function checks to see if the specified AWS Identity and Access Management
# (IAM) user already exists.
#
# Parameters:
#     $1 - The name of the IAM user to check.
#
# Returns:
#     0 - If the user already exists.
#     1 - If the user doesn't exist.
#####
function iam_user_exists() {
    local user_name
    user_name=$1

    # Check whether the IAM user already exists.
    # We suppress all output - we're interested only in the return code.

    local errors
    errors=$(aws iam get-user \
        --user-name "$user_name" 2>&1 >/dev/null)

    local error_code=${?}

    if [[ $error_code -eq 0 ]]; then
        return 0 # 0 in Bash script means true.
    else
        if [[ $errors != *"error**"(NoSuchEntity)* ]]; then
            aws_cli_error_log $error_code
            errecho "Error calling iam get-user $errors"
        fi

        return 1 # 1 in Bash script means false.
    fi
}

```

- For API details, see [GetUser](#) in *AWS CLI Command Reference*.

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```

// UserWrapper encapsulates user actions used in the examples.
// It contains an IAM service client that is used to perform user actions.
type UserWrapper struct {
    IamClient *iam.Client
}

```

```
// GetUser gets data about a user.
func (wrapper UserWrapper) GetUser(userName string) (*types.User, error) {
    var user *types.User
    result, err := wrapper.IamClient.GetUser(context.TODO(), &iam.GetUserInput{
        UserName: aws.String(userName),
    })
    if err != nil {
        var apiError smithy.APIError
        if errors.As(err, &apiError) {
            switch apiError.(type) {
            case *types.NoSuchEntityException:
                log.Printf("User %v does not exist.\n", userName)
                err = nil
            default:
                log.Printf("Couldn't get user %v. Here's why: %v\n", userName, err)
            }
        }
    } else {
        user = result.User
    }
    return user, err
}
```

- For API details, see [GetUser](#) in [AWS SDK for Go API Reference](#).

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Get data about the last use of an IAM access key using an AWS SDK

The following code examples show how to get data about the last use of an IAM access key.

Warning

To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as [AWS IAM Identity Center \(successor to AWS Single Sign-On\)](#).

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Manage access keys \(p. 980\)](#)

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
bool AwsDoc::IAM::accessKeyLastUsed(const Aws::String &secretKeyID,
                                      const Aws::Client::ClientConfiguration
                                      &clientConfig) {
```

```

Aws::IAM::IAMClient iam(clientConfig);
Aws::IAM::Model::GetAccessKeyLastUsedRequest request;

request.SetAccessKeyId(secretKeyID);

Aws::IAM::Model::GetAccessKeyLastUsedOutcome outcome =
iam.GetAccessKeyLastUsed(
    request);

if (!outcome.IsSuccess()) {
    std::cerr << "Error querying last used time for access key " <<
        secretKeyID << ":" << outcome.GetError().GetMessage() <<
    std::endl;
}
else {
    Aws::String lastUsedTimeString =
        outcome.GetResult()
            .GetAccessKeyLastUsed()
            .GetLastUsedDate()
            .ToGmtString(Aws::Utils::DateFormat::ISO_8601);
    std::cout << "Access key " << secretKeyID << " last used at time " <<
        lastUsedTimeString << std::endl;
}

return outcome.IsSuccess();
}

```

- For API details, see [GetAccessKeyLastUsed](#) in *AWS SDK for C++ API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Get the access key.

```

import { GetAccessKeyLastUsedCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} accessKeyId
 */
export const getAccessKeyLastUsed = async (accessKeyId) => {
    const command = new GetAccessKeyLastUsedCommand({
        AccessKeyId: accessKeyId,
    });

    const response = await client.send(command);

    if (response.AccessKeyLastUsed?.LastUsedDate) {
        console.log(`
            ${accessKeyId} was last used by ${response.UserName} via
            the ${response.AccessKeyLastUsed.ServiceName} service on
            ${response.AccessKeyLastUsed.LastUsedDate.toISOString()}
        `);
    }
}

```

```
    return response;
};
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [GetAccessKeyLastUsed](#) in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// Load the AWS SDK for Node.js
var AWS = require('aws-sdk');
// Set the region
AWS.config.update({region: 'REGION'});

// Create the IAM service object
var iam = new AWS.IAM({apiVersion: '2010-05-08'});

iam.getAccessKeyLastUsed({AccessKeyId: 'ACCESS_KEY_ID'}, function(err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.AccessKeyLastUsed);
  }
});
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [GetAccessKeyLastUsed](#) in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
def get_last_use(key_id):
    """
    Gets information about when and how a key was last used.

    :param key_id: The ID of the key to look up.
    :return: Information about the key's last use.
    """
    try:
        response = iam.meta.client.get_access_key_last_used(AccessKeyId=key_id)
        last_used_date = response['AccessKeyLastUsed'].get('LastUsedDate', None)
        last_service = response['AccessKeyLastUsed'].get('ServiceName', None)
        logger.info(
            "Key %s was last used by %s on %s to access %s.", key_id,
            response['UserName'], last_used_date, last_service)
    except ClientError:
        logger.exception("Couldn't get last use of key %s.", key_id)
        raise
    else:
        return response
```

- For API details, see [GetAccessKeyLastUsed](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Get the IAM account password policy using an AWS SDK

The following code examples show how to get the IAM account password policy.

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Gets the IAM password policy for an AWS account.
/// </summary>
/// <returns>The PasswordPolicy for the AWS account.</returns>
public async Task<PasswordPolicy> GetAccountPasswordPolicyAsync()
{
    var response = await _IAMService.GetAccountPasswordPolicyAsync(new
GetAccountPasswordPolicyRequest());
    return response.PasswordPolicy;
}
```

- For API details, see [GetAccountPasswordPolicy](#) in *AWS SDK for .NET API Reference*.

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// AccountWrapper encapsulates AWS Identity and Access Management (IAM) account
// actions
// used in the examples.
// It contains an IAM service client that is used to perform account actions.
type AccountWrapper struct {
    IamClient *iam.Client
}

// GetAccountPasswordPolicy gets the account password policy for the current
// account.
// If no policy has been set, a NoSuchEntityException is error is returned.
```

```
func (wrapper AccountWrapper) GetAccountPasswordPolicy() (*types.PasswordPolicy, error) {
    var pwPolicy *types.PasswordPolicy
    result, err := wrapper.IamClient.GetAccountPasswordPolicy(context.TODO(),
        &iam.GetAccountPasswordPolicyInput{})
    if err != nil {
        log.Printf("Couldn't get account password policy. Here's why: %v\n", err)
    } else {
        pwPolicy = result.PasswordPolicy
    }
    return pwPolicy, err
}
```

- For API details, see [GetAccountPasswordPolicy](#) in *AWS SDK for Go API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Get the account password policy.

```
import {
    GetAccountPasswordPolicyCommand,
    IAMClient,
} from "@aws-sdk/client-iam";

const client = new IAMClient({});

export const getAccountPasswordPolicy = async () => {
    const command = new GetAccountPasswordPolicyCommand({});

    const response = await client.send(command);
    console.log(response.PasswordPolicy);
    return response;
};
```

- For API details, see [GetAccountPasswordPolicy](#) in *AWS SDK for JavaScript API Reference*.

PHP

SDK for PHP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
$uuid = uniqid();
 getService();

    public function getAccountPasswordPolicy()
    {
        return $this->iamClient->getAccountPasswordPolicy();
    }
}
```

- For API details, see [GetAccountPasswordPolicy](#) in *AWS SDK for PHP API Reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
def print_password_policy():
    """
    Prints the password policy for the account.
    """
    try:
        pw_policy = iam.AccountPasswordPolicy()
        print("Current account password policy:")
        print(f"\tallow_users_to_change_password: {pw_policy.allow_users_to_change_password}")
        print(f"\texpire_passwords: {pw_policy.expire_passwords}")
        print(f"\thard_expiry: {pw_policy.hard_expiry}")
        print(f"\tmax_password_age: {pw_policy.max_password_age}")
        print(f"\tminimum_password_length: {pw_policy.minimum_password_length}")
        print(f"\tpassword_reuse_prevention: {pw_policy.password_reuse_prevention}")
        print(f"\trequire_lowercase_characters: {pw_policy.require_lowercase_characters}")
        print(f"\trequire_numbers: {pw_policy.require_numbers}")
        print(f"\trequire_symbols: {pw_policy.require_symbols}")
        print(f"\trequire_uppercase_characters: {pw_policy.require_uppercase_characters}")
        printed = True
    except ClientError as error:
        if error.response['Error']['Code'] == 'NoSuchEntity':
            print("The account does not have a password policy set.")
        else:
            logger.exception("Couldn't get account password policy.")
            raise
    else:
        return printed
```

- For API details, see [GetAccountPasswordPolicy](#) in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
# Prints the password policy for the account.
def print_account_password_policy
  policy = @iam_resource.account_password_policy
  policy.load
  puts("The account password policy is:")
```

```
    puts(policy.data.to_h)
rescue Aws::Errors::ServiceError => e
  if e.code == "NoSuchEntity"
    puts("The account does not have a password policy.")
  else
    puts("Couldn't print the account password policy. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
end
```

- For API details, see [GetAccountPasswordPolicy](#) in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
pub async fn get_account_password_policy(
    client: &iamClient,
) -> Result<GetAccountPasswordPolicyOutput,
SdkError<GetAccountPasswordPolicyError>> {
    let response = client.get_account_password_policy().send().await?;

    Ok(response)
}
```

- For API details, see [GetAccountPasswordPolicy](#) in *AWS SDK for Rust API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

List SAML providers for IAM using an AWS SDK

The following code examples show how to list SAML providers for IAM.

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// List SAML authentication providers.
/// </summary>
```

```
/// <returns>A list of SAML providers.</returns>
public async Task<List<SAMLProviderListEntry>> ListSAMLProvidersAsync()
{
    var response = await _IAMService.ListSAMLProvidersAsync(new
ListSAMLProvidersRequest());
    return response.SAMLProviderList;
}
```

- For API details, see [ListSAMLProviders](#) in *AWS SDK for .NET API Reference*.

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// AccountWrapper encapsulates AWS Identity and Access Management (IAM) account
// actions
// used in the examples.
// It contains an IAM service client that is used to perform account actions.
type AccountWrapper struct {
    IamClient *iam.Client
}

// ListSAMLProviders gets the SAML providers for the account.
func (wrapper AccountWrapper) ListSAMLProviders() ([]types.SAMLProviderListEntry,
error) {
var providers []types.SAMLProviderListEntry
result, err := wrapper.IamClient.ListSAMLProviders(context.TODO(),
&iam.ListSAMLProvidersInput{})
if err != nil {
    log.Printf("Couldn't list SAML providers. Here's why: %v\n", err)
} else {
    providers = result.SAMLProviderList
}
return providers, err
}
```

- For API details, see [ListSAMLProviders](#) in *AWS SDK for Go API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

List the SAML providers.

```
import { ListSAMLProvidersCommand, IAMClient } from "@aws-sdk/client-iam";
```

```
const client = new IAMClient({});

export const listSamlProviders = async () => {
    const command = new ListSAMLProvidersCommand({});

    const response = await client.send(command);
    console.log(response);
    return response;
};
```

- For API details, see [ListSAMLProviders](#) in *AWS SDK for JavaScript API Reference*.

PHP

SDK for PHP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
$uuid = uniqid();
 getService();

 public function listSAMLProviders()
{
    return $this->iامClient->listSAMLProviders();
}
```

- For API details, see [ListSAMLProviders](#) in *AWS SDK for PHP API Reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
def list_saml_providers(count):
    """
    Lists the SAML providers for the account.

    :param count: The maximum number of providers to list.
    """
    try:
        found = 0
        for provider in iam.saml_providers.limit(count):
            logger.info('Got SAML provider %s.', provider.arn)
            found += 1
        if found == 0:
            logger.info("Your account has no SAML providers.")
    except ClientError:
        logger.exception("Couldn't list SAML providers.")
        raise
```

- For API details, see [ListSAMLProviders](#) in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
# Lists up to a specified number of SAML providers for the account.  
#  
# @param count [Integer] The maximum number of providers to list.  
def list_saml_providers(count)  
    @iam_resource.saml_providers.limit(count).each do |provider|  
        puts("\t#{provider.arn}")  
    end  
rescue Aws::Errors::ServiceError => e  
    puts("Couldn't list SAML providers. Here's why:")  
    puts("\t#{e.code}: #{e.message}")  
    raise  
end
```

- For API details, see [ListSAMLProviders](#) in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
pub async fn list_saml_providers(  
    client: &Client,  
) -> Result<ListSamlProvidersOutput, SdkError<ListSAMLProvidersError>> {  
    let response = client.list_saml_providers().send().await?  
  
    Ok(response)  
}
```

- For API details, see [ListSAMLProviders](#) in *AWS SDK for Rust API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

List a user's IAM access keys using an AWS SDK

The following code examples show how to list a user's IAM access keys.

Warning

To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as [AWS IAM Identity Center \(successor to AWS Single Sign-On\)](#).

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Manage access keys \(p. 980\)](#)

Bash

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
    printf "%s\n" "$*" 2>&1
}

#####
# function iam_list_access_keys
#
# This function lists the access keys for the specified user.
#
# Parameters:
#     -u user_name -- The name of the IAM user.
#
# Returns:
#     access_key_ids
#     And:
#         0 - If successful.
#         1 - If it fails.
#####
function iam_list_access_keys() {

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_list_access_keys"
        echo "Lists the AWS Identity and Access Management (IAM) access key IDs for the"
        echo "specified user."
        echo "  -u user_name  The name of the IAM user."
        echo ""
    }

    local user_name response
    local option OPTARG # Required to use getopt command in a function.
    # Retrieve the calling parameters.
    while getopts "u:h" option; do
        case "${option}" in
            u) user_name="${OPTARG}" ;;
            h)
                usage
                return 0
                ;;
        esac
    done
}
```

```
\?)  
    echo "Invalid parameter"  
    usage  
    return 1  
;;  
esac  
done  
export OPTIND=1  
  
if [[ -z "$user_name" ]]; then  
    errecho "ERROR: You must provide a username with the -u parameter."  
    usage  
    return 1  
fi  
  
response=$(aws iam list-access-keys \  
    --user-name "$user_name" \  
    --output text \  
    --query 'AccessKeyMetadata[].AccessKeyId')  
  
local error_code=${?}  
  
if [[ $error_code -ne 0 ]]; then  
    aws_cli_error_log $error_code  
    errecho "ERROR: AWS reports list-access-keys operation failed.$response"  
    return 1  
fi  
  
echo "$response"  
  
return 0  
}
```

- For API details, see [ListAccessKeys](#) in *AWS CLI Command Reference*.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
bool AwsDoc::IAM::listAccessKeys(const Aws::String &userName,  
                                 const Aws::Client::ClientConfiguration  
&clientConfig) {  
    Aws::IAM::IAMClient iam(clientConfig);  
    Aws::IAM::Model::ListAccessKeysRequest request;  
    request.SetUserName(userName);  
  
    bool done = false;  
    bool header = false;  
    while (!done) {  
        auto outcome = iam.ListAccessKeys(request);  
        if (!outcome.IsSuccess()) {  
            std::cerr << "Failed to list access keys for user " << userName  
                  << ":" << outcome.GetError().GetMessage() << std::endl;  
        }  
        return false;  
    }  
  
    if (!header) {
```

```

        std::cout << std::left << std::setw(32) << "UserName" <<
            std::setw(30) << "KeyID" << std::setw(20) << "Status" <<
            std::setw(20) << "CreateDate" << std::endl;
        header = true;
    }

    const auto &keys = outcome.GetResult().GetAccessKeyMetadata();
    const Aws::String DATE_FORMAT = "%Y-%m-%d";

    for (const auto &key: keys) {
        Aws::String statusString =
            Aws::IAM::Model::StatusTypeMapper::GetNameForStatusType(
                key.GetStatus());
        std::cout << std::left << std::setw(32) << key.GetUserName() <<
            std::setw(30) << key.GetAccessKeyId() << std::setw(20) <<
            statusString << std::setw(20) <<
            key.GetCreateDate().ToGmtString(DATE_FORMAT.c_str()) <<
        std::endl;
    }

    if (outcome.GetResult().GetIsTruncated()) {
        request.SetMarker(outcome.GetResult().GetMarker());
    }
    else {
        done = true;
    }
}

return true;
}

```

- For API details, see [ListAccessKeys](#) in *AWS SDK for C++ API Reference*.

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```

// UserWrapper encapsulates user actions used in the examples.
// It contains an IAM service client that is used to perform user actions.
type UserWrapper struct {
    iam.Client
}

// ListAccessKeys lists the access keys for the specified user.
func (wrapper UserWrapper) ListAccessKeys(userName string)
    ([]types.AccessKeyMetadata, error) {
    var keys []types.AccessKeyMetadata
    result, err := wrapper.IamClient.ListAccessKeys(context.TODO(),
    &iam.ListAccessKeysInput{
        UserName: aws.String(userName),
    })
    if err != nil {
        log.Printf("Couldn't list access keys for user %v. Here's why: %v\n", userName,
        err)
    } else {

```

```
        keys = result.AccessKeyMetadata
    }
    return keys, err
}
```

- For API details, see [ListAccessKeys](#) in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public static void listKeys( IamClient iam, String userName ){

    try {
        boolean done = false;
        String newMarker = null;

        while (!done) {
            ListAccessKeysResponse response;

            if(newMarker == null) {
                ListAccessKeysRequest request = ListAccessKeysRequest.builder()
                    .userName(userName)
                    .build();

                response = iam.listAccessKeys(request);
            } else {
                ListAccessKeysRequest request = ListAccessKeysRequest.builder()
                    .userName(userName)
                    .marker(newMarker)
                    .build();

                response = iam.listAccessKeys(request);
            }

            for (AccessKeyMetadata metadata : response.accessKeyMetadata()) {
                System.out.format("Retrieved access key %s",
                    metadata.accessKeyId());
            }

            if (!response.isTruncated()) {
                done = true;
            } else {
                newMarker = response.marker();
            }
        }
    } catch (IamException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

- For API details, see [ListAccessKeys](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

List the access keys.

```
import { ListAccessKeysCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/index.html#paginator} functions to simplify this.
 *
 * @param {string} userName
 */
export async function* listAccessKeys(userName) {
    const command = new ListAccessKeysCommand({
        MaxItems: 5,
        UserName: userName,
    });

    /**
     * @type {import("@aws-sdk/client-iam").ListAccessKeysCommandOutput | undefined}
     */
    let response = await client.send(command);

    while (response?.AccessKeyMetadata?.length) {
        for (const key of response.AccessKeyMetadata) {
            yield key;
        }

        if (response.IsTruncated) {
            response = await client.send(
                new ListAccessKeysCommand({
                    Marker: response.Marker,
                })
            );
        } else {
            break;
        }
    }
}
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [ListAccessKeys](#) in [AWS SDK for JavaScript API Reference](#).

SDK for JavaScript (v2)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// Load the AWS SDK for Node.js
var AWS = require('aws-sdk');
// Set the region
AWS.config.update({region: 'REGION'});
```

```
// Create the IAM service object
var iam = new AWS.IAM({apiVersion: '2010-05-08'});

var params = {
  MaxItems: 5,
  UserName: 'IAM_USER_NAME'
};

iam.listAccessKeys(params, function(err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [ListAccessKeys](#) in [AWS SDK for JavaScript API Reference](#).

Kotlin

SDK for Kotlin

Note

This is prerelease documentation for a feature in preview release. It is subject to change.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
suspend fun listKeys(userNameVal: String?) {

    val request = ListAccessKeysRequest {
        userName = userNameVal
    }
    IamClient { region = "AWS_GLOBAL" }.use { iamClient ->
        val response = iamClient.listAccessKeys(request)
        response.accessKeyMetadata?.forEach { md ->
            println("Retrieved access key ${md.accessKeyId}")
        }
    }
}
```

- For API details, see [ListAccessKeys](#) in [AWS SDK for Kotlin API reference](#).

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
def list_keys(user_name):
    """
    Lists the keys owned by the specified user.

```

```
:param user_name: The name of the user.
:return: The list of keys owned by the user.
"""
try:
    keys = list(iam.User(user_name).access_keys.all())
    logger.info("Got %s access keys for %s.", len(keys), user_name)
except ClientError:
    logger.exception("Couldn't get access keys for %s.", user_name)
else:
    return keys
```

- For API details, see [ListAccessKeys](#) in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
# Deletes a user. If the user has inline policies or access keys, they are
deleted
# before the user is deleted.
#
# @param user [Aws::IAM::User] The user to delete.
def delete_user(user)
    user.policies.each do |policy|
        name = policy.name
        policy.delete
        puts("Deleted user policy #{name}.")
    end
    user.access_keys.each do |key|
        key.delete
        puts("Deleted access key for user #{user.name}.")
    end
    name = user.name
    user.delete
    puts("Deleted user #{name}.")
rescue Aws::Errors::ServiceError => e
    puts("Couldn't detach policies and delete user #{user.name}. Here's why:")
    puts("\t#{e.code}: #{e.message}")
end
```

- For API details, see [ListAccessKeys](#) in *AWS SDK for Ruby API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

List IAM account aliases using an AWS SDK

The following code examples show how to list IAM account aliases.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Manage your account \(p. 989\)](#)

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
bool AwsDoc::IAM::listAccountAliases(const Aws::Client::ClientConfiguration &clientConfig) {
    Aws::IAM::IAMClient iam(clientConfig);
    Aws::IAM::Model::ListAccountAliasesRequest request;

    bool done = false;
    bool header = false;
    while (!done) {
        auto outcome = iam.ListAccountAliases(request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to list account aliases: " <<
                outcome.GetError().GetMessage() << std::endl;
            return false;
        }

        const auto &aliases = outcome.GetResult().GetAccountAliases();
        if (!header) {
            if (aliases.size() == 0) {
                std::cout << "Account has no aliases" << std::endl;
                break;
            }
            std::cout << std::left << std::setw(32) << "Alias" << std::endl;
            header = true;
        }

        for (const auto &alias: aliases) {
            std::cout << std::left << std::setw(32) << alias << std::endl;
        }

        if (outcome.GetResult().GetIsTruncated()) {
            request.SetMarker(outcome.GetResult().GetMarker());
        }
        else {
            done = true;
        }
    }

    return true;
}
```

- For API details, see [ListAccountAliases](#) in *AWS SDK for C++ API Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public static void listAliases(IamClient iam) {  
  
    try {  
        ListAccountAliasesResponse response = iam.listAccountAliases();  
        for (String alias : response.accountAliases()) {  
            System.out.printf("Retrieved account alias %s", alias);  
        }  
  
    } catch (IamException e) {  
        System.err.println(e.awsErrorDetails().errorMessage());  
        System.exit(1);  
    }  
}
```

- For API details, see [ListAccountAliases](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

List the account aliases.

```
import { ListAccountAliasesCommand, IAMClient } from "@aws-sdk/client-iam";  
  
const client = new IAMClient({});  
  
/**  
 * A generator function that handles paginated results.  
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/index.html#index.html#paginators | paginator} functions to simplify this.  
 */  
export async function* listAccountAliases() {  
    const command = new ListAccountAliasesCommand({ MaxItems: 5 });  
  
    let response = await client.send(command);  
  
    while (response.AccountAliases?.length) {  
        for (const alias of response.AccountAliases) {  
            yield alias;  
        }  
  
        if (response.IsTruncated) {  
            response = await client.send(  
                new ListAccountAliasesCommand({  
                    Marker: response.Marker,  
                    MaxItems: 5,  
                })  
            );  
        } else {  
            break;  
        }  
    }  
}
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).

- For API details, see [ListAccountAliases](#) in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// Load the AWS SDK for Node.js
var AWS = require('aws-sdk');
// Set the region
AWS.config.update({region: 'REGION'});

// Create the IAM service object
var iam = new AWS.IAM({apiVersion: '2010-05-08'});

iam.listAccountAliases({MaxItems: 10}, function(err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [ListAccountAliases](#) in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

Note

This is prerelease documentation for a feature in preview release. It is subject to change.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
suspend fun listAliases() {

    IamClient { region = "AWS_GLOBAL" }.use { iamClient ->
        val response = iamClient.listAccountAliases(ListAccountAliasesRequest {})
        response.accountAliases?.forEach { alias ->
            println("Retrieved account alias $alias")
        }
    }
}
```

- For API details, see [ListAccountAliases](#) in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
def list_aliases():
    """
    Gets the list of aliases for the current account. An account has at most one
    alias.

    :return: The list of aliases for the account.
    """
    try:
        response = iam.meta.client.list_account_aliases()
        aliases = response['AccountAliases']
        if len(aliases) > 0:
            logger.info("Got aliases for your account: %s.", ','.join(aliases))
        else:
            logger.info("Got no aliases for your account.")
    except ClientError:
        logger.exception("Couldn't list aliases for your account.")
        raise
    else:
        return response['AccountAliases']
```

- For API details, see [ListAccountAliases](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

List IAM groups using an AWS SDK

The following code examples show how to list IAM groups.

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// List IAM groups.
/// </summary>
/// <returns>A list of IAM groups.</returns>
public async Task<List<Group>> ListGroupsAsync()
{
    var groupsPaginator = _IAMService.Paginator.ListGroups(new
ListGroupsRequest());
    var groups = new List<Group>();

    await foreach (var response in groupsPaginator.Responses)
    {
        groups.AddRange(response.Groups);
    }

    return groups;
}
```

- For API details, see [ListGroups](#) in *AWS SDK for .NET API Reference*.

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// GroupWrapper encapsulates AWS Identity and Access Management (IAM) group actions
// used in the examples.
// It contains an IAM service client that is used to perform group actions.
type GroupWrapper struct {
    IamClient *iam.Client
}

// ListGroups lists up to maxGroups number of groups.
func (wrapper GroupWrapper) ListGroups(maxGroups int32) ([]types.Group, error) {
    var groups []types.Group
    result, err := wrapper.IamClient.ListGroups(context.TODO(), &iam.ListGroupsInput{
        MaxItems: aws.Int32(maxGroups),
    })
    if err != nil {
        log.Printf("Couldn't list groups. Here's why: %v\n", err)
    } else {
        groups = result.Groups
    }
    return groups, err
}
```

- For API details, see [ListGroups](#) in *AWS SDK for Go API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

List the groups.

```
import { ListGroupsCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/index.html#paginator} functions to simplify this.
 */
export async function* listGroups() {
    const command = new ListGroupsCommand({
        MaxItems: 10,
    });
}
```

```
let response = await client.send(command);

while (response.Groups?.length) {
    for (const group of response.Groups) {
        yield group;
    }

    if (response.IsTruncated) {
        response = await client.send(
            new ListGroupsCommand({
                Marker: response.Marker,
                MaxItems: 10,
            })
        );
    } else {
        break;
    }
}
```

- For API details, see [ListGroups](#) in *AWS SDK for JavaScript API Reference*.

PHP

SDK for PHP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
$uuid = uniqid();
$service = new IAMService();

public function listGroups($pathPrefix = "", $marker = "", $maxItems = 0)
{
    $listGroupsArguments = [];
    if ($pathPrefix) {
        $listGroupsArguments["PathPrefix"] = $pathPrefix;
    }
    if ($marker) {
        $listGroupsArguments["Marker"] = $marker;
    }
    if ($maxItems) {
        $listGroupsArguments["MaxItems"] = $maxItems;
    }

    return $this->iamClient->listGroups($listGroupsArguments);
}
```

- For API details, see [ListGroups](#) in *AWS SDK for PHP API Reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
def list_groups(count):
    """
    Lists the specified number of groups for the account.

    :param count: The number of groups to list.
    """
    try:
        for group in iam.groups.limit(count):
            logger.info("Group: %s", group.name)
    except ClientError:
        logger.exception("Couldn't list groups for the account.")
        raise
```

- For API details, see [ListGroups](#) in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
# Lists up to a specified number of groups for the account.
#
# @param count [Integer] The maximum number of groups to list.
def list_groups(count)
    @iam_resource.groups.limit(count).each do |group|
        puts("\t#{group.name}")
    end
rescue Aws::Errors::ServiceError => e
    puts("Couldn't list groups for the account. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
end
```

- For API details, see [ListGroups](#) in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
pub async fn list_groups(
    client: &iamClient,
    path_prefix: Option<String>,
    marker: Option<String>,
    max_items: Option<i32>,
) -> Result<ListGroupsOutput, SdkError<ListGroupsError>> {
```

```
let response = client
    .list_groups()
    .set_path_prefix(path_prefix)
    .set_marker(marker)
    .set_max_items(max_items)
    .send()
    .await?;

Ok(response)
}
```

- For API details, see [ListGroups](#) in *AWS SDK for Rust API reference*.

Swift

SDK for Swift

Note

This is prerelease documentation for an SDK in preview release. It is subject to change.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public func listGroups() async throws -> [String] {
    var groupList: [String] = []
    var marker: String? = nil
    var isTruncated: Bool

    repeat {
        let input = ListGroupsInput(marker: marker)
        let output = try await client.listGroups(input: input)

        guard let groups = output.groups else {
            return groupList
        }

        for group in groups {
            if let name = group.groupName {
                groupList.append(name)
            }
        }
        marker = output.marker
        isTruncated = output.isTruncated
    } while isTruncated == true
    return groupList
}
```

- For API details, see [ListGroups](#) in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

List inline policies for an IAM role using an AWS SDK

The following code examples show how to list inline policies for an IAM role.

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// List IAM role policies.
/// </summary>
/// <param name="roleName">The IAM role for which to list IAM policies.</param>
/// <returns>A list of IAM policy names.</returns>
public async Task<List<string>> ListRolePoliciesAsync(string roleName)
{
    var listRolePoliciesPaginator = _IAMService.Paginator.ListRolePolicies(new
ListRolePoliciesRequest { RoleName = roleName });
    var policyNames = new List<string>();

    await foreach (var response in listRolePoliciesPaginator.Responses)
    {
        policyNames.AddRange(response.PolicyNames);
    }

    return policyNames;
}
```

- For API details, see [ListRolePolicies](#) in *AWS SDK for .NET API Reference*.

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
// It contains an IAM service client that is used to perform role actions.
type RoleWrapper struct {
    IamClient *iam.Client
}

// ListRolePolicies lists the inline policies for a role.
func (wrapper RoleWrapper) ListRolePolicies(roleName string) ([]string, error) {
    var policies []string
    result, err := wrapper.IamClient.ListRolePolicies(context.TODO(),
    &iam.ListRolePoliciesInput{
        RoleName: aws.String(roleName),
    })
    if err != nil {
        log.Printf("Couldn't list policies for role %v. Here's why: %v\n", roleName, err)
    } else {
        policies = result.PolicyNames
    }
    return policies, err
}
```

```
    }
    return policies, err
}
```

- For API details, see [ListRolePolicies](#) in *AWS SDK for Go API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

List the policies.

```
import { ListRolePoliciesCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/index.html#paginator} functions to simplify this.
 *
 * @param {string} roleName
 */
export async function* listRolePolicies(roleName) {
    const command = new ListRolePoliciesCommand({
        RoleName: roleName,
        MaxItems: 10,
    });

    let response = await client.send(command);

    while (response.PolicyNames?.length) {
        for (const policyName of response.PolicyNames) {
            yield policyName;
        }

        if (response.IsTruncated) {
            response = await client.send(
                new ListRolePoliciesCommand({
                    RoleName: roleName,
                    MaxItems: 10,
                    Marker: response.Marker,
                })
            );
        } else {
            break;
        }
    }
}
```

- For API details, see [ListRolePolicies](#) in *AWS SDK for JavaScript API Reference*.

PHP

SDK for PHP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
$uuid = uniqid();
$service = new IAMService();

    public function listRolePolicies($roleName, $marker = "", $maxItems = 0)
    {
        $listRolePoliciesArguments = ['RoleName' => $roleName];
        if ($marker) {
            $listRolePoliciesArguments['Marker'] = $marker;
        }
        if ($maxItems) {
            $listRolePoliciesArguments['MaxItems'] = $maxItems;
        }
        return $this->customWaiter(function () use ($listRolePoliciesArguments) {
            return $this->iamClient->listRolePolicies($listRolePoliciesArguments);
        });
    }
```

- For API details, see [ListRolePolicies](#) in *AWS SDK for PHP API Reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
def list_policies(role_name):
    """
    Lists inline policies for a role.

    :param role_name: The name of the role to query.
    """
    try:
        role = iam.Role(role_name)
        for policy in role.policies.all():
            logger.info("Got inline policy %s.", policy.name)
    except ClientError:
        logger.exception("Couldn't list inline policies for %s.", role_name)
        raise
```

- For API details, see [ListRolePolicies](#) in *AWS SDK for Python (Boto3) API Reference*.

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
pub async fn list_role_policies(
    client: &iامClient,
    role_name: &str,
    marker: Option<String>,
    max_items: Option<i32>,
) -> Result<ListRolePoliciesOutput, SdkError<ListRolePoliciesError>> {
    let response = client
        .list_role_policies()
        .role_name(role_name)
        .set_marker(marker)
        .set_max_items(max_items)
        .send()
        .await?;

    Ok(response)
}
```

- For API details, see [ListRolePolicies](#) in *AWS SDK for Rust API reference*.

Swift

SDK for Swift

Note

This is prerelease documentation for an SDK in preview release. It is subject to change.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public func listRolePolicies(role: String) async throws -> [String] {
    var policyList: [String] = []
    var marker: String? = nil
    var isTruncated: Bool

    repeat {
        let input = ListRolePoliciesInput(
            marker: marker,
            roleName: role
        )
        let output = try await client.listRolePolicies(input: input)

        guard let policies = output.policyNames else {
            return policyList
        }

        for policy in policies {
            policyList.append(policy)
        }
        marker = output.marker
        isTruncated = output.isTruncated
    } while isTruncated == true
    return policyList
}
```

- For API details, see [ListRolePolicies](#) in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

List inline IAM policies for a user using an AWS SDK

The following code example shows how to list inline IAM policies for a user.

Warning

To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as [AWS IAM Identity Center \(successor to AWS Single Sign-On\)](#).

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// UserWrapper encapsulates user actions used in the examples.  
// It contains an IAM service client that is used to perform user actions.  
type UserWrapper struct {  
    IamClient *iam.Client  
}  
  
  
// ListUserPolicies lists the inline policies for the specified user.  
func (wrapper UserWrapper) ListUserPolicies(userName string) ([]string, error) {  
    var policies []string  
    result, err := wrapper.IamClient.ListUserPolicies(context.TODO(),  
&iam.ListUserPoliciesInput{  
    UserName: aws.String(userName),  
})  
    if err != nil {  
        log.Printf("Couldn't list policies for user %v. Here's why: %v\n", userName, err)  
    } else {  
        policies = result.PolicyNames  
    }  
    return policies, err  
}
```

- For API details, see [ListUserPolicies](#) in *AWS SDK for Go API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

List IAM policies using an AWS SDK

The following code examples show how to list IAM policies.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Manage policies \(p. 982\)](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// List IAM policies.
/// </summary>
/// <returns>A list of the IAM policies.</returns>
public async Task<List<ManagedPolicy>> ListPoliciesAsync()
{
    var listPoliciesPaginator = _IAMService.Paginator.ListPolicies(new
ListPoliciesRequest());
    var policies = new List<ManagedPolicy>();

    await foreach (var response in listPoliciesPaginator.Responses)
    {
        policies.AddRange(response.Policies);
    }

    return policies;
}
```

- For API details, see [ListPolicies in AWS SDK for .NET API Reference](#).

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
bool AwsDoc::IAM::listPolicies(const Aws::Client::ClientConfiguration
&clientConfig) {
    const Aws::String DATE_FORMAT("%Y-%m-%d");
    Aws::IAM::IAMClient iam(clientConfig);
    Aws::IAM::Model::ListPoliciesRequest request;

    bool done = false;
    bool header = false;
    while (!done) {
        auto outcome = iam.ListPolicies(request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to list iam policies: " <<
                outcome.GetError().GetMessage() << std::endl;
            return false;
    }
}
```

```

        if (!header) {
            std::cout << std::left << std::setw(55) << "Name" <<
                std::setw(30) << std::setw(80) << "Arn" <<
                std::setw(64) << "Description" << std::setw(12) <<
                "CreateDate" << std::endl;
            header = true;
        }

        const auto &policies = outcome.GetResult().GetPolicies();
        for (const auto &policy: policies) {
            std::cout << std::left << std::setw(55) <<
                policy.GetPolicyName() << std::setw(30) <<
                policy.GetPolicyId() << std::setw(80) << policy.GetArn() <<
                std::setw(64) << policy.GetDescription() << std::setw(12) <<
                policy.GetCreateDate().ToGmtString(DATE_FORMAT.c_str()) <<
                std::endl;
        }

        if (outcome.GetResult().GetIsTruncated()) {
            request.SetMarker(outcome.GetResult().GetMarker());
        }
        else {
            done = true;
        }
    }

    return true;
}

```

- For API details, see [ListPolicies](#) in *AWS SDK for C++ API Reference*.

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```

// PolicyWrapper encapsulates AWS Identity and Access Management (IAM) policy
// actions
// used in the examples.
// It contains an IAM service client that is used to perform policy actions.
type PolicyWrapper struct {
    IamClient *iam.Client
}

// ListPolicies gets up to maxPolicies policies.
func (wrapper PolicyWrapper) ListPolicies(maxPolicies int32) ([]types.Policy,
    error) {
    var policies []types.Policy
    result, err := wrapper.IamClient.ListPolicies(context.TODO(),
        &iam.ListPoliciesInput{
            MaxItems: aws.Int32(maxPolicies),
        })
    if err != nil {
        log.Printf("Couldn't list policies. Here's why: %v\n", err)
    } else {

```

```
    policies = result.Policies
}
return policies, err
}
```

- For API details, see [ListPolicies](#) in *AWS SDK for Go API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

List the policies.

```
import { ListPoliciesCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/index.html#paginator} functions to simplify this.
 */
export async function* listPolicies() {
    const command = new ListPoliciesCommand({
        MaxItems: 10,
        OnlyAttached: false,
        // List only the customer managed policies in your Amazon Web Services account.
        Scope: "Local",
    });

    let response = await client.send(command);

    while (response.Policies?.length) {
        for (const policy of response.Policies) {
            yield policy;
        }

        if (response.IsTruncated) {
            response = await client.send(
                new ListPoliciesCommand({
                    Marker: response.Marker,
                    MaxItems: 10,
                    OnlyAttached: false,
                    Scope: "Local",
                })
            );
        } else {
            break;
        }
    }
}
```

- For API details, see [ListPolicies](#) in *AWS SDK for JavaScript API Reference*.

PHP

SDK for PHP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
$uuid = uniqid();
$service = new IAMService();

    public function listPolicies($pathPrefix = "", $marker = "", $maxItems = 0)
    {
        $listPoliciesArguments = [];
        if ($pathPrefix) {
            $listPoliciesArguments["PathPrefix"] = $pathPrefix;
        }
        if ($marker) {
            $listPoliciesArguments["Marker"] = $marker;
        }
        if ($maxItems) {
            $listPoliciesArguments["MaxItems"] = $maxItems;
        }

        return $this->iamClient->listPolicies($listPoliciesArguments);
    }
}
```

- For API details, see [ListPolicies](#) in *AWS SDK for PHP API Reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
def list_policies(scope):
    """
    Lists the policies in the current account.

    :param scope: Limits the kinds of policies that are returned. For example,
                  'Local' specifies that only locally managed policies are
                  returned.
    :return: The list of policies.
    """
    try:
        policies = list(iam.policies.filter(Scope=scope))
        logger.info("Got %s policies in scope '%s'.", len(policies), scope)
    except ClientError:
        logger.exception("Couldn't get policies for scope '%s'.", scope)
        raise
    else:
        return policies
```

- For API details, see [ListPolicies](#) in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
# Lists up to a specified number of policies in the account.  
#  
# @param count [Integer] The maximum number of policies to list.  
# @return [Array] The Amazon Resource Names (ARNs) of the policies listed.  
def list_policies(count)  
    policy_arns = []  
    @iam_resource.policies.limit(count).each_with_index do |policy, index|  
        puts("\t#{index + 1}: #{policy.policy_name}: #{policy.arn}")  
        policy_arns.append(policy.arn)  
    end  
    rescue Aws::Errors::ServiceError => e  
        puts("Couldn't list policies for the account. Here's why:")  
        puts("\t#{e.code}: #{e.message}")  
        raise  
    else  
        policy_arns  
    end
```

- For API details, see [ListPolicies](#) in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
pub async fn list_policies(  
    client: iamClient,  
    path_prefix: String,  
) -> Result<Vec<String>, SdkError<ListPoliciesError>> {  
    let mut list_policies = client  
        .list_policies()  
        .path_prefix(path_prefix)  
        .scope(PolicyScopeType::Local)  
        .into_paginator()  
        .send();  
  
    let mut v = Vec::new();  
  
    while let Some(list_policies_output) = list_policies.next().await {  
        match list_policies_output {  
            Ok(list_policies) => {  
                if let Some(policies) = list_policies.policies() {  
                    for policy in policies {  
                        let policy_name = policy  
                            .policy_name()  
                            .unwrap_or("Missing policy name.")
```

```
        .to_string();
        println!("{}", policy_name);
        v.push(policy_name);
    }
}

Err(err) => return Err(err),
}
Ok(v)
}
```

- For API details, see [ListPolicies](#) in *AWS SDK for Rust API reference*.

Swift

SDK for Swift

Note

This is prerelease documentation for an SDK in preview release. It is subject to change.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public func listPolicies() async throws -> [MyPolicyRecord] {
    var policyList: [MyPolicyRecord] = []
    var marker: String? = nil
    var isTruncated: Bool

    repeat {
        let input = ListPoliciesInput(marker: marker)
        let output = try await client.listPolicies(input: input)

        guard let policies = output.policies else {
            return policyList
        }

        for policy in policies {
            guard let name = policy.policyName,
                  let id = policy.policyId,
                  let arn = policy.arn else {
                throw ServiceHandlerError.noSuchPolicy
            }
            policyList.append(MyPolicyRecord(name: name, id: id, arn: arn))
        }
        marker = output.marker
        isTruncated = output.isTruncated
    } while isTruncated == true
    return policyList
}
```

- For API details, see [ListPolicies](#) in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

List policies attached to an IAM role using an AWS SDK

The following code examples show how to list policies attached to an IAM role.

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// List the IAM role policies that are attached to an IAM role.
/// </summary>
/// <param name="roleName">The IAM role to list IAM policies for.</param>
/// <returns>A list of the IAM policies attached to the IAM role.</returns>
public async Task<List<AttachedPolicyType>>
ListAttachedRolePoliciesAsync(string roleName)
{
    var attachedPolicies = new List<AttachedPolicyType>();
    var attachedRolePoliciesPaginator =
_IAMService.Paginator.ListAttachedRolePolicies(new
ListAttachedRolePoliciesRequest { RoleName = roleName });

    await foreach (var response in attachedRolePoliciesPaginator.Responses)
    {
        attachedPolicies.AddRange(response.AttachedPolicies);
    }

    return attachedPolicies;
}
```

- For API details, see [ListAttachedRolePolicies](#) in *AWS SDK for .NET API Reference*.

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
// It contains an IAM service client that is used to perform role actions.
type RoleWrapper struct {
    IamClient *iam.Client
}

// ListAttachedRolePolicies lists the policies that are attached to the specified
// role.
func (wrapper RoleWrapper) ListAttachedRolePolicies(roleName string)
([]types.AttachedPolicy, error) {
    var policies []types.AttachedPolicy
```

```
result, err := wrapper.IamClient.ListAttachedRolePolicies(context.TODO(),
&iam.ListAttachedRolePoliciesInput{
    RoleName: aws.String(roleName),
})
if err != nil {
    log.Printf("Couldn't list attached policies for role %v. Here's why: %v\n",
    roleName, err)
} else {
    policies = result.AttachedPolicies
}
return policies, err
}
```

- For API details, see [ListAttachedRolePolicies](#) in *AWS SDK for Go API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

List the policies that are attached to a role.

```
import {
    ListAttachedRolePoliciesCommand,
    IAMClient,
} from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/index.html#paginator} functions to simplify this.
 * @param {string} roleName
 */
export async function* listAttachedRolePolicies(roleName) {
    const command = new ListAttachedRolePoliciesCommand({
        RoleName: roleName,
    });

    let response = await client.send(command);

    while (response.AttachedPolicies?.length) {
        for (const policy of response.AttachedPolicies) {
            yield policy;
        }

        if (response.IsTruncated) {
            response = await client.send(
                new ListAttachedRolePoliciesCommand({
                    RoleName: roleName,
                    Marker: response.Marker,
                })
            );
        } else {
            break;
        }
    }
}
```

```
}
```

- For API details, see [ListAttachedRolePolicies](#) in *AWS SDK for JavaScript API Reference*.

PHP

SDK for PHP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
$uuid = uniqid();
$service = new IAMService();

    public function listAttachedRolePolicies($roleName, $pathPrefix = "", $marker =
    "", $maxItems = 0)
    {
        $listAttachRolePoliciesArguments = ['RoleName' => $roleName];
        if ($pathPrefix) {
            $listAttachRolePoliciesArguments['PathPrefix'] = $pathPrefix;
        }
        if ($marker) {
            $listAttachRolePoliciesArguments['Marker'] = $marker;
        }
        if ($maxItems) {
            $listAttachRolePoliciesArguments['MaxItems'] = $maxItems;
        }
        return $this->iamClient-
>listAttachedRolePolicies($listAttachRolePoliciesArguments);
    }
```

- For API details, see [ListAttachedRolePolicies](#) in *AWS SDK for PHP API Reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
def list_attached_policies(role_name):
    """
    Lists policies attached to a role.

    :param role_name: The name of the role to query.
    """
    try:
        role = iam.Role(role_name)
        for policy in role.attached_policies.all():
            logger.info("Got policy %s.", policy.arn)
    except ClientError:
        logger.exception("Couldn't list attached policies for %s.", role_name)
        raise
```

- For API details, see [ListAttachedRolePolicies](#) in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
# Deletes a role. If the role has policies attached, they are detached and
# deleted before the role is deleted.
#
# @param role [Aws::IAM::Role] The role to delete.
def delete_role(role)
    role.attached_policies.each do |policy|
        name = policy.policy_name
        policy.detach_role(role_name: role.name)
        policy.delete
        puts("Deleted policy #{name}.")
    end
    name = role.name
    role.delete
    puts("Deleted role #{name}.")
rescue Aws::Errors::ServiceError => e
    puts("Couldn't detach policies and delete role #{role.name}. Here's why:")
    puts("  +#{e.code}: #{e.message}")
    raise
end
```

- For API details, see [ListAttachedRolePolicies](#) in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
pub async fn list_attached_role_policies(
    client: &iamClient,
    role_name: String,
    path_prefix: Option<String>,
    marker: Option<String>,
    max_items: Option<i32>,
) -> Result<ListAttachedRolePoliciesOutput,
SdkError<ListAttachedRolePoliciesError>> {
    let response = client
        .list_attached_role_policies()
        .role_name(role_name)
        .set_path_prefix(path_prefix)
        .set_marker(marker)
        .set_max_items(max_items)
        .send()
        .await?;

    Ok(response)
```

```
}
```

- For API details, see [ListAttachedRolePolicies](#) in *AWS SDK for Rust API reference*.

Swift

SDK for Swift

Note

This is prerelease documentation for an SDK in preview release. It is subject to change.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// Returns a list of AWS Identity and Access Management (IAM) policies
/// that are attached to the role.
///
/// - Parameter role: The IAM role to return the policy list for.
///
/// - Returns: An array of `IAMClientTypes.AttachedPolicy` objects
///   describing each managed policy that's attached to the role.
public func listAttachedRolePolicies(role: String) async throws ->
    [IAMClientTypes.AttachedPolicy] {
    var policyList: [IAMClientTypes.AttachedPolicy] = []
    var marker: String? = nil
    var isTruncated: Bool

    repeat {
        let input = ListAttachedRolePoliciesInput(
            marker: marker,
            roleName: role
        )
        let output = try await client.listAttachedRolePolicies(input: input)

        guard let attachedPolicies = output.attachedPolicies else {
            return policyList
        }

        for attachedPolicy in attachedPolicies {
            policyList.append(attachedPolicy)
        }
        marker = output.marker
        isTruncated = output.isTruncated
    } while isTruncated == true
    return policyList
}
```

- For API details, see [ListAttachedRolePolicies](#) in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

List IAM roles using an AWS SDK

The following code examples show how to list IAM roles.

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// List IAM roles.
/// </summary>
/// <returns>A list of IAM roles.</returns>
public async Task<List<Role>> ListRolesAsync()
{
    var listRolesPaginator = _IAMService.Paginator.ListRoles(new
ListRolesRequest());
    var roles = new List<Role>();

    await foreach (var response in listRolesPaginator.Responses)
    {
        roles.AddRange(response.Roles);
    }

    return roles;
}
```

- For API details, see [ListRoles](#) in *AWS SDK for .NET API Reference*.

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
// It contains an IAM service client that is used to perform role actions.
type RoleWrapper struct {
    IamClient *iam.Client
}

// ListRoles gets up to maxRoles roles.
func (wrapper RoleWrapper) ListRoles(maxRoles int32) ([]types.Role, error) {
    var roles []types.Role
    result, err := wrapper.IamClient.ListRoles(context.TODO(),
        &iam.ListRolesInput{MaxItems: aws.Int32(maxRoles)},
    )
    if err != nil {
        log.Printf("Couldn't list roles. Here's why: %v\n", err)
    } else {
        roles = result.Roles
    }
    return roles, err
}
```

- For API details, see [ListRoles](#) in *AWS SDK for Go API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

List the roles.

```
import { ListRolesCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/index.html#paginator} functions to simplify this.
 */
export async function* listRoles() {
    const command = new ListRolesCommand({
        MaxItems: 10,
    });

    /**
     * @type {import("@aws-sdk/client-iam").ListRolesCommandOutput | undefined}
     */
    let response = await client.send(command);

    while (response?.Roles?.length) {
        for (const role of response.Roles) {
            yield role;
        }

        if (response.IsTruncated) {
            response = await client.send(
                new ListRolesCommand({
                    Marker: response.Marker,
                })
            );
        } else {
            break;
        }
    }
}
```

- For API details, see [ListRoles](#) in *AWS SDK for JavaScript API Reference*.

PHP

SDK for PHP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
$uuid = uniqid();
$service = new IAMService();

/**
 * @param string $pathPrefix
 * @param string $marker
 * @param int $maxItems
 * @return Result
 * $roles = $service->listRoles();
 */
public function listRoles($pathPrefix = "", $marker = "", $maxItems = 0)
{
    $listRolesArguments = [];
    if ($pathPrefix) {
        $listRolesArguments["PathPrefix"] = $pathPrefix;
    }
    if ($marker) {
        $listRolesArguments["Marker"] = $marker;
    }
    if ($maxItems) {
        $listRolesArguments["MaxItems"] = $maxItems;
    }
    return $this->iامClient->listRoles($listRolesArguments);
}
```

- For API details, see [ListRoles](#) in *AWS SDK for PHP API Reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
def list_roles(count):
    """
    Lists the specified number of roles for the account.

    :param count: The number of roles to list.
    """
    try:
        roles = list(iام.roles.limit(count=count))
        for role in roles:
            logger.info("Role: %s", role.name)
    except ClientError:
        logger.exception("Couldn't list roles for the account.")
        raise
    else:
        return roles
```

- For API details, see [ListRoles](#) in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
# Lists up to a specified number of roles for the account.  
#  
# @param count [Integer] The maximum number of roles to list.  
# @return [Array] The names of the listed roles.  
def list_roles(count)  
  role_names = []  
  @iam_resource.roles.limit(count).each_with_index do |role, index|  
    puts("\t#{index + 1}: #{role.name}")  
    role_names.append(role.name)  
  end  
  rescue Aws::Errors::ServiceError => e  
    puts("Couldn't list roles for the account. Here's why:")  
    puts("\t#{e.code}: #{e.message}")  
    raise  
  else  
    role_names  
  end
```

- For API details, see [ListRoles](#) in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
pub async fn list_roles(  
    client: &iamClient,  
    path_prefix: Option<String>,  
    marker: Option<String>,  
    max_items: Option<i32>,  
) -> Result<ListRolesOutput, SdkError<ListRolesError>> {  
    let response = client  
        .list_roles()  
        .set_path_prefix(path_prefix)  
        .set_marker(marker)  
        .set_max_items(max_items)  
        .send()  
        .await?;  
    Ok(response)  
}
```

- For API details, see [ListRoles](#) in *AWS SDK for Rust API reference*.

Swift

SDK for Swift

Note

This is prerelease documentation for an SDK in preview release. It is subject to change.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public func listRoles() async throws -> [String] {
    var roleList: [String] = []
    var marker: String? = nil
    var isTruncated: Bool

    repeat {
        let input = ListRolesInput(marker: marker)
        let output = try await client.listRoles(input: input)

        guard let roles = output.roles else {
            return roleList
        }

        for role in roles {
            if let name = role.roleName {
                roleList.append(name)
            }
        }
        marker = output.marker
        isTruncated = output.isTruncated
    } while isTruncated == true
    return roleList
}
```

- For API details, see [ListRoles](#) in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

List IAM server certificates using an AWS SDK

The following code examples show how to list IAM server certificates.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
bool AwsDoc::IAM::listServerCertificates(
    const Aws::Client::ClientConfiguration &clientConfig) {
    const Aws::String DATE_FORMAT = "%Y-%m-%d";

    Aws::IAM::IAMClient iam(clientConfig);
    Aws::IAM::Model::ListServerCertificatesRequest request;
```

```

bool done = false;
bool header = false;
while (!done) {
    auto outcome = iam.ListServerCertificates(request);
    if (!outcome.IsSuccess()) {
        std::cerr << "Failed to list server certificates: " <<
            outcome.GetError().GetMessage() << std::endl;
        return false;
    }

    if (!header) {
        std::cout << std::left << std::setw(55) << "Name" <<
            std::setw(30) << std::setw(80) << "Arn" <<
            std::setw(14) << "UploadDate" << std::setw(14) <<
            "ExpirationDate" << std::endl;
        header = true;
    }

    const auto &certificates =
        outcome.GetResult().GetServerCertificateMetadataList();

    for (const auto &certificate: certificates) {
        std::cout << std::left << std::setw(55) <<
            certificate.GetServerCertificateName() << std::setw(30) <<
            certificate.GetServerCertificateId() << std::setw(80) <<
            certificate.GetArn() << std::setw(14) <<
            certificate.GetUploadDate().ToGmtString(DATE_FORMAT.c_str())
<<
            std::setw(14) <<
            certificate.GetExpiration().ToGmtString(DATE_FORMAT.c_str())
<<
            std::endl;
    }

    if (outcome.GetResult().GetIsTruncated()) {
        request.SetMarker(outcome.GetResult().GetMarker());
    }
    else {
        done = true;
    }
}

return true;
}

```

- For API details, see [ListServerCertificates](#) in *AWS SDK for C++ API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

List the certificates.

```

import { ListServerCertificatesCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**

```

```
* A generator function that handles paginated results.
* The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/index.html#paginator} functions to simplify this.
*/
export async function* listServerCertificates() {
    const command = new ListServerCertificatesCommand({});
    let response = await client.send(command);

    while (response.ServerCertificateMetadataList?.length) {
        for await (const cert of response.ServerCertificateMetadataList) {
            yield cert;
        }

        if (response.IsTruncated) {
            response = await client.send(new ListServerCertificatesCommand({}));
        } else {
            break;
        }
    }
}
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [ListServerCertificates in AWS SDK for JavaScript API Reference](#).

SDK for JavaScript (v2)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// Load the AWS SDK for Node.js
var AWS = require('aws-sdk');
// Set the region
AWS.config.update({region: 'REGION'});

// Create the IAM service object
var iam = new AWS.IAM({apiVersion: '2010-05-08'});

iam.listServerCertificates({}, function(err, data) {
    if (err) {
        console.log("Error", err);
    } else {
        console.log("Success", data);
    }
});
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [ListServerCertificates in AWS SDK for JavaScript API Reference](#).

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

List IAM users using an AWS SDK

The following code examples show how to list IAM users.

Warning

To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as [AWS IAM Identity Center \(successor to AWS Single Sign-On\)](#).

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Create read-only and read-write users \(p. 973\)](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// List IAM users.
/// </summary>
/// <returns>A list of IAM users.</returns>
public async Task<List<User>> ListUsersAsync()
{
    var listUsersPaginator = _IAMService.Paginator.ListUsers(new
ListUsersRequest());
    var users = new List<User>();

    await foreach (var response in listUsersPaginator.Responses)
    {
        users.AddRange(response.Users);
    }

    return users;
}
```

- For API details, see [ListUsers](#) in *AWS SDK for .NET API Reference*.

Bash

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
    printf "%s\n" "$*" 2>&1
}

#####
# function iam_list_users
```

```

#
# List the IAM users in the account.
#
# Returns:
#     The list of users names
#     And:
#         0 - If the user already exists.
#         1 - If the user doesn't exist.
#####
function iam_list_users() {
    local option OPTARG # Required to use getopts command in a function.
    local all_users error_code
    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_list_users"
        echo "Lists the AWS Identity and Access Management (IAM) user in the account."
        echo ""
    }

    # Retrieve the calling parameters.
    while getopts "h" option; do
        case "${option}" in
            h)
                usage
                return 0
            ;;
            \?)
                echo "Invalid parameter"
                usage
                return 1
            ;;
        esac
    done
    export OPTIND=1

    local response

    response=$(aws iam list-users \
        --output text \
        --query "Users[].UserName")
    error_code=${?}

    if [[ $error_code -ne 0 ]]; then
        aws_cli_error_log $error_code
        errecho "ERROR: AWS reports list-users operation failed.$response"
        return 1
    fi

    echo "$response"

    return 0
}

```

- For API details, see [ListUsers](#) in *AWS CLI Command Reference*.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```

bool AwsDoc::IAM::listUsers(const Aws::Client::ClientConfiguration &clientConfig) {
    const Aws::String DATE_FORMAT = "%Y-%m-%d";
    Aws::IAM::IAMClient iam(clientConfig);
    Aws::IAM::Model::ListUsersRequest request;

    bool done = false;
    bool header = false;
    while (!done) {
        auto outcome = iam.ListUsers(request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to list iam users:" <<
                outcome.GetError().GetMessage() << std::endl;
            return false;
        }

        if (!header) {
            std::cout << std::left << std::setw(32) << "Name" <<
                std::setw(30) << "ID" << std::setw(64) << "Arn" <<
                std::setw(20) << "CreateDate" << std::endl;
            header = true;
        }

        const auto &users = outcome.GetResult().GetUsers();
        for (const auto &user: users) {
            std::cout << std::left << std::setw(32) << user.GetUserName() <<
                std::setw(30) << user.GetUserId() << std::setw(64) <<
                user.GetArn() << std::setw(20) <<
                user.GetCreateDate().ToGmtString(DATE_FORMAT.c_str())
                << std::endl;
        }

        if (outcome.GetResult().GetIsTruncated()) {
            request.SetMarker(outcome.GetResult().GetMarker());
        }
        else {
            done = true;
        }
    }

    return true;
}

```

- For API details, see [ListUsers](#) in *AWS SDK for C++ API Reference*.

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```

// UserWrapper encapsulates user actions used in the examples.
// It contains an IAM service client that is used to perform user actions.
type UserWrapper struct {
    IamClient *iam.Client
}

```

```
// ListUsers gets up to maxUsers number of users.
func (wrapper UserWrapper) ListUsers(maxUsers int32) ([]types.User, error) {
    var users []types.User
    result, err := wrapper.IamClient.ListUsers(context.TODO(), &iam.ListUsersInput{
        MaxItems: aws.Int32(maxUsers),
    })
    if err != nil {
        log.Printf("Couldn't list users. Here's why: %v\n", err)
    } else {
        users = result.Users
    }
    return users, err
}
```

- For API details, see [ListUsers](#) in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public static void listAllUsers(IamClient iam ) {

    try {

        boolean done = false;
        String newMarker = null;

        while(!done) {
            ListUsersResponse response;
            if (newMarker == null) {
                ListUsersRequest request = ListUsersRequest.builder().build();
                response = iam.listUsers(request);
            } else {
                ListUsersRequest request = ListUsersRequest.builder()
                    .marker(newMarker)
                    .build();

                response = iam.listUsers(request);
            }

            for(User user : response.users()) {
                System.out.format("\n Retrieved user %s", user.userName());
                AttachedPermissionsBoundary permissionsBoundary =
user.permissionsBoundary();
                if (permissionsBoundary != null)
                    System.out.format("\n Permissions boundary details %s",
permissionsBoundary.permissionsBoundaryAsString());
            }

            if(!response.isTruncated()) {
                done = true;
            } else {
                newMarker = response.marker();
            }
        }
    } catch (IamException e) {
```

```
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

- For API details, see [ListUsers](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

List the users.

```
import { ListUsersCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

export const listUsers = async () => {
    const command = new ListUsersCommand({ MaxItems: 10 });

    const response = await client.send(command);
    response.Users?.forEach(({ UserName, CreateDate }) => {
        console.log(`#${UserName} created on: ${CreateDate}`);
    });
    return response;
};
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [ListUsers](#) in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// Load the AWS SDK for Node.js
var AWS = require('aws-sdk');
// Set the region
AWS.config.update({region: 'REGION'});

// Create the IAM service object
var iam = new AWS.IAM({apiVersion: '2010-05-08'});

var params = {
    MaxItems: 10
};

iam.listUsers(params, function(err, data) {
    if (err) {
        console.log("Error", err);
    } else {
        var users = data.Users || [];
        users.forEach(function(user) {
            console.log("User " + user.UserName + " created", user.CreateDate);
        });
    }
});
```

```
    });
});
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [ListUsers](#) in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

Note

This is prerelease documentation for a feature in preview release. It is subject to change.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
suspend fun listAllUsers() {

    IamClient { region = "AWS_GLOBAL" }.use { iamClient ->
        val response = iamClient.listUsers(ListUsersRequest { })
        response.users?.forEach { user ->
            println("Retrieved user ${user.userName}")
            val permissionsBoundary = user.permissionsBoundary
            if (permissionsBoundary != null)
                println("Permissions boundary details
${permissionsBoundary.permissionsBoundaryType}")
        }
    }
}
```

- For API details, see [ListUsers](#) in *AWS SDK for Kotlin API reference*.

PHP

SDK for PHP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
$uuid = uniqid();
 getService();

 public function listUsers($pathPrefix = "", $marker = "", $maxItems = 0)
{
    $listUsersArguments = [];
    if ($pathPrefix) {
        $listUsersArguments["PathPrefix"] = $pathPrefix;
    }
    if ($marker) {
        $listUsersArguments["Marker"] = $marker;
    }
    if ($maxItems) {
```

```
        $listUsersArguments["MaxItems"] = $maxItems;
    }

    return $this->iamClient->listUsers($listUsersArguments);
}
```

- For API details, see [ListUsers](#) in *AWS SDK for PHP API Reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
def list_users():
    """
    Lists the users in the current account.

    :return: The list of users.
    """
    try:
        users = list(iam.users.all())
        logger.info("Got %s users.", len(users))
    except ClientError:
        logger.exception("Couldn't get users.")
        raise
    else:
        return users
```

- For API details, see [ListUsers](#) in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
# Lists up to a specified number of users in the account.
#
# @param count [Integer] The maximum number of users to list.
def list_users(count)
    @iam_resource.users.limit(count).each do |user|
        puts("\t#{user.name}")
    end
rescue Aws::Errors::ServiceError => e
    puts("Couldn't list users for the account. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
end
```

- For API details, see [ListUsers](#) in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
pub async fn list_users(
    client: &iamClient,
    path_prefix: Option<String>,
    marker: Option<String>,
    max_items: Option<i32>,
) -> Result<ListUsersOutput, SdkError<ListUsersError>> {
    let response = client
        .list_users()
        .set_path_prefix(path_prefix)
        .set_marker(marker)
        .set_max_items(max_items)
        .send()
        .await?;
    Ok(response)
}
```

- For API details, see [ListUsers](#) in *AWS SDK for Rust API reference*.

Swift

SDK for Swift

Note

This is prerelease documentation for an SDK in preview release. It is subject to change.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public func listUsers() async throws -> [MyUserRecord] {
    var userList: [MyUserRecord] = []
    var marker: String? = nil
    var isTruncated: Bool

    repeat {
        let input = ListUsersInput(marker: marker)
        let output = try await client.listUsers(input: input)

        guard let users = output.users else {
            return userList
        }

        for user in users {
            if let id = user.userId, let name = user.userName {
                userList.append(MyUserRecord(id: id, name: name))
            }
        }
        marker = output.marker
    }
}
```

```
        isTruncated = output.isTruncated
    } while isTruncated == true
    return userList
}
```

- For API details, see [ListUsers](#) in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Remove an IAM user from a group using an AWS SDK

The following code example shows how to remove a user from an IAM group.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Create a group and add a user \(p. 874\)](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
///<summary>
/// Remove a user from an IAM group.
///</summary>
///<param name="userName">The username of the user to remove.</param>
///<param name="groupName">The name of the IAM group to remove the user
from.</param>
///<returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> RemoveUserFromGroupAsync(string userName, string
groupName)
{
    // Remove the user from the group.
    var removeUserRequest = new RemoveUserFromGroupRequest()
    {
        UserName = userName,
        GroupName = groupName,
    };

    var response = await
_IAMService.RemoveUserFromGroupAsync(removeUserRequest);
    return response.HttpStatusCode == HttpStatusCode.OK;
}
```

- For API details, see [RemoveUserFromGroup](#) in *AWS SDK for .NET API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Update an IAM server certificate using an AWS SDK

The following code examples show how to update an IAM server certificate.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
bool AwsDoc::IAM::updateServerCertificate(const Aws::String
    &currentCertificateName,
                                              const Aws::String &newCertificateName,
                                              const Aws::Client::ClientConfiguration
    &clientConfig) {
    Aws::IAM::IAMClient iam(clientConfig);
    Aws::IAM::Model::UpdateServerCertificateRequest request;
    request.SetServerCertificateName(currentCertificateName);
    request.SetNewServerCertificateName(newCertificateName);

    auto outcome = iam.UpdateServerCertificate(request);
    bool result = true;
    if (outcome.IsSuccess()) {
        std::cout << "Server certificate " << currentCertificateName
            << " successfully renamed as " << newCertificateName
            << std::endl;
    }
    else {
        if (outcome.GetError().GetErrorCode() !=
            Aws::IAM::IAMErrors::NO_SUCH_ENTITY) {
            std::cerr << "Error changing name of server certificate " <<
                currentCertificateName << " to " << newCertificateName << ":" <<
                outcome.GetError().GetMessage() << std::endl;
            result = false;
        }
        else {
            std::cout << "Certificate '" << currentCertificateName
                << "' not found." << std::endl;
        }
    }
    return result;
}
```

- For API details, see [UpdateServerCertificate](#) in *AWS SDK for C++ API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Update a server certificate.

```
import { UpdateServerCertificateCommand, IAMClient } from "@aws-sdk/client-iam";
```

```
const client = new IAMClient({});

/**
 *
 * @param {string} currentName
 * @param {string} newName
 */
export const updateServerCertificate = (currentName, newName) => {
    const command = new UpdateServerCertificateCommand({
        ServerCertificateName: currentName,
        NewServerCertificateName: newName,
    });

    return client.send(command);
};
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [UpdateServerCertificate](#) in [AWS SDK for JavaScript API Reference](#).

SDK for JavaScript (v2)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// Load the AWS SDK for Node.js
var AWS = require('aws-sdk');
// Set the region
AWS.config.update({region: 'REGION'});

// Create the IAM service object
var iam = new AWS.IAM({apiVersion: '2010-05-08'});

var params = {
    ServerCertificateName: 'CERTIFICATE_NAME',
    NewServerCertificateName: 'NEW_CERTIFICATE_NAME'
};

iam.updateServerCertificate(params, function(err, data) {
    if (err) {
        console.log("Error", err);
    } else {
        console.log("Success", data);
    }
});
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [UpdateServerCertificate](#) in [AWS SDK for JavaScript API Reference](#).

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Update an IAM user using an AWS SDK

The following code examples show how to update an IAM user.

Warning

To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as [AWS IAM Identity Center \(successor to AWS Single Sign-On\)](#).

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Create read-only and read-write users \(p. 973\)](#)

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
bool AwsDoc::IAM::updateUser(const Aws::String &currentUserName,
                             const Aws::String &newUserName,
                             const Aws::Client::ClientConfiguration &clientConfig)
{
    Aws::IAM::IAMClient iam(clientConfig);

    Aws::IAM::Model::UpdateUserRequest request;
    request.SetUserName(currentUserName);
    request.SetNewUserName(newUserName);

    auto outcome = iam.UpdateUser(request);
    if (outcome.IsSuccess()) {
        std::cout << "IAM user " << currentUserName <<
                    " successfully updated with new user name " << newUserName <<
                    std::endl;
    }
    else {
        std::cerr << "Error updating user name for IAM user " << currentUserName <<
                    ":" << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- For API details, see [UpdateUser in AWS SDK for C++ API Reference](#).

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public static void updateIAMUser(IamClient iam, String curName, String newName )
{
    try {
        UpdateUserRequest request = UpdateUserRequest.builder()
            .userName(curName)
            .newUserName(newName)
```

```
        .build();

        iam.updateUser(request);
        System.out.printf("Successfully updated user to username %s", newName);

    } catch (IamException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

- For API details, see [UpdateUser in AWS SDK for Java 2.x API Reference](#).

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Update the user.

```
import { UpdateUserCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} currentUserName
 * @param {string} newUserName
 */
export const updateUser = (currentUserName, newUserName) => {
    const command = new UpdateUserCommand({
        UserName: currentUserName,
        NewUserName: newUserName,
    });

    return client.send(command);
};
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [UpdateUser in AWS SDK for JavaScript API Reference](#).

SDK for JavaScript (v2)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// Load the AWS SDK for Node.js
var AWS = require('aws-sdk');
// Set the region
AWS.config.update({region: 'REGION'});

// Create the IAM service object
var iam = new AWS.IAM({apiVersion: '2010-05-08'});

var params = {
    UserName: process.argv[2],
```

```
    NewUserName: process.argv[3]
};

iam.updateUser(params, function(err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [UpdateUser](#) in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

Note

This is prerelease documentation for a feature in preview release. It is subject to change.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
suspend fun updateIAMUser(curName: String?, newName: String?) {

    val request = UpdateUserRequest {
        userName = curName
        newUserName = newName
    }

    IamClient { region = "AWS_GLOBAL" }.use { iamClient ->
        iamClient.updateUser(request)
        println("Successfully updated user to $newName")
    }
}
```

- For API details, see [UpdateUser](#) in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
def update_user(user_name, new_user_name):
    """
    Updates a user's name.

    :param user_name: The current name of the user to update.
    :param new_user_name: The new name to assign to the user.
    :return: The updated user.
    """
    try:
```

```
user = iam.User(user_name)
user.update(NewUserName=new_user_name)
logger.info("Renamed %s to %s.", user_name, new_user_name)
except ClientError:
    logger.exception("Couldn't update name for user %s.", user_name)
    raise
return user
```

- For API details, see [UpdateUser in AWS SDK for Python \(Boto3\) API Reference](#).

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Update an IAM access key using an AWS SDK

The following code examples show how to update an IAM access key.

Warning

To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as [AWS IAM Identity Center \(successor to AWS Single Sign-On\)](#).

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Manage access keys \(p. 980\)](#)

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
bool AwsDoc::IAM::updateAccessKey(const Aws::String &userName,
                                    const Aws::String &accessKeyID,
                                    Aws::IAM::Model::StatusType status,
                                    const Aws::Client::ClientConfiguration
&clientConfig) {
    Aws::IAM::IAMClient iam(clientConfig);
    Aws::IAM::Model::UpdateAccessKeyRequest request;
    request.SetUserName(userName);
    request.SetAccessKeyId(accessKeyID);
    request.SetStatus(status);

    auto outcome = iam.UpdateAccessKey(request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully updated status of access key "
              << accessKeyID << " for user " << userName << std::endl;
    }
    else {
        std::cerr << "Error updated status of access key " << accessKeyID <<
                    " for user " << userName << ":" <<
                    outcome.GetError().GetMessage() << std::endl;
    }
    return outcome.IsSuccess();
}
```

- For API details, see [UpdateAccessKey](#) in *AWS SDK for C++ API Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public static void updateKey(IamClient iam, String username, String accessId,
String status) {

    try {
        if (status.toLowerCase().equalsIgnoreCase("active")) {
            statusType = StatusType.ACTIVE;
        } else if (status.toLowerCase().equalsIgnoreCase("inactive")) {
            statusType = StatusType.INACTIVE;
        } else {
            statusType = StatusType.UNKNOWN_TO_SDK_VERSION;
        }

        UpdateAccessKeyRequest request = UpdateAccessKeyRequest.builder()
            .accessKeyId(accessId)
            .userName(username)
            .status(statusType)
            .build();

        iam.updateAccessKey(request);
        System.out.printf("Successfully updated the status of access key %s to"
+
                           "status %s for user %s", accessId, status, username);

    } catch (IamException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

- For API details, see [UpdateAccessKey](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Update the access key.

```
import {
  UpdateAccessKeyCommand,
  IAMClient,
  StatusType,
} from "@aws-sdk/client-iam";

const client = new IAMClient({});
```

```
/**  
 * @param {string} userName  
 * @param {string} accessKeyId  
 */  
export const updateAccessKey = (userName, accessKeyId) => {  
    const command = new UpdateAccessKeyCommand({  
        AccessKeyId: accessKeyId,  
        Status: StatusType.Inactive,  
        UserName: userName,  
    });  
  
    return client.send(command);  
};
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [UpdateAccessKey](#) in [AWS SDK for JavaScript API Reference](#).

SDK for JavaScript (v2)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// Load the AWS SDK for Node.js  
var AWS = require('aws-sdk');  
// Set the region  
AWS.config.update({region: 'REGION'});  
  
// Create the IAM service object  
var iam = new AWS.IAM({apiVersion: '2010-05-08'});  
  
var params = {  
    AccessKeyId: 'ACCESS_KEY_ID',  
    Status: 'Active',  
    UserName: 'USER_NAME'  
};  
  
iam.updateAccessKey(params, function(err, data) {  
    if (err) {  
        console.log("Error", err);  
    } else {  
        console.log("Success", data);  
    }  
});
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [UpdateAccessKey](#) in [AWS SDK for JavaScript API Reference](#).

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
def update_key(user_name, key_id, activate):
```

```
"""
Updates the status of a key.

:param user_name: The user that owns the key.
:param key_id: The ID of the key to update.
:param activate: When True, the key is activated. Otherwise, the key is
deactivated.
"""

try:
    key = iam.User(user_name).AccessKey(key_id)
    if activate:
        key.activate()
    else:
        key.deactivate()
    logger.info("%s key %s.", 'Activated' if activate else 'Deactivated',
key_id)
except ClientError:
    logger.exception(
        "Couldn't %s key %s.", 'Activate' if activate else 'Deactivate',
key_id)
    raise
```

- For API details, see [UpdateAccessKey](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Upload an IAM server certificate using an AWS SDK

The following code example shows how to upload an AWS Identity and Access Management (IAM) server certificate.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import { UploadServerCertificateCommand, IAMClient } from "@aws-sdk/client-iam";
import { readFileSync } from "fs";
import { dirnameFromMetaUrl } from "libs/utils/util-fs.js";
import * as path from "path";

const client = new IAMClient({});

/**
 * The certificate body and private key were generated with the
 * following command.
 *
 * ``
 * openssl req -x509 -newkey rsa:4096 -sha256 -days 3650 -nodes \
 * -keyout example.key -out example.crt -subj "/CN=example.com" \
 * -addext "subjectAltName=DNS:example.com,DNS:www.example.net,IP:10.0.0.1"
 * ``
 */

```

```
const certBody = readFileSync(
  path.join(
    dirnameFromMetaUrl(import.meta.url),
    "../../../../../resources/sample_files/sample_cert.pem"
  )
);

const privateKey = readFileSync(
  path.join(
    dirnameFromMetaUrl(import.meta.url),
    "../../../../../resources/sample_files/sample_private_key.pem"
  )
);

/**
 *
 * @param {string} certificateName
 */
export const uploadServerCertificate = (certificateName) => {
  const command = new UploadServerCertificateCommand({
    ServerCertificateName: certificateName,
    CertificateBody: certBody.toString(),
    PrivateKey: privateKey.toString(),
  });

  return client.send(command);
};
```

- For API details, see [UploadServerCertificate](#) in *AWS SDK for JavaScript API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Scenarios for IAM using AWS SDKs

The following code examples show you how to implement common scenarios in IAM with AWS SDKs. These scenarios show you how to accomplish specific tasks by calling multiple functions within IAM. Each scenario includes a link to GitHub, where you can find instructions on how to set up and run the code.

Examples

- [Create an IAM group and add a user to the group using an AWS SDK \(p. 874\)](#)
- [Create an IAM user and assume a role with AWS STS using an AWS SDK \(p. 891\)](#)
- [Create read-only and read-write IAM users using an AWS SDK \(p. 973\)](#)
- [Manage IAM access keys using an AWS SDK \(p. 980\)](#)
- [Manage IAM policies using an AWS SDK \(p. 982\)](#)
- [Manage IAM roles using an AWS SDK \(p. 986\)](#)
- [Manage your IAM account using an AWS SDK \(p. 989\)](#)
- [Roll back an IAM policy version using an AWS SDK \(p. 993\)](#)
- [Work with the IAM Policy Builder API using an AWS SDK \(p. 994\)](#)

Create an IAM group and add a user to the group using an AWS SDK

The following code example shows how to:

- Create a group and grant full Amazon S3 access permissions to it.
- Create a new user with no permissions to access Amazon S3.
- Add the user to the group and show that they now have permissions for Amazon S3, then clean up resources.

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
global using Amazon;
global using Amazon.IdentityManagement;
global using Amazon.S3;
global using Amazon.SecurityToken;
global using IAMActions;
global using IamScenariosCommon;
global using Microsoft.Extensions.DependencyInjection;
global using Microsoft.Extensions.Hosting;
global using Microsoft.Extensions.Logging;
global using Microsoft.Extensions.Logging.Console;
global using Microsoft.Extensions.Logging.Debug;

namespace IAMActions;

public class IAMWrapper
{
    private readonly IAmazonIdentityManagementService _IAMService;

    /// <summary>
    /// Constructor for the IAMWrapper class.
    /// </summary>
    /// <param name="IAMService">An IAM client object.</param>
    public IAMWrapper(IAmazonIdentityManagementService IAMService)
    {
        _IAMService = IAMService;
    }

    /// <summary>
    /// Add an existing IAM user to an existing IAM group.
    /// </summary>
    /// <param name="userName">The username of the user to add.</param>
    /// <param name="groupName">The name of the group to add the user to.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> AddUserToGroupAsync(string userName, string groupName)
    {
        var response = await _IAMService.AddUserToGroupAsync(new
AddUserToGroupRequest
        {
            GroupName = groupName,
            UserName = userName,
        });

        return response.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Attach an IAM policy to a role.
    /// </summary>
}
```

```

    ///> /summary>
    ///> <param name="policyArn">The policy to attach.</param>
    ///> <param name="roleName">The role that the policy will be attached to.</param>
    ///> <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> AttachRolePolicyAsync(string policyArn, string
    roleName)
    {
        var response = await _IAMService.AttachRolePolicyAsync(new
        AttachRolePolicyRequest
        {
            PolicyArn = policyArn,
            RoleName = roleName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    ///> <summary>
    ///> Create an IAM access key for a user.
    ///> </summary>
    ///> <param name="userName">The username for which to create the IAM access
    ///> key.</param>
    ///> <returns>The AccessKey.</returns>
    public async Task<AccessKey> CreateAccessKeyAsync(string userName)
    {
        var response = await _IAMService.CreateAccessKeyAsync(new
        CreateAccessKeyRequest
        {
            UserName = userName,
        });

        return response.AccessKey;
    }

    ///> <summary>
    ///> Create an IAM group.
    ///> </summary>
    ///> <param name="groupName">The name to give the IAM group.</param>
    ///> <returns>The IAM group that was created.</returns>
    public async Task<Group> CreateGroupAsync(string groupName)
    {
        var response = await _IAMService.CreateGroupAsync(new CreateGroupRequest
        { GroupName = groupName });
        return response.Group;
    }

    ///> <summary>
    ///> Create an IAM policy.
    ///> </summary>
    ///> <param name="policyName">The name to give the new IAM policy.</param>
    ///> <param name="policyDocument">The policy document for the new policy.</param>
    ///> <returns>The new IAM policy object.</returns>
    public async Task<ManagedPolicy> CreatePolicyAsync(string policyName, string
    policyDocument)
    {
        var response = await _IAMService.CreatePolicyAsync(new CreatePolicyRequest
        {
            PolicyDocument = policyDocument,
            PolicyName = policyName,
        });
    }
}

```

```

        return response.Policy;
    }

    ///<summary>
    ///<summary>
    /// Create a new IAM role.
    ///</summary>
    ///<param name="roleName">The name of the IAM role.</param>
    ///<param name="rolePolicyDocument">The name of the IAM policy document
    /// for the new role.</param>
    ///<returns>The Amazon Resource Name (ARN) of the role.</returns>
    public async Task<string> CreateRoleAsync(string roleName, string
rolePolicyDocument)
    {
        var request = new CreateRoleRequest
        {
            RoleName = roleName,
            AssumeRolePolicyDocument = rolePolicyDocument,
        };

        var response = await _IAMService.CreateRoleAsync(request);
        return response.Role.Arn;
    }

    ///<summary>
    ///<summary>
    /// Create an IAM service-linked role.
    ///</summary>
    ///<param name="serviceName">The name of the AWS Service.</param>
    ///<param name="description">A description of the IAM service-linked role.</
param>
    ///<returns>The IAM role that was created.</returns>
    public async Task<Role> CreateServiceLinkedRoleAsync(string serviceName, string
description)
    {
        var request = new CreateServiceLinkedRoleRequest
        {
            AWSServiceName = serviceName,
            Description = description
        };

        var response = await _IAMService.CreateServiceLinkedRoleAsync(request);
        return response.Role;
    }

    ///<summary>
    ///<summary>
    /// Create an IAM user.
    ///</summary>
    ///<param name="userName">The username for the new IAM user.</param>
    ///<returns>The IAM user that was created.</returns>
    public async Task<User> CreateUserAsync(string userName)
    {
        var response = await _IAMService.CreateUserAsync(new CreateUserRequest
{ UserName = userName });
        return response.User;
    }

    ///<summary>
    ///<summary>
    /// Delete an IAM user's access key.
    ///</summary>
    ///<param name="accessKeyId">The Id for the IAM access key.</param>
    ///<param name="userName">The username of the user that owns the IAM
    /// access key.</param>

```

```

    ///<returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteAccessKeyAsync(string accessKeyId, string
userName)
    {
        var response = await _IAMService.DeleteAccessKeyAsync(new
DeleteAccessKeyRequest
        {
            AccessKeyId = accessKeyId,
            UserName = userName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    ///<summary>
    ///<summary>Delete an IAM group.
    ///</summary>
    ///<param name="groupName">The name of the IAM group to delete.</param>
    ///<returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteGroupAsync(string groupName)
    {
        var response = await _IAMService.DeleteGroupAsync(new DeleteGroupRequest
{ GroupName = groupName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }

    ///<summary>
    ///<summary>Delete an IAM policy associated with an IAM group.
    ///</summary>
    ///<param name="groupName">The name of the IAM group associated with the
    ///policy.</param>
    ///<param name="policyName">The name of the policy to delete.</param>
    ///<returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteGroupPolicyAsync(string groupName, string
policyName)
    {
        var request = new DeleteGroupPolicyRequest()
        {
            GroupName = groupName,
            PolicyName = policyName,
        };

        var response = await _IAMService.DeleteGroupPolicyAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    ///<summary>
    ///<summary>Delete an IAM policy.
    ///</summary>
    ///<param name="policyArn">The Amazon Resource Name (ARN) of the policy to
    ///delete.</param>
    ///<returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeletePolicyAsync(string policyArn)
    {
        var response = await _IAMService.DeletePolicyAsync(new DeletePolicyRequest
{ PolicyArn = policyArn });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    ///<summary>
    ///<summary>Delete an IAM role.
    ///</summary>

```

```

    ///> The name of the IAM role to delete.</param>
    ///> A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteRoleAsync(string roleName)
    {
        var response = await _IAMService.DeleteRoleAsync(new DeleteRoleRequest
        { RoleName = roleName });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    ///> Delete an IAM role policy.
    ///>
    ///> The name of the IAM role.</param>
    ///> The name of the IAM role policy to delete.</param>
    ///> A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteRolePolicyAsync(string roleName, string
policyName)
    {
        var response = await _IAMService.DeleteRolePolicyAsync(new
DeleteRolePolicyRequest
        {
            PolicyName = policyName,
            RoleName = roleName,
        });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    ///> Delete an IAM user.
    ///>
    ///> The username of the IAM user to delete.</param>
    ///> A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteUserAsync(string userName)
    {
        var response = await _IAMService.DeleteUserAsync(new DeleteUserRequest
        { UserName = userName });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    ///> Delete an IAM user policy.
    ///>
    ///> The name of the IAM policy to delete.</param>
    ///> The username of the IAM user.</param>
    ///> A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteUserPolicyAsync(string policyName, string
userName)
    {
        var response = await _IAMService.DeleteUserPolicyAsync(new
DeleteUserPolicyRequest { PolicyName = policyName, UserName = userName });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    ///> Detach an IAM policy from an IAM role.
    ///>
    ///> The Amazon Resource Name (ARN) of the IAM policy.</param>
    ///> The name of the IAM role.</param>

```

```

    ///<returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DetachRolePolicyAsync(string policyArn, string
roleName)
    {
        var response = await _IAMService.DetachRolePolicyAsync(new
DetachRolePolicyRequest
        {
            PolicyArn = policyArn,
            RoleName = roleName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    ///<summary>
    /// Gets the IAM password policy for an AWS account.
    /// </summary>
    ///<returns>The PasswordPolicy for the AWS account.</returns>
    public async Task<PasswordPolicy> GetAccountPasswordPolicyAsync()
    {
        var response = await _IAMService.GetAccountPasswordPolicyAsync(new
GetAccountPasswordPolicyRequest());
        return response.PasswordPolicy;
    }

    ///<summary>
    /// Get information about an IAM policy.
    /// </summary>
    ///<param name="policyArn">The IAM policy to retrieve information for.</param>
    ///<returns>The IAM policy.</returns>
    public async Task<ManagedPolicy> GetPolicyAsync(string policyArn)
    {

        var response = await _IAMService.GetPolicyAsync(new GetPolicyRequest
{ PolicyArn = policyArn });
        return response.Policy;
    }

    ///<summary>
    /// Get information about an IAM role.
    /// </summary>
    ///<param name="roleName">The name of the IAM role to retrieve information
    /// for.</param>
    ///<returns>The IAM role that was retrieved.</returns>
    public async Task<Role> GetRoleAsync(string roleName)
    {
        var response = await _IAMService.GetRoleAsync(new GetRoleRequest
        {
            RoleName = roleName,
        });

        return response.Role;
    }

    ///<summary>
    /// Get information about an IAM user.
    /// </summary>
    ///<param name="userName">The username of the user.</param>
    ///<returns>An IAM user object.</returns>
    public async Task<User> GetUserAsync(string userName)
    {

```

```

        var response = await _IAMService.GetUserAsync(new GetUserRequest { UserName
= userName });
            return response.User;
    }

    /// <summary>
    /// List the IAM role policies that are attached to an IAM role.
    /// </summary>
    /// <param name="roleName">The IAM role to list IAM policies for.</param>
    /// <returns>A list of the IAM policies attached to the IAM role.</returns>
    public async Task<List<AttachedPolicyType>> ListAttachedRolePoliciesAsync(string roleName)
    {
        var attachedPolicies = new List<AttachedPolicyType>();
        var attachedRolePoliciesPaginator =
_IAMService.Paginator.ListAttachedRolePolicies(new
ListAttachedRolePoliciesRequest { RoleName = roleName });

        await foreach (var response in attachedRolePoliciesPaginator.Responses)
        {
            attachedPolicies.AddRange(response.AttachedPolicies);
        }

        return attachedPolicies;
    }

    /// <summary>
    /// List IAM groups.
    /// </summary>
    /// <returns>A list of IAM groups.</returns>
    public async Task<List<Group>> ListGroupsAsync()
    {
        var groupsPaginator = _IAMService.Paginator.ListGroups(new
ListGroupsRequest());
        var groups = new List<Group>();

        await foreach (var response in groupsPaginator.Responses)
        {
            groups.AddRange(response.Groups);
        }

        return groups;
    }

    /// <summary>
    /// List IAM policies.
    /// </summary>
    /// <returns>A list of the IAM policies.</returns>
    public async Task<List<ManagedPolicy>> ListPoliciesAsync()
    {
        var listPoliciesPaginator = _IAMService.Paginator.ListPolicies(new
ListPoliciesRequest());
        var policies = new List<ManagedPolicy>();

        await foreach (var response in listPoliciesPaginator.Responses)
        {
            policies.AddRange(response.Policies);
        }

        return policies;
    }
}

```

```

    ///<summary>
    /// List IAM role policies.
    /// </summary>
    /// <param name="roleName">The IAM role for which to list IAM policies.</param>
    /// <returns>A list of IAM policy names.</returns>
    public async Task<List<string>> ListRolePoliciesAsync(string roleName)
    {
        var listRolePoliciesPaginator = _IAMService.Paginator.ListRolePolicies(new
ListRolePoliciesRequest { RoleName = roleName });
        var policyNames = new List<string>();

        await foreach (var response in listRolePoliciesPaginator.Responses)
        {
            policyNames.AddRange(response.PolicyNames);
        }

        return policyNames;
    }

    ///<summary>
    /// List IAM roles.
    /// </summary>
    /// <returns>A list of IAM roles.</returns>
    public async Task<List<Role>> ListRolesAsync()
    {
        var listRolesPaginator = _IAMService.Paginator.ListRoles(new
ListRolesRequest());
        var roles = new List<Role>();

        await foreach (var response in listRolesPaginator.Responses)
        {
            roles.AddRange(response.Roles);
        }

        return roles;
    }

    ///<summary>
    /// List SAML authentication providers.
    /// </summary>
    /// <returns>A list of SAML providers.</returns>
    public async Task<List<SAMLProviderListEntry>> ListSAMLProvidersAsync()
    {
        var response = await _IAMService.ListSAMLProvidersAsync(new
ListSAMLProvidersRequest());
        return response.SAMLProviderList;
    }

    ///<summary>
    /// List IAM users.
    /// </summary>
    /// <returns>A list of IAM users.</returns>
    public async Task<List<User>> ListUsersAsync()
    {
        var listUsersPaginator = _IAMService.Paginator.ListUsers(new
ListUsersRequest());
        var users = new List<User>();

        await foreach (var response in listUsersPaginator.Responses)
        {
            users.AddRange(response.Users);
        }
    }

```

```

        return users;
    }

    ///<summary>
    /// Remove a user from an IAM group.
    ///</summary>
    ///<param name="userName">The username of the user to remove.</param>
    ///<param name="groupName">The name of the IAM group to remove the user
from.</param>
    ///<returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> RemoveUserFromGroupAsync(string userName, string
groupName)
{
    // Remove the user from the group.
    var removeUserRequest = new RemoveUserFromGroupRequest()
    {
        UserName = userName,
        GroupName = groupName,
    };

    var response = await
_IAMService.RemoveUserFromGroupAsync(removeUserRequest);
    return response.HttpStatusCode == HttpStatusCode.OK;
}

    ///<summary>
    /// Add or update an inline policy document that is embedded in an IAM group.
    ///</summary>
    ///<param name="groupName">The name of the IAM group.</param>
    ///<param name="policyName">The name of the IAM policy.</param>
    ///<param name="policyDocument">The policy document defining the IAM policy.</
param>
    ///<returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> PutGroupPolicyAsync(string groupName, string
policyName, string policyDocument)
{
    var request = new PutGroupPolicyRequest
    {
        GroupName = groupName,
        PolicyName = policyName,
        PolicyDocument = policyDocument
    };

    var response = await _IAMService.PutGroupPolicyAsync(request);
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

    ///<summary>
    /// Update the inline policy document embedded in a role.
    ///</summary>
    ///<param name="policyName">The name of the policy to embed.</param>
    ///<param name="roleName">The name of the role to update.</param>
    ///<param name="policyDocument">The policy document that defines the role.</
param>
    ///<returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> PutRolePolicyAsync(string policyName, string roleName,
string policyDocument)
{
    var request = new PutRolePolicyRequest
    {
        PolicyName = policyName,
        RoleName = roleName,
        PolicyDocument = policyDocument
}

```

```

    };

    var response = await _IAMService.PutRolePolicyAsync(request);
    return response.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// Add or update an inline policy document that is embedded in an IAM user.
/// </summary>
/// <param name="userName">The name of the IAM user.</param>
/// <param name="policyName">The name of the IAM policy.</param>
/// <param name="policyDocument">The policy document defining the IAM policy.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> PutUserPolicyAsync(string userName, string policyName,
string policyDocument)
{
    var request = new PutUserPolicyRequest
    {
        UserName = userName,
        PolicyName = policyName,
        PolicyDocument = policyDocument
    };

    var response = await _IAMService.PutUserPolicyAsync(request);
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

/// <summary>
/// Wait for a new access key to be ready to use.
/// </summary>
/// <param name="accessKeyId">The Id of the access key.</param>
/// <returns>A boolean value indicating the success of the action.</returns>
public async Task<bool> WaitUntilAccessKeyIsReady(string accessKeyId)
{
    var keyReady = false;

    do
    {
        try
        {
            var response = await _IAMService.GetAccessKeyLastUsedAsync(
                new GetAccessKeyLastUsedRequest { AccessKeyId = accessKeyId });
            if (response.UserName is not null)
            {
                keyReady = true;
            }
        }
        catch (NoSuchEntityException)
        {
            keyReady = false;
        }
    } while (!keyReady);

    return keyReady;
}
}

using Amazon.Runtime;
using Microsoft.Extensions.Configuration;
namespace IAMGroups;
public class IAMGroups

```

```
{
    private static ILogger logger = null!;

    // Represents JSON code for AWS full access policy for Amazon Simple
    // Storage Service (Amazon S3).
    private const string S3FullAccessPolicyDocument = "{" +
        " \"Statement\" : [{" +
            " \"Action\" : ["s3:*"], " +
            " \"Effect\" : \"Allow\", " +
            " \"Resource\" : \"*\" " +
        "}]" +
    "}";

    static async Task Main(string[] args)
    {
        // Set up dependency injection for the AWS service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
                        LogLevel.Information)
                .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices(_<services> =>
                services.AddAWSService<IAmazonIdentityManagementService>()
                    .AddTransient<IAMWrapper>()
                    .AddTransient<UIWrapper>()
                )
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<IAMGroups>();

        IConfiguration configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load test settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally load local settings.
            .Build();

        var groupUserName = configuration["GroupUserName"];
        var groupName = configuration["GroupName"];
        var groupPolicyName = configuration["GroupPolicyName"];
        var groupBucketName = configuration["GroupBucketName"];

        var wrapper = host.Services.GetRequiredService<IAMWrapper>();
        var uiWrapper = host.Services.GetRequiredService<UIWrapper>();

        uiWrapper.DisplayGroupsOverview();
        uiWrapper.PressEnter();

        // Create an IAM group.
        uiWrapper.DisplayTitle("Create IAM group");
        Console.WriteLine("Let's begin by creating a new IAM group.");
        var group = await wrapper.CreateGroupAsync(groupName);

        // Add an inline IAM policy to the group.
        uiWrapper.DisplayTitle("Add policy to group");
        Console.WriteLine("Add an inline policy to the group that allows members to
        have full access to");
        Console.WriteLine("Amazon Simple Storage Service (Amazon S3) buckets.");

        await wrapper.PutGroupPolicyAsync(group.GroupName, groupPolicyName,
            S3FullAccessPolicyDocument);

        uiWrapper.PressEnter();
    }
}
```

```

// Now create a new user.
uiWrapper.DisplayTitle("Create an IAM user");
Console.WriteLine("Now let's create a new IAM user.");
var groupUser = await wrapper.CreateUserAsync(groupUserName);

// Add the new user to the group.
uiWrapper.DisplayTitle("Add the user to the group");
Console.WriteLine("Adding the user to the group, which will give the user
the same permissions as the group.");
await wrapper.AddUserToGroupAsync(groupUser.UserName, group.GroupName);

Console.WriteLine($"User, {groupUser.UserName}, has been added to the
group, {group.GroupName}.");
uiWrapper.PressEnter();

Console.WriteLine("Now that we have created a user, and added the user to
the group, let's create an IAM access key.");

// Create access and secret keys for the user.
var accessKey = await wrapper.CreateAccessKeyAsync(groupUserName);
Console.WriteLine("Key created.");
uiWrapper.WaitABit(15, "Waiting for the access key to be ready for use.");

uiWrapper.DisplayTitle("List buckets");
Console.WriteLine("To prove that the user has access to Amazon S3, list the
S3 buckets for the account");

var s3Client = new AmazonS3Client(accessKey.AccessKeyId,
accessKey.SecretAccessKey);
var stsClient = new AmazonSecurityTokenServiceClient(accessKey.AccessKeyId,
accessKey.SecretAccessKey);

var s3Wrapper = new S3Wrapper(s3Client, stsClient);

var buckets = await s3Wrapper.ListMyBucketsAsync();

if (buckets is not null)
{
    buckets.ForEach(bucket =>
    {
        Console.WriteLine($"{bucket.BucketName}\tcreated on:
{bucket.CreationDate}");
    });
}

// Show that the user also has write access to Amazon S3 by creating
// a new bucket.
uiWrapper.DisplayTitle("Create a bucket");
Console.WriteLine("Since group members have full access to Amazon S3, let's
create a bucket.");
var success = await s3Wrapper.PutBucketAsync(groupBucketName);

if (success)
{
    Console.WriteLine($"Successfully created the bucket:
{groupBucketName}.");
}

uiWrapper.PressEnter();

Console.WriteLine("Let's list the user's S3 buckets again to show the new
bucket.");
buckets = await s3Wrapper.ListMyBucketsAsync();

```

```

        if (buckets is not null)
        {
            buckets.ForEach(bucket =>
            {
                Console.WriteLine($"{bucket.BucketName}\tcreated on:
{bucket.CreationDate}");
            });
        }

        uiWrapper.PressEnter();

        uiWrapper.DisplayTitle("Clean up resources");
        Console.WriteLine("First delete the bucket we created.");
        await s3Wrapper.DeleteBucketAsync(groupBucketName);

        Console.WriteLine($"Now remove the user, {groupUserName}, from the group,
{groupName}.");
        await wrapper.RemoveUserFromGroupAsync(groupUserName, groupName);

        Console.WriteLine("Delete the user's access key.");
        await wrapper.DeleteAccessKeyAsync(accessKey.AccessKeyId, groupUserName);

        // Now we can safely delete the user.
        Console.WriteLine("Now we can delete the user.");
        await wrapper.DeleteUserAsync(groupUserName);

        uiWrapper.PressEnter();

        Console.WriteLine("Now we will delete the IAM policy attached to the
group.");
        await wrapper.DeleteGroupPolicyAsync(groupName, groupPolicyName);

        Console.WriteLine("Now we delete the IAM group.");
        await wrapper.DeleteGroupAsync(groupName);

        uiWrapper.PressEnter();

        Console.WriteLine("The IAM groups demo has completed.");
        uiWrapper.PressEnter();
    }
}

namespace IamScenariosCommon;

using System.Net;

/// <summary>
/// A class to perform Amazon Simple Storage Service (Amazon S3) actions for
/// the IAM Basics scenario.
/// </summary>
public class S3Wrapper
{
    private IAmazonS3 _s3Service;
    private IAmazonSecurityTokenService _stsService;

    /// <summary>
    /// Constructor for the S3Wrapper class.
    /// </summary>
    /// <param name="s3Service">An Amazon S3 client object.</param>
    /// <param name="stsService">An AWS Security Token Service (AWS STS)
    /// client object.</param>
    public S3Wrapper(IAmazonS3 s3Service, IAmazonSecurityTokenService stsService)
    {
        _s3Service = s3Service;
    }
}

```

```

        _stsService = stsService;
    }

    // snippet.start:[STS.dotnetv3.AssumeS3Role]
    /// <summary>
    /// Assumes an AWS Identity and Access Management (IAM) role that allows
    /// Amazon S3 access for the current session.
    /// </summary>
    /// <param name="roleSession">A string representing the current session.</param>
    public async Task<Credentials> AssumeS3RoleAsync(string roleSession, string
    roleToAssume)
    {
        // Create the request to use with the AssumeRoleAsync call.
        var request = new AssumeRoleRequest()
        {
            RoleSessionName = roleSession,
            RoleArn = roleToAssume,
        };

        var response = await _stsService.AssumeRoleAsync(request);

        return response.Credentials;
    }

    // snippet.end:[STS.dotnetv3.AssumeS3Role]

    /// <summary>
    /// Delete an S3 bucket.
    /// </summary>
    /// <param name="bucketName">Name of the S3 bucket to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteBucketAsync(string bucketName)
    {
        var result = await _s3Service.DeleteBucketAsync(new DeleteBucketRequest
        { BucketName = bucketName });
        return result.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// List the buckets that are owned by the user's account.
    /// </summary>
    /// <returns>Async Task.</returns>
    public async Task<List<S3Bucket>> ListMyBucketsAsync()
    {
        try
        {
            // Get the list of buckets accessible by the new user.
            var response = await _s3Service.ListBucketsAsync();

            return response.Buckets;
        }
        catch (AmazonS3Exception ex)
        {
            // Something else went wrong. Display the error message.
            Console.WriteLine($"Error: {ex.Message}");
            return null;
        }
    }

    /// <summary>
    /// Create a new S3 bucket.
    /// </summary>
    /// <param name="bucketName">The name for the new bucket.</param>

```

```

    ///>A Boolean value indicating whether the action completed
    ///>successfully.</returns>
    public async Task<bool> PutBucketAsync(string bucketName)
    {
        var response = await _s3Service.PutBucketAsync(new PutBucketRequest
{ BucketName = bucketName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }

    ///>
    ///> Update the client objects with new client objects. This is available
    ///> because the scenario uses the methods of this class without and then
    ///> with the proper permissions to list S3 buckets.
    ///>
    ///> <param name="s3Service">The Amazon S3 client object.</param>
    ///> <param name="stsService">The AWS STS client object.</param>
    public void UpdateClients(IAmazonS3 s3Service, IAmazonSecurityTokenService
stsService)
{
    _s3Service = s3Service;
    _stsService = stsService;
}
}

namespace IamScenariosCommon;

public class UIWrapper
{
    public readonly string SepBar = new(' ', Console.WindowWidth);

    ///>
    ///> Show information about the IAM Groups scenario.
    ///>
    public void DisplayGroupsOverview()
    {
        Console.Clear();

        DisplayTitle("Welcome to the IAM Groups Demo");
        Console.WriteLine("This example application does the following:");
        Console.WriteLine("\t1. Creates an Amazon Identity and Access Management
(IAM) group.");
        Console.WriteLine("\t2. Adds an IAM policy to the IAM group giving it full
access to Amazon S3.");
        Console.WriteLine("\t3. Creates a new IAM user.");
        Console.WriteLine("\t4. Creates an IAM access key for the user.");
        Console.WriteLine("\t5. Adds the user to the IAM group.");
        Console.WriteLine("\t6. Lists the buckets on the account.");
        Console.WriteLine("\t7. Proves that the user has full Amazon S3 access by
creating a bucket.");
        Console.WriteLine("\t8. List the buckets again to show the new bucket.");
        Console.WriteLine("\t9. Cleans up all the resources created.");
    }

    ///>
    ///> Show information about the IAM Basics scenario.
    ///>
    public void DisplayBasicsOverview()
    {
        Console.Clear();

        DisplayTitle("Welcome to IAM Basics");
        Console.WriteLine("This example application does the following:");
        Console.WriteLine("\t1. Creates a user with no permissions.");
        Console.WriteLine("\t2. Creates a role and policy that grant
s3>ListAllMyBuckets permission.");
    }
}

```

```

        Console.WriteLine("\t3. Grants the user permission to assume the role.");
        Console.WriteLine("\t4. Creates an S3 client object as the user and tries
to list buckets (this will fail).");
        Console.WriteLine("\t5. Gets temporary credentials by assuming the role.");
        Console.WriteLine("\t6. Creates a new S3 client object with the temporary
credentials and lists the buckets (this will succeed).");
        Console.WriteLine("\t7. Deletes all the resources.");
    }

    ///<summary>
    /// Display a message and wait until the user presses enter.
    ///</summary>
    public void PressEnter()
    {
        Console.Write("\nPress <Enter> to continue. ");
        _ = Console.ReadLine();
        Console.WriteLine();
    }

    ///<summary>
    /// Pad a string with spaces to center it on the console display.
    ///</summary>
    ///<param name="strToCenter">The string to be centered.</param>
    ///<returns>The padded string.</returns>
    public string CenterString(string strToCenter)
    {
        var padAmount = (Console.WindowWidth - strToCenter.Length) / 2;
        var leftPad = new string(' ', padAmount);
        return $"{leftPad}{strToCenter}";
    }

    ///<summary>
    /// Display a line of hyphens, the centered text of the title, and another
    /// line of hyphens.
    ///</summary>
    ///<param name="strTitle">The string to be displayed.</param>
    public void DisplayTitle(string strTitle)
    {
        Console.WriteLine(SepBar);
        Console.WriteLine(CenterString(strTitle));
        Console.WriteLine(SepBar);
    }

    ///<summary>
    /// Display a countdown and wait for a number of seconds.
    ///</summary>
    ///<param name="numSeconds">The number of seconds to wait.</param>
    public void WaitABit(int numSeconds, string msg)
    {
        Console.WriteLine(msg);

        // Wait for the requested number of seconds.
        for (int i = numSeconds; i > 0; i--)
        {
            System.Threading.Thread.Sleep(1000);
            Console.Write($"{i}...");
        }

        PressEnter();
    }
}

```

- For API details, see the following topics in *AWS SDK for .NET API Reference*.

- [AddUserToGroup](#)
- [AttachRolePolicy](#)
- [CreateAccessKey](#)
- [CreateGroup](#)
- [CreatePolicy](#)
- [CreateRole](#)
- [CreateUser](#)
- [DeleteAccessKey](#)
- [DeleteGroup](#)
- [DeleteGroupPolicy](#)
- [DeleteUser](#)
- [PutGroupPolicy](#)
- [RemoveUserFromGroup](#)

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Create an IAM user and assume a role with AWS STS using an AWS SDK

The following code examples show how to create a user and assume a role.

Warning

To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as [AWS IAM Identity Center \(successor to AWS Single Sign-On\)](#).

- Create a user with no permissions.
- Create a role that grants permission to list Amazon S3 buckets for the account.
- Add a policy to let the user assume the role.
- Assume the role and list S3 buckets using temporary credentials, then clean up resources.

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
global using Amazon;
global using Amazon.IdentityManagement;
global using Amazon.S3;
global using Amazon.S3.Model;
global using Amazon.SecurityToken;
global using Amazon.SecurityToken.Model;
global using IAMActions;
global using IamScenariosCommon;
global using Microsoft.Extensions.DependencyInjection;
global using Microsoft.Extensions.Hosting;
global using Microsoft.Extensions.Logging;
global using Microsoft.Extensions.Logging.Console;
```

```

global using Microsoft.Extensions.Logging.Debug;

namespace IAMActions;

public class IAMWrapper
{
    private readonly IAmazonIdentityManagementService _IAMService;

    /// <summary>
    /// Constructor for the IAMWrapper class.
    /// </summary>
    /// <param name="IAMService">An IAM client object.</param>
    public IAMWrapper(IAmazonIdentityManagementService IAMService)
    {
        _IAMService = IAMService;
    }

    /// <summary>
    /// Add an existing IAM user to an existing IAM group.
    /// </summary>
    /// <param name="userName">The username of the user to add.</param>
    /// <param name="groupName">The name of the group to add the user to.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> AddUserToGroupAsync(string userName, string groupName)
    {
        var response = await _IAMService.AddUserToGroupAsync(new
AddUserToGroupRequest
{
    GroupName = groupName,
    UserName = userName,
});

        return response.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Attach an IAM policy to a role.
    /// </summary>
    /// <param name="policyArn">The policy to attach.</param>
    /// <param name="roleName">The role that the policy will be attached to.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> AttachRolePolicyAsync(string policyArn, string
roleName)
    {
        var response = await _IAMService.AttachRolePolicyAsync(new
AttachRolePolicyRequest
{
    PolicyArn = policyArn,
    RoleName = roleName,
});

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Create an IAM access key for a user.
    /// </summary>
    /// <param name="userName">The username for which to create the IAM access
key.</param>
    /// <returns>The AccessKey.</returns>
    public async Task<AccessKey> CreateAccessKeyAsync(string userName)
    {

```

```

        var response = await _IAMService.CreateAccessKeyAsync(new
CreateAccessKeyRequest
{
    UserName = userName,
});

return response.AccessKey;
}

/// <summary>
/// Create an IAM group.
/// </summary>
/// <param name="groupName">The name to give the IAM group.</param>
/// <returns>The IAM group that was created.</returns>
public async Task<Group> CreateGroupAsync(string groupName)
{
    var response = await _IAMService.CreateGroupAsync(new CreateGroupRequest
{ GroupName = groupName });
    return response.Group;
}

/// <summary>
/// Create an IAM policy.
/// </summary>
/// <param name="policyName">The name to give the new IAM policy.</param>
/// <param name="policyDocument">The policy document for the new policy.</
param>
/// <returns>The new IAM policy object.</returns>
public async Task<ManagedPolicy> CreatePolicyAsync(string policyName, string
policyDocument)
{
    var response = await _IAMService.CreatePolicyAsync(new CreatePolicyRequest
{
    PolicyDocument = policyDocument,
    PolicyName = policyName,
});
    return response.Policy;
}

/// <summary>
/// Create a new IAM role.
/// </summary>
/// <param name="roleName">The name of the IAM role.</param>
/// <param name="rolePolicyDocument">The name of the IAM policy document
/// for the new role.</param>
/// <returns>The Amazon Resource Name (ARN) of the role.</returns>
public async Task<string> CreateRoleAsync(string roleName, string
rolePolicyDocument)
{
    var request = new CreateRoleRequest
{
    RoleName = roleName,
    AssumeRolePolicyDocument = rolePolicyDocument,
};

var response = await _IAMService.CreateRoleAsync(request);
return response.Role.Arn;
}

/// <summary>

```

```

    ///> Create an IAM service-linked role.
    ///</summary>
    ///<param name="serviceName">The name of the AWS Service.</param>
    ///<param name="description">A description of the IAM service-linked role.</param>
    ///<returns>The IAM role that was created.</returns>
    public async Task<Role> CreateServiceLinkedRoleAsync(string serviceName, string
description)
{
    var request = new CreateServiceLinkedRoleRequest
    {
        AWSServiceName = serviceName,
        Description = description
    };

    var response = await _IAMService.CreateServiceLinkedRoleAsync(request);
    return response.Role;
}

    ///<summary>
    ///<param name="userName">The username for the new IAM user.</param>
    ///<returns>The IAM user that was created.</returns>
    public async Task<User> CreateUserAsync(string userName)
{
    var response = await _IAMService.CreateUserAsync(new CreateUserRequest
{ UserName = userName });
    return response.User;
}

    ///<summary>
    ///<param name="accessKeyId">The Id for the IAM access key.</param>
    ///<param name="userName">The username of the user that owns the IAM
    ///access key.</param>
    ///<returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteAccessKeyAsync(string accessKeyId, string
userName)
{
    var response = await _IAMService.DeleteAccessKeyAsync(new
DeleteAccessKeyRequest
{
    AccessKeyId = accessKeyId,
    UserName = userName,
});
    return response.StatusCode == System.Net.HttpStatusCode.OK;
}

    ///<summary>
    ///<param name="groupName">The name of the IAM group to delete.</param>
    ///<returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteGroupAsync(string groupName)
{
    var response = await _IAMService.DeleteGroupAsync(new DeleteGroupRequest
{ GroupName = groupName });
    return response.StatusCode == HttpStatusCode.OK;
}

```

```

    ///<summary>
    /// Delete an IAM policy associated with an IAM group.
    ///</summary>
    ///<param name="groupName">The name of the IAM group associated with the
    /// policy.</param>
    ///<param name="policyName">The name of the policy to delete.</param>
    ///<returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteGroupPolicyAsync(string groupName, string
policyName)
    {
        var request = new DeleteGroupPolicyRequest()
        {
            GroupName = groupName,
            PolicyName = policyName,
        };

        var response = await _IAMService.DeleteGroupPolicyAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    ///<summary>
    /// Delete an IAM policy.
    ///</summary>
    ///<param name="policyArn">The Amazon Resource Name (ARN) of the policy to
    /// delete.</param>
    ///<returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeletePolicyAsync(string policyArn)
    {
        var response = await _IAMService.DeletePolicyAsync(new DeletePolicyRequest
{ PolicyArn = policyArn });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    ///<summary>
    /// Delete an IAM role.
    ///</summary>
    ///<param name="roleName">The name of the IAM role to delete.</param>
    ///<returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteRoleAsync(string roleName)
    {
        var response = await _IAMService.DeleteRoleAsync(new DeleteRoleRequest
{ RoleName = roleName });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    ///<summary>
    /// Delete an IAM role policy.
    ///</summary>
    ///<param name="roleName">The name of the IAM role.</param>
    ///<param name="policyName">The name of the IAM role policy to delete.</param>
    ///<returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteRolePolicyAsync(string roleName, string
policyName)
    {
        var response = await _IAMService.DeleteRolePolicyAsync(new
DeleteRolePolicyRequest
        {
            PolicyName = policyName,
            RoleName = roleName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

```

```

}

    /// <summary>
    /// Delete an IAM user.
    /// </summary>
    /// <param name="userName">The username of the IAM user to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteUserAsync(string userName)
    {
        var response = await _IAMService.DeleteUserAsync(new DeleteUserRequest
        { UserName = userName });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Delete an IAM user policy.
    /// </summary>
    /// <param name="policyName">The name of the IAM policy to delete.</param>
    /// <param name="userName">The username of the IAM user.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteUserPolicyAsync(string policyName, string
    userName)
    {
        var response = await _IAMService.DeleteUserPolicyAsync(new
        DeleteUserPolicyRequest { PolicyName = policyName, UserName = userName });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Detach an IAM policy from an IAM role.
    /// </summary>
    /// <param name="policyArn">The Amazon Resource Name (ARN) of the IAM policy.</
    param>
    /// <param name="roleName">The name of the IAM role.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DetachRolePolicyAsync(string policyArn, string
    roleName)
    {
        var response = await _IAMService.DetachRolePolicyAsync(new
        DetachRolePolicyRequest
        {
            PolicyArn = policyArn,
            RoleName = roleName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Gets the IAM password policy for an AWS account.
    /// </summary>
    /// <returns>The PasswordPolicy for the AWS account.</returns>
    public async Task<PasswordPolicy> GetAccountPasswordPolicyAsync()
    {
        var response = await _IAMService.GetAccountPasswordPolicyAsync(new
        GetAccountPasswordPolicyRequest());
        return response.PasswordPolicy;
    }
}

```

```

    ///<summary>
    /// Get information about an IAM policy.
    /// </summary>
    /// <param name="policyArn">The IAM policy to retrieve information for.</param>
    /// <returns>The IAM policy.</returns>
    public async Task<ManagedPolicy> GetPolicyAsync(string policyArn)
    {

        var response = await _IAMService.GetPolicyAsync(new GetPolicyRequest
        { PolicyArn = policyArn });
        return response.Policy;
    }

    ///<summary>
    /// Get information about an IAM role.
    /// </summary>
    /// <param name="roleName">The name of the IAM role to retrieve information
    /// for.</param>
    /// <returns>The IAM role that was retrieved.</returns>
    public async Task<Role> GetRoleAsync(string roleName)
    {
        var response = await _IAMService.GetRoleAsync(new GetRoleRequest
        {
            RoleName = roleName,
        });

        return response.Role;
    }

    ///<summary>
    /// Get information about an IAM user.
    /// </summary>
    /// <param name="userName">The username of the user.</param>
    /// <returns>An IAM user object.</returns>
    public async Task<User> GetUserAsync(string userName)
    {
        var response = await _IAMService.GetUserAsync(new GetUserRequest { UserName
= userName });
        return response.User;
    }

    ///<summary>
    /// List the IAM role policies that are attached to an IAM role.
    /// </summary>
    /// <param name="roleName">The IAM role to list IAM policies for.</param>
    /// <returns>A list of the IAM policies attached to the IAM role.</returns>
    public async Task<List<AttachedPolicyType>>
ListAttachedRolePoliciesAsync(string roleName)
    {
        var attachedPolicies = new List<AttachedPolicyType>();
        var attachedRolePoliciesPaginator =
_IAMService.Paginator.ListAttachedRolePolicies(new
ListAttachedRolePoliciesRequest { RoleName = roleName });

        await foreach (var response in attachedRolePoliciesPaginator.Responses)
        {
            attachedPolicies.AddRange(response.AttachedPolicies);
        }

        return attachedPolicies;
    }

```

```

    ///> <summary>
    ///> List IAM groups.
    ///> </summary>
    ///> <returns>A list of IAM groups.</returns>
    public async Task<List<Group>> ListGroupsAsync()
    {
        var groupsPaginator = _IAMService.Paginator.ListGroups(new
ListGroupsRequest());
        var groups = new List<Group>();

        await foreach (var response in groupsPaginator.Responses)
        {
            groups.AddRange(response.Groups);
        }

        return groups;
    }

    ///> <summary>
    ///> List IAM policies.
    ///> </summary>
    ///> <returns>A list of the IAM policies.</returns>
    public async Task<List<ManagedPolicy>> ListPoliciesAsync()
    {
        var listPoliciesPaginator = _IAMService.Paginator.ListPolicies(new
ListPoliciesRequest());
        var policies = new List<ManagedPolicy>();

        await foreach (var response in listPoliciesPaginator.Responses)
        {
            policies.AddRange(response.Policies);
        }

        return policies;
    }

    ///> <summary>
    ///> List IAM role policies.
    ///> </summary>
    ///> <param name="roleName">The IAM role for which to list IAM policies.</param>
    ///> <returns>A list of IAM policy names.</returns>
    public async Task<List<string>> ListRolePoliciesAsync(string roleName)
    {
        var listRolePoliciesPaginator = _IAMService.Paginator.ListRolePolicies(new
ListRolePoliciesRequest { RoleName = roleName });
        var policyNames = new List<string>();

        await foreach (var response in listRolePoliciesPaginator.Responses)
        {
            policyNames.AddRange(response.PolicyNames);
        }

        return policyNames;
    }

    ///> <summary>
    ///> List IAM roles.
    ///> </summary>
    ///> <returns>A list of IAM roles.</returns>
    public async Task<List<Role>> ListRolesAsync()
    {
        var listRolesPaginator = _IAMService.Paginator.ListRoles(new
ListRolesRequest());

```

```

        var roles = new List<Role>();

        await foreach (var response in listRolesPaginator.Responses)
        {
            roles.AddRange(response.Roles);
        }

        return roles;
    }

    ///<summary>
    ///<summary>List SAML authentication providers.
    ///</summary>
    ///<returns>A list of SAML providers.</returns>
    public async Task<List<SAMLProviderListEntry>> ListSAMLProvidersAsync()
    {
        var response = await _IAMService.ListSAMLProvidersAsync(new
ListSAMLProvidersRequest());
        return response.SAMLProviderList;
    }

    ///<summary>
    ///<summary>List IAM users.
    ///</summary>
    ///<returns>A list of IAM users.</returns>
    public async Task<List<User>> ListUsersAsync()
    {
        var listUsersPaginator = _IAMService.Paginator.ListUsers(new
ListUsersRequest());
        var users = new List<User>();

        await foreach (var response in listUsersPaginator.Responses)
        {
            users.AddRange(response.Users);
        }

        return users;
    }

    ///<summary>
    ///<summary>Remove a user from an IAM group.
    ///</summary>
    ///<param name="userName">The username of the user to remove.</param>
    ///<param name="groupName">The name of the IAM group to remove the user
from.</param>
    ///<returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> RemoveUserFromGroupAsync(string userName, string
groupName)
    {
        // Remove the user from the group.
        var removeUserRequest = new RemoveUserFromGroupRequest()
        {
            UserName = userName,
            GroupName = groupName,
        };

        var response = await
_IAMService.RemoveUserFromGroupAsync(removeUserRequest);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }

    ///<summary>

```

```

    ///>summary>
    ///<param name="groupName">The name of the IAM group.</param>
    ///<param name="policyName">The name of the IAM policy.</param>
    ///<param name="policyDocument">The policy document defining the IAM policy.</param>
    ///<returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> PutGroupPolicyAsync(string groupName, string
    policyName, string policyDocument)
    {
        var request = new PutGroupPolicyRequest
        {
            GroupName = groupName,
            PolicyName = policyName,
            PolicyDocument = policyDocument
        };

        var response = await _IAMService.PutGroupPolicyAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    ///<summary>
    /// Update the inline policy document embedded in a role.
    ///</summary>
    ///<param name="policyName">The name of the policy to embed.</param>
    ///<param name="roleName">The name of the role to update.</param>
    ///<param name="policyDocument">The policy document that defines the role.</param>
    ///<returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> PutRolePolicyAsync(string policyName, string roleName,
    string policyDocument)
    {
        var request = new PutRolePolicyRequest
        {
            PolicyName = policyName,
            RoleName = roleName,
            PolicyDocument = policyDocument
        };

        var response = await _IAMService.PutRolePolicyAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }

    ///<summary>
    /// Add or update an inline policy document that is embedded in an IAM user.
    ///</summary>
    ///<param name="userName">The name of the IAM user.</param>
    ///<param name="policyName">The name of the IAM policy.</param>
    ///<param name="policyDocument">The policy document defining the IAM policy.</param>
    ///<returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> PutUserPolicyAsync(string userName, string policyName,
    string policyDocument)
    {
        var request = new PutUserPolicyRequest
        {
            UserName = userName,
            PolicyName = policyName,
            PolicyDocument = policyDocument
        };

        var response = await _IAMService.PutUserPolicyAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

```

```

    ///<summary>
    /// Wait for a new access key to be ready to use.
    ///</summary>
    ///<param name="accessKeyId">The Id of the access key.</param>
    ///<returns>A boolean value indicating the success of the action.</returns>
    public async Task<bool> WaitUntilAccessKeyIsReady(string accessKeyId)
    {
        var keyReady = false;

        do
        {
            try
            {
                var response = await _IAMService.GetAccessKeyLastUsedAsync(
                    new GetAccessKeyLastUsedRequest { AccessKeyId = accessKeyId });
                if (response.UserName is not null)
                {
                    keyReady = true;
                }
            }
            catch (NoSuchEntityException)
            {
                keyReady = false;
            }
        } while (!keyReady);

        return keyReady;
    }
}

using Microsoft.Extensions.Configuration;
namespace IAMBasics;
public class IAMBasics
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for the AWS service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
                        LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices(_> services =>
                services.AddAWSService<IAmazonIdentityManagementService>()
                    .AddTransient<IAMWrapper>()
                    .AddTransient<UIWrapper>()
                    .Build());

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<IAMBasics>();

        IConfiguration configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load test settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally load local settings.
    }
}

```

```

    .Build();

    // Values needed for user, role, and policies.
    string userName = configuration["UserName"];
    string s3PolicyName = configuration["S3PolicyName"];
    string roleName = configuration["RoleName"];

    var iamWrapper = host.Services.GetRequiredService<IAMWrapper>();
    var uiWrapper = host.Services.GetRequiredService<UIWrapper>();

    uiWrapper.DisplayBasicsOverview();
    uiWrapper.PressEnter();

    // First create a user. By default, the new user has
    // no permissions.
    uiWrapper.DisplayTitle("Create User");
    Console.WriteLine($"Creating a new user with user name: {userName}.");
    var user = await iamWrapper.CreateUserAsync(userName);
    var userArn = user.Arn;

    Console.WriteLine($"Successfully created user: {userName} with ARN:
{userArn}.");
    uiWrapper.WaitABit(15, "Now let's wait for the user to be ready for use.");

    // Define a role policy document that allows the new user
    // to assume the role.
    string assumeRolePolicyDocument = "{" +
        "\"Version\": \"2012-10-17\", " +
        "\"Statement\": [{" +
            "\"Effect\": \"Allow\", " +
            "\"Principal\": {" +
                $"\"AWS\": \"{userArn}\", " +
            "}, " +
            "\"Action\": \"sts:AssumeRole\"" +
        "}]" +
    "}";

    // Permissions to list all buckets.
    string policyDocument = "{" +
        "\"Version\": \"2012-10-17\", " +
        " \"Statement\": [{" +
            " \"Action\": [\"s3>ListAllMyBuckets\"], " +
            " \"Effect\": \"Allow\", " +
            " \"Resource\": \"*\"" +
        "}]" +
    "}";

    // Create an AccessKey for the user.
    uiWrapper.DisplayTitle("Create access key");
    Console.WriteLine("Now let's create an access key for the new user.");
    var accessKey = await iamWrapper.CreateAccessKeyAsync(userName);

    var accessKeyId = accessKey.AccessKeyId;
    var secretAccessKey = accessKey.SecretAccessKey;

    Console.WriteLine($"We have created the access key with Access key id:
{accessKeyId}.");

    Console.WriteLine("Now let's wait until the IAM access key is ready to
use.");
    var keyReady = await iamWrapper.WaitUntilAccessKeyIsReady(accessKeyId);

    // Now try listing the Amazon Simple Storage Service (Amazon S3)
    // buckets. This should fail at this point because the user doesn't
    // have permissions to perform this task.

```

```

uiWrapper.DisplayTitle("Try to display Amazon S3 buckets");
Console.WriteLine("Now let's try to display a list of the user's Amazon S3
buckets.");
var s3Client1 = new AmazonS3Client(accessKeyId, secretAccessKey);
var stsClient1 = new AmazonSecurityTokenServiceClient(accessKeyId,
secretAccessKey);

var s3Wrapper = new S3Wrapper(s3Client1, stsClient1);
var buckets = await s3Wrapper.ListMyBucketsAsync();

Console.WriteLine(buckets is null
    ? "As expected, the call to list the buckets has returned a null list."
    : "Something went wrong. This shouldn't have worked.");

uiWrapper.PressEnter();

uiWrapper.DisplayTitle("Create IAM role");
Console.WriteLine($"Creating the role: {roleName}");

// Creating an IAM role to allow listing the S3 buckets. A role name
// is not case sensitive and must be unique to the account for which it
// is created.
var roleArn = await iamWrapper.CreateRoleAsync(roleName,
assumeRolePolicyDocument);

uiWrapper.PressEnter();

// Create a policy with permissions to list S3 buckets.
uiWrapper.DisplayTitle("Create IAM policy");
Console.WriteLine($"Creating the policy: {s3PolicyName}");
Console.WriteLine("with permissions to list the Amazon S3 buckets for the
account.");
var policy = await iamWrapper.CreatePolicyAsync(s3PolicyName,
policyDocument);

// Wait 15 seconds for the IAM policy to be available.
uiWrapper.WaitABit(15, "Waiting for the policy to be available.");

// Attach the policy to the role you created earlier.
uiWrapper.DisplayTitle("Attach new IAM policy");
Console.WriteLine("Now let's attach the policy to the role.");
await iamWrapper.AttachRolePolicyAsync(policy.Arn, roleName);

// Wait 15 seconds for the role to be updated.
Console.WriteLine();
uiWrapper.WaitABit(15, "Waiting for the policy to be attached.");

// Use the AWS Security Token Service (AWS STS) to have the user
// assume the role we created.
var stsClient2 = new AmazonSecurityTokenServiceClient(accessKeyId,
secretAccessKey);

// Wait for the new credentials to become valid.
uiWrapper.WaitABit(10, "Waiting for the credentials to be valid.");

var assumedRoleCredentials = await s3Wrapper.AssumeS3RoleAsync("temporary-
session", roleArn);

// Try again to list the buckets using the client created with
// the new user's credentials. This time, it should work.
var s3Client2 = new AmazonS3Client(assumedRoleCredentials);

s3Wrapper.UpdateClients(s3Client2, stsClient2);
buckets = await s3Wrapper.ListMyBucketsAsync();

```

```

        uiWrapper.DisplayTitle("List Amazon S3 buckets");
        Console.WriteLine("This time we should have buckets to list.");
        if (buckets is not null)
        {
            buckets.ForEach(bucket =>
            {
                Console.WriteLine($"{bucket.BucketName} created:
{bucket.CreationDate}");
            });
        }

        uiWrapper.PressEnter();

        // Now clean up all the resources used in the example.
        uiWrapper.DisplayTitle("Clean up resources");
        Console.WriteLine("Thank you for watching. The IAM Basics demo is
complete.");
        Console.WriteLine("Please wait while we clean up the resources we
created.");

        await iamWrapper.DetachRolePolicyAsync(policy.Arn, roleName);

        await iamWrapper.DeletePolicyAsync(policy.Arn);

        await iamWrapper.DeleteRoleAsync(roleName);

        await iamWrapper.DeleteAccessKeyAsync(accessKeyId, userName);

        await iamWrapper.DeleteUserAsync(userName);

        uiWrapper.PressEnter();

        Console.WriteLine("All done cleaning up our resources. Thank you for your
patience.");
    }
}

namespace IamScenariosCommon;

using System.Net;

/// <summary>
/// A class to perform Amazon Simple Storage Service (Amazon S3) actions for
/// the IAM Basics scenario.
/// </summary>
public class S3Wrapper
{
    private IAmazonS3 _s3Service;
    private IAmazonSecurityTokenService _stsService;

    /// <summary>
    /// Constructor for the S3Wrapper class.
    /// </summary>
    /// <param name="s3Service">An Amazon S3 client object.</param>
    /// <param name="stsService">An AWS Security Token Service (AWS STS)
    /// client object.</param>
    public S3Wrapper(IAmazonS3 s3Service, IAmazonSecurityTokenService stsService)
    {
        _s3Service = s3Service;
        _stsService = stsService;
    }

    // snippet.start:[STS.dotnetv3.AssumeS3Role]
    /// <summary>
    /// Assumes an AWS Identity and Access Management (IAM) role that allows

```

```

    ///> Amazon S3 access for the current session.
    ///> </summary>
    ///> <param name="roleSession">A string representing the current session.</param>
public async Task<Credentials> AssumeS3RoleAsync(string roleSession, string
roleToAssume)
{
    // Create the request to use with the AssumeRoleAsync call.
    var request = new AssumeRoleRequest()
    {
        RoleSessionName = roleSession,
        RoleArn = roleToAssume,
    };

    var response = await _stsService.AssumeRoleAsync(request);

    return response.Credentials;
}

// snippet.end:[STS.dotnetv3.AssumeS3Role]

    ///> <summary>
    ///> Delete an S3 bucket.
    ///> </summary>
    ///> <param name="bucketName">Name of the S3 bucket to delete.</param>
    ///> <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteBucketAsync(string bucketName)
{
    var result = await _s3Service.DeleteBucketAsync(new DeleteBucketRequest
{ BucketName = bucketName });
    return result.HttpStatusCode == HttpStatusCode.OK;
}

    ///> <summary>
    ///> List the buckets that are owned by the user's account.
    ///> </summary>
    ///> <returns>Async Task.</returns>
public async Task<List<S3Bucket>?> ListMyBucketsAsync()
{
    try
    {
        // Get the list of buckets accessible by the new user.
        var response = await _s3Service.ListBucketsAsync();

        return response.Buckets;
    }
    catch (AmazonS3Exception ex)
    {
        // Something else went wrong. Display the error message.
        Console.WriteLine($"Error: {ex.Message}");
        return null;
    }
}

    ///> <summary>
    ///> Create a new S3 bucket.
    ///> </summary>
    ///> <param name="bucketName">The name for the new bucket.</param>
    ///> <returns>A Boolean value indicating whether the action completed
    ///> successfully.</returns>
public async Task<bool> PutBucketAsync(string bucketName)
{
    var response = await _s3Service.PutBucketAsync(new PutBucketRequest
{ BucketName = bucketName });
}

```

```

        return response.HttpStatusCode == HttpStatusCode.OK;
    }

    ///<summary>
    /// Update the client objects with new client objects. This is available
    /// because the scenario uses the methods of this class without and then
    /// with the proper permissions to list S3 buckets.
    ///</summary>
    ///<param name="s3Service">The Amazon S3 client object.</param>
    ///<param name="stsService">The AWS STS client object.</param>
    public void UpdateClients(IAmazonS3 s3Service, IAmazonSecurityTokenService
stsService)
    {
        _s3Service = s3Service;
        _stsService = stsService;
    }
}

namespace IamScenariosCommon;

public class UIWrapper
{
    public readonly string SepBar = new(' ', Console.WindowWidth);

    ///<summary>
    /// Show information about the IAM Groups scenario.
    ///</summary>
    public void DisplayGroupsOverview()
    {
        Console.Clear();

        DisplayTitle("Welcome to the IAM Groups Demo");
        Console.WriteLine("This example application does the following:");
        Console.WriteLine("\t1. Creates an Amazon Identity and Access Management
(IAM) group.");
        Console.WriteLine("\t2. Adds an IAM policy to the IAM group giving it full
access to Amazon S3.");
        Console.WriteLine("\t3. Creates a new IAM user.");
        Console.WriteLine("\t4. Creates an IAM access key for the user.");
        Console.WriteLine("\t5. Adds the user to the IAM group.");
        Console.WriteLine("\t6. Lists the buckets on the account.");
        Console.WriteLine("\t7. Proves that the user has full Amazon S3 access by
creating a bucket.");
        Console.WriteLine("\t8. List the buckets again to show the new bucket.");
        Console.WriteLine("\t9. Cleans up all the resources created.");
    }

    ///<summary>
    /// Show information about the IAM Basics scenario.
    ///</summary>
    public void DisplayBasicsOverview()
    {
        Console.Clear();

        DisplayTitle("Welcome to IAM Basics");
        Console.WriteLine("This example application does the following:");
        Console.WriteLine("\t1. Creates a user with no permissions.");
        Console.WriteLine("\t2. Creates a role and policy that grant
s3>ListAllMyBuckets permission.");
        Console.WriteLine("\t3. Grants the user permission to assume the role.");
        Console.WriteLine("\t4. Creates an S3 client object as the user and tries
to list buckets (this will fail).");
        Console.WriteLine("\t5. Gets temporary credentials by assuming the role.");
        Console.WriteLine("\t6. Creates a new S3 client object with the temporary
credentials and lists the buckets (this will succeed).");
    }
}

```

```

        Console.WriteLine("\t7. Deletes all the resources.");
    }

    ///<summary>
    ///<summary>
    ///<summary>
    public void PressEnter()
    {
        Console.Write("\nPress <Enter> to continue. ");
        _ = Console.ReadLine();
        Console.WriteLine();
    }

    ///<summary>
    ///<summary>
    ///<param name="strToCenter">The string to be centered.</param>
    ///<returns>The padded string.</returns>
    public string CenterString(string strToCenter)
    {
        var padAmount = (Console.WindowWidth - strToCenter.Length) / 2;
        var leftPad = new string(' ', padAmount);
        return $"{leftPad}{strToCenter}";
    }

    ///<summary>
    ///<summary>
    ///<param name="strTitle">The string to be displayed.</param>
    public void DisplayTitle(string strTitle)
    {
        Console.WriteLine(SepBar);
        Console.WriteLine(CenterString(strTitle));
        Console.WriteLine(SepBar);
    }

    ///<summary>
    ///<summary>
    ///<param name="numSeconds">The number of seconds to wait.</param>
    public void WaitABit(int numSeconds, string msg)
    {
        Console.WriteLine(msg);

        // Wait for the requested number of seconds.
        for (int i = numSeconds; i > 0; i--)
        {
            System.Threading.Thread.Sleep(1000);
            Console.Write($"{i}...");
        }

        PressEnter();
    }
}

```

- For API details, see the following topics in *AWS SDK for .NET API Reference*.
 - [AttachRolePolicy](#)
 - [CreateAccessKey](#)
 - [CreatePolicy](#)
 - [CreateRole](#)

- [CreateUser](#)
- [DeleteAccessKey](#)
- [DeletePolicy](#)
- [DeleteRole](#)
- [DeleteUser](#)
- [DeleteUserPolicy](#)
- [DetachRolePolicy](#)
- [PutUserPolicy](#)

Bash

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function iam_create_user_assume_role
#
# Scenario to create an IAM user, create an IAM role, and apply the role to the
# user.
#
#      "IAM access" permissions are needed to run this code.
#      "STS assume role" permissions are needed to run this code. (Note: It might be
#      necessary to
#          create a custom policy).
#
# Returns:
#      0 - If successful.
#      1 - If an error occurred.
#####
function iam_create_user_assume_role() {
    if [ "$IAM_OPERATIONS_SOURCED" != "True" ]; then
        # shellcheck disable=SC1091
        source ./iam_operations.sh
    fi
}

echo_repeat "*" 88
echo "Welcome to the IAM create user and assume role demo."
echo
echo "This demo will create an IAM user, create an IAM role, and apply the role
to the user."
echo_repeat "*" 88
echo

echo -n "Enter a name for a new IAM user: "
get_input
user_name=$get_input_result

local user_arn
user_arn=$(iam_create_user -u "$user_name")

# shellcheck disable=SC2181
if [[ ${?} == 0 ]]; then
    echo "Created demo IAM user named $user_name"
else
    errecho "$user_arn"
```

```

errecho "The user failed to create. This demo will exit."
return 1
fi

local access_key_response
access_key_response=$(iam_create_user_access_key -u "$user_name")
# shellcheck disable=SC2181
if [[ ${?} != 0 ]]; then
    errecho "The access key failed to create. This demo will exit."
    clean_up "$user_name"
    return 1
fi

IFS=$'\t ' read -r -a access_key_values <<<"$access_key_response"
local key_name=${access_key_values[0]}
local key_secret=${access_key_values[1]}

echo "Created access key named $key_name"

echo "Wait 10 seconds for the user to be ready."
sleep 10
echo_repeat "*" 88
echo

local iam_role_name
iam_role_name=$(generate_random_name "test-role")
echo "Creating a role named $iam_role_name with user $user_name as the
principal."

local assume_role_policy_document={}
\\"Version\\": \"2012-10-17\",
\\"Statement\\": [{{
    \\"Effect\\": \"Allow\",
    \\"Principal\\": {\"AWS\": \"$user_arn\"},
    \\"Action\\": \"sts:AssumeRole\"
}}]
}""

local role_arn
role_arn=$(iam_create_role -n "$iam_role_name" -p "$assume_role_policy_document")

# shellcheck disable=SC2181
if [ ${?} == 0 ]; then
    echo "Created IAM role named $iam_role_name"
else
    errecho "The role failed to create. This demo will exit."
    clean_up "$user_name" "$key_name"
    return 1
fi

local policy_name
policy_name=$(generate_random_name "test-policy")
local policy_document={}
\\"Version\\": \"2012-10-17\",
\\"Statement\\": [{{
    \\"Effect\\": \"Allow\",
    \\"Action\\": \"s3>ListAllMyBuckets\",
    \\"Resource\\": \"arn:aws:s3:::*\"}}]}"

local policy_arn
policy_arn=$(iam_create_policy -n "$policy_name" -p "$policy_document")
# shellcheck disable=SC2181
if [[ ${?} == 0 ]]; then
    echo "Created IAM policy named $policy_name"
else
    errecho "The policy failed to create."

```

```

        clean_up "$user_name" "$key_name" "$iam_role_name"
        return 1
    fi

    if (iam_attach_role_policy -n "$iam_role_name" -p "$policy_arn"); then
        echo "Attached policy $policy_arn to role $iam_role_name"
    else
        errecho "The policy failed to attach."
        clean_up "$user_name" "$key_name" "$iam_role_name" "$policy_arn"
        return 1
    fi

    local assume_role_policy_document="{
        \"Version\": \"2012-10-17\",
        \"Statement\": [{{
            \"Effect\": \"Allow\",
            \"Action\": \"sts:AssumeRole\",
            \"Resource\": \"$role_arn\"}}]}"

    local assume_role_policy_name
    assume_role_policy_name=$(generate_random_name "test-assume-role-")

    # shellcheck disable=SC2181
    local assume_role_policy_arn
    assume_role_policy_arn=$(iam_create_policy -n "$assume_role_policy_name" -p
"$assume_role_policy_document")
    # shellcheck disable=SC2181
    if [ ${?} == 0 ]; then
        echo "Created IAM policy named $assume_role_policy_name for sts assume role"
    else
        errecho "The policy failed to create."
        clean_up "$user_name" "$key_name" "$iam_role_name" "$policy_arn" "$policy_arn"
        return 1
    fi

    echo "Wait 10 seconds to give AWS time to propagate these new resources and
connections."
    sleep 10
    echo_repeat "*" 88
    echo

    echo "Try to list buckets without the new user assuming the role."
    echo_repeat "*" 88
    echo

    # Set the environment variables for the created user.
    # bashsupport disable=BP2001
    export AWS_ACCESS_KEY_ID=$key_name
    # bashsupport disable=BP2001
    export AWS_SECRET_ACCESS_KEY=$key_secret

    local buckets
    buckets=$(s3_list_buckets)

    # shellcheck disable=SC2181
    if [ ${?} == 0 ]; then
        local bucket_count
        bucket_count=$(echo "$buckets" | wc -w | xargs)
        echo "There are $bucket_count buckets in the account. This should not have
happened."
    else
        errecho "Because the role with permissions has not been assumed, listing
buckets failed."
    fi

    echo

```

```

echo_repeat "*" 88
echo "Now assume the role $iam_role_name and list the buckets."
echo_repeat "*" 88
echo

local credentials

credentials=$(sts_assume_role -r "$role_arn" -n "AssumeRoleDemoSession")
# shellcheck disable=SC2181
if [ ${?} == 0 ]; then
    echo "Assumed role $iam_role_name"
else
    errecho "Failed to assume role."
    export AWS_ACCESS_KEY_ID=""
    export AWS_SECRET_ACCESS_KEY=""
    clean_up "$user_name" "$key_name" "$iam_role_name" "$policy_arn" "$policy_arn"
"$assume_role_policy_arn"
    return 1
fi

IFS=$'\t ' read -r -a credentials <<<"$credentials"

export AWS_ACCESS_KEY_ID=${credentials[0]}
export AWS_SECRET_ACCESS_KEY=${credentials[1]}
# bashsupport disable=BP2001
export AWS_SESSION_TOKEN=${credentials[2]}

buckets=$(s3_list_buckets)

# shellcheck disable=SC2181
if [ ${?} == 0 ]; then
    local bucket_count
    bucket_count=$(echo "$buckets" | wc -w | xargs)
    echo "There are $bucket_count buckets in the account. Listing buckets succeeded
because of "
    echo "the assumed role."
else
    errecho "Failed to list buckets. This should not happen."
    export AWS_ACCESS_KEY_ID=""
    export AWS_SECRET_ACCESS_KEY=""
    export AWS_SESSION_TOKEN=""
    clean_up "$user_name" "$key_name" "$iam_role_name" "$policy_arn" "$policy_arn"
"$assume_role_policy_arn"
    return 1
fi

local result=0
export AWS_ACCESS_KEY_ID=""
export AWS_SECRET_ACCESS_KEY=""

echo
echo_repeat "*" 88
echo "The created resources will now be deleted."
echo_repeat "*" 88
echo

clean_up "$user_name" "$key_name" "$iam_role_name" "$policy_arn" "$policy_arn"
"$assume_role_policy_arn"

# shellcheck disable=SC2181
if [[ ${?} -ne 0 ]]; then
    result=1
fi

return $result
}

```

```
#####
# function iam_user_exists
#
# This function checks to see if the specified AWS Identity and Access Management
# (IAM) user already exists.
#
# Parameters:
#     $1 - The name of the IAM user to check.
#
# Returns:
#     0 - If the user already exists.
#     1 - If the user doesn't exist.
#####
function iam_user_exists() {
    local user_name
    user_name=$1

    # Check whether the IAM user already exists.
    # We suppress all output - we're interested only in the return code.

    local errors
    errors=$(aws iam get-user \
        --user-name "$user_name" 2>&1 >/dev/null)

    local error_code=${?}

    if [[ $error_code -eq 0 ]]; then
        return 0 # 0 in Bash script means true.
    else
        if [[ $errors != *"error""*(NoSuchEntity)"* ]]; then
            aws_cli_error_log $error_code
            errecho "Error calling iam get-user $errors"
        fi

        return 1 # 1 in Bash script means false.
    fi
}

#####
# function iam_create_user
#
# This function creates the specified IAM user, unless
# it already exists.
#
# Parameters:
#     -u user_name -- The name of the user to create.
#
# Returns:
#     The ARN of the user.
#     And:
#         0 - If successful.
#         1 - If it fails.
#####
function iam_create_user() {
    local user_name response
    local option OPTARG # Required to use getopt command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_create_user"
        echo "Creates an AWS Identity and Access Management (IAM) user. You must supply"
        echo "a username:"
        echo "    -u user_name      The name of the user. It must be unique within the"
        echo "account."
        echo ""
    }
}
```

```

}

# Retrieve the calling parameters.
while getopts "u:h" option; do
    case "${option}" in
        u) user_name="${OPTARG}" ;;
        h)
            usage
            return 0
        ;;
        \?)
            echo "Invalid parameter"
            usage
            return 1
        ;;
    esac
done
export OPTIND=1

if [[ -z "$user_name" ]]; then
    errecho "ERROR: You must provide a username with the -u parameter."
    usage
    return 1
fi

iecho "Parameters:\n"
iecho "    User name: $user_name"
iecho ""

# If the user already exists, we don't want to try to create it.
if (iam_user_exists "$user_name"); then
    errecho "ERROR: A user with that name already exists in the account."
    return 1
fi

response=$(aws iam create-user --user-name "$user_name" \
    --output text \
    --query 'User.Arn')

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports create-user operation failed.$response"
    return 1
fi

echo "$response"

return 0
}

#####
# function iam_create_user_access_key
#
# This function creates an IAM access key for the specified user.
#
# Parameters:
#     -u user_name -- The name of the IAM user.
#     [-f file_name] -- The optional file name for the access key output.
#
# Returns:
#     [access_key_id access_key_secret]
#     And:
#         0 - If successful.
#         1 - If it fails.
#
# 
```

```
#####
function iam_create_user_access_key() {
    local user_name file_name response
    local option OPTARG # Required to use getopt command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_create_user_access_key"
        echo "Creates an AWS Identity and Access Management (IAM) key pair."
        echo " -u user_name   The name of the IAM user."
        echo " [-f file_name] Optional file name for the access key output."
        echo ""
    }

    # Retrieve the calling parameters.
    while getopts "u:f:h" option; do
        case "${option}" in
            u) user_name="${OPTARG}" ;;
            f) file_name="${OPTARG}" ;;
            h)
                usage
                return 0
            ;;
            \?)
                echo "Invalid parameter"
                usage
                return 1
            ;;
        esac
    done
    export OPTIND=1

    if [[ -z "$user_name" ]]; then
        errecho "ERROR: You must provide a username with the -u parameter."
        usage
        return 1
    fi

    response=$(aws iam create-access-key \
        --user-name "$user_name" \
        --output text)

    local error_code=${?}

    if [[ $error_code -ne 0 ]]; then
        aws_cli_error_log $error_code
        errecho "ERROR: AWS reports create-access-key operation failed.$response"
        return 1
    fi

    if [[ -n "$file_name" ]]; then
        echo "$response" >"$file_name"
    fi

    local key_id key_secret
    # shellcheck disable=SC2086
    key_id=$(echo $response | cut -f 2 -d ' ')
    # shellcheck disable=SC2086
    key_secret=$(echo $response | cut -f 4 -d ' ')

    echo "$key_id $key_secret"

    return 0
}

#####
```

```

# function iam_create_role
#
# This function creates an IAM role.
#
# Parameters:
#     -n role_name -- The name of the IAM role.
#     -p policy_json -- The assume role policy document.
#
# Returns:
#     The ARN of the role.
#     And:
#         0 - If successful.
#         1 - If it fails.
#####
function iam_create_role() {
    local role_name policy_document response
    local option OPTARG # Required to use getopts command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_create_user_access_key"
        echo "Creates an AWS Identity and Access Management (IAM) role."
        echo " -n role_name   The name of the IAM role."
        echo " -p policy_json -- The assume role policy document."
        echo ""
    }

    # Retrieve the calling parameters.
    while getopts "n:p:h" option; do
        case "${option}" in
            n) role_name="${OPTARG}" ;;
            p) policy_document="${OPTARG}" ;;
            h)
                usage
                return 0
                ;;
            \?)
                echo "Invalid parameter"
                usage
                return 1
                ;;
        esac
    done
    export OPTIND=1

    if [[ -z "$role_name" ]]; then
        errecho "ERROR: You must provide a role name with the -n parameter."
        usage
        return 1
    fi

    if [[ -z "$policy_document" ]]; then
        errecho "ERROR: You must provide a policy document with the -p parameter."
        usage
        return 1
    fi

    response=$(aws iam create-role \
        --role-name "$role_name" \
        --assume-role-policy-document "$policy_document" \
        --output text \
        --query Role.Arn)

    local error_code=${?}

    if [[ $error_code -ne 0 ]]; then

```

```

aws_cli_error_log $error_code
errecho "ERROR: AWS reports create-role operation failed.\n$response"
return 1
fi

echo "$response"

return 0
}

#####
# function iam_create_policy
#
# This function creates an IAM policy.
#
# Parameters:
#   -n policy_name -- The name of the IAM policy.
#   -p policy_json -- The policy document.
#
# Returns:
#   0 - If successful.
#   1 - If it fails.
#####
function iam_create_policy() {
    local policy_name policy_document response
    local option OPTARG # Required to use getopts command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_create_policy"
        echo "Creates an AWS Identity and Access Management (IAM) policy."
        echo " -n policy_name  The name of the IAM policy."
        echo " -p policy_json -- The policy document."
        echo ""
    }

    # Retrieve the calling parameters.
    while getopts "n:p:h" option; do
        case "${option}" in
            n) policy_name="${OPTARG}" ;;
            p) policy_document="${OPTARG}" ;;
            h)
                usage
                return 0
                ;;
            \?)
                echo "Invalid parameter"
                usage
                return 1
                ;;
        esac
    done
    export OPTIND=1

    if [[ -z "$policy_name" ]]; then
        errecho "ERROR: You must provide a policy name with the -n parameter."
        usage
        return 1
    fi

    if [[ -z "$policy_document" ]]; then
        errecho "ERROR: You must provide a policy document with the -p parameter."
        usage
        return 1
    fi
}

```

```

response=$(aws iam create-policy \
--policy-name "$policy_name" \
--policy-document "$policy_document" \
--output text \
--query Policy.Arn)

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports create-policy operation failed.\n$response"
    return 1
fi

echo "$response"
}

#####
# function iam_attach_role_policy
#
# This function attaches an IAM policy to a role.
#
# Parameters:
#     -n role_name -- The name of the IAM role.
#     -p policy_ARN -- The IAM policy document ARN..
#
# Returns:
#     0 - If successful.
#     1 - If it fails.
#####
function iam_attach_role_policy() {
    local role_name policy_arn response
    local option OPTARG # Required to use getopts command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_attach_role_policy"
        echo "Attaches an AWS Identity and Access Management (IAM) policy to an IAM"
        echo "role."
        echo "  -n role_name      The name of the IAM role."
        echo "  -p policy_ARN -- The IAM policy document ARN."
        echo ""
    }

    # Retrieve the calling parameters.
    while getopts "n:p:h" option; do
        case "${option}" in
            n) role_name="${OPTARG}";;
            p) policy_arn="${OPTARG}";;
            h)
                usage
                return 0
                ;;
            \?)
                echo "Invalid parameter"
                usage
                return 1
                ;;
        esac
    done
    export OPTIND=1

    if [[ -z "$role_name" ]]; then
        errecho "ERROR: You must provide a role name with the -n parameter."
        usage
        return 1
    fi
}

```

```

fi

if [[ -z "$policy_arn" ]]; then
    errecho "ERROR: You must provide a policy ARN with the -p parameter."
    usage
    return 1
fi

response=$(aws iam attach-role-policy \
    --role-name "$role_name" \
    --policy-arn "$policy_arn")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports attach-role-policy operation failed.\n$response"
    return 1
fi

echo "$response"

return 0
}

#####
# function iam_detach_role_policy
#
# This function detaches an IAM policy to a role.
#
# Parameters:
#     -n role_name -- The name of the IAM role.
#     -p policy_ARN -- The IAM policy document ARN..
#
# Returns:
#     0 - If successful.
#     1 - If it fails.
#####
function iam_detach_role_policy() {
    local role_name policy_arn response
    local option OPTARG # Required to use getopts command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_detach_role_policy"
        echo "Detaches an AWS Identity and Access Management (IAM) policy to an IAM role."
        echo " -n role_name   The name of the IAM role."
        echo " -p policy_ARN -- The IAM policy document ARN."
        echo ""
    }

    # Retrieve the calling parameters.
    while getopts "n:p:h" option; do
        case "${option}" in
            n) role_name="${OPTARG}" ;;
            p) policy_arn="${OPTARG}" ;;
            h)
                usage
                return 0
                ;;
            \?)
                echo "Invalid parameter"
                usage
                return 1
                ;;
        esac
    done
}

```

```

        esac
done
export OPTIND=1

if [[ -z "$role_name" ]]; then
    errecho "ERROR: You must provide a role name with the -n parameter."
    usage
    return 1
fi

if [[ -z "$policy_arn" ]]; then
    errecho "ERROR: You must provide a policy ARN with the -p parameter."
    usage
    return 1
fi

response=$(aws iam detach-role-policy \
    --role-name "$role_name" \
    --policy-arn "$policy_arn")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports detach-role-policy operation failed.\n$response"
    return 1
fi

echo "$response"

return 0
}

#####
# function iam_delete_policy
#
# This function deletes an IAM policy.
#
# Parameters:
#     -n policy_arn -- The name of the IAM policy arn.
#
# Returns:
#     0 - If successful.
#     1 - If it fails.
#####
function iam_delete_policy() {
    local policy_arn response
    local option OPTARG # Required to use getopt command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_delete_policy"
        echo "Deletes an WS Identity and Access Management (IAM) policy"
        echo "  -n policy_arn -- The name of the IAM policy arn."
        echo ""
    }

    # Retrieve the calling parameters.
    while getopts "n:h" option; do
        case "${option}" in
            n) policy_arn="${OPTARG}" ;;
            h)
                usage
                return 0
                ;;
            \?) ;;
        esac
    done
}

```

```

        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
done
export OPTIND=1

if [[ -z "$policy_arn" ]]; then
    errecho "ERROR: You must provide a policy arn with the -n parameter."
    usage
    return 1
fi

iecho "Parameters:\n"
iecho "    Policy arn: $policy_arn"
iecho ""

response=$(aws iam delete-policy \
    --policy-arn "$policy_arn")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-policy operation failed.\n$response"
    return 1
fi

iecho "delete-policy response:$response"
iecho

return 0
}

#####
# function iam_delete_role
#
# This function deletes an IAM role.
#
# Parameters:
#     -n role_name -- The name of the IAM role.
#
# Returns:
#     0 - If successful.
#     1 - If it fails.
#####
function iam_delete_role() {
    local role_name response
    local option OPTARG # Required to use getopts command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_delete_role"
        echo "Deletes an AWS Identity and Access Management (IAM) role"
        echo "    -n role_name -- The name of the IAM role."
        echo ""
    }

    # Retrieve the calling parameters.
    while getopts "n:h" option; do
        case "${option}" in
            n) role_name="${OPTARG}" ;;
            h)
                usage
                return 0
        esac
    done
}

```

```

;;
\?)"
echo "Invalid parameter"
usage
return 1
;;
esac
done
export OPTIND=1

echo "role_name:$role_name"
if [[ -z "$role_name" ]]; then
    errecho "ERROR: You must provide a role name with the -n parameter."
    usage
    return 1
fi

iecho "Parameters:\n"
iecho "    Role name: $role_name"
iecho ""

response=$(aws iam delete-role \
--role-name "$role_name")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-role operation failed.\n$response"
    return 1
fi

iecho "delete-role response:$response"
iecho

return 0
}

#####
# function iam_delete_access_key
#
# This function deletes an IAM access key for the specified IAM user.
#
# Parameters:
#     -u user_name -- The name of the user.
#     -k access_key -- The access key to delete.
#
# Returns:
#     0 - If successful.
#     1 - If it fails.
#####
function iam_delete_access_key() {
    local user_name access_key response
    local option OPTARG # Required to use getopt command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_delete_access_key"
        echo "Deletes an AWS Identity and Access Management (IAM) access key for the"
        echo "specified IAM user"
        echo "    -u user_name    The name of the user."
        echo "    -k access_key   The access key to delete."
        echo ""
    }

    # Retrieve the calling parameters.

```

```

while getopts "u:k:h" option; do
    case "${option}" in
        u) user_name="${OPTARG}" ;;
        k) access_key="${OPTARG}" ;;
        h)
            usage
            return 0
            ;;
        \?)
            echo "Invalid parameter"
            usage
            return 1
            ;;
    esac
done
export OPTIND=1

if [[ -z "$user_name" ]]; then
    errecho "ERROR: You must provide a username with the -u parameter."
    usage
    return 1
fi

if [[ -z "$access_key" ]]; then
    errecho "ERROR: You must provide an access key with the -k parameter."
    usage
    return 1
fi

iecho "Parameters:\n"
iecho "    Username: $user_name"
iecho "    Access key: $access_key"
iecho ""

response=$(aws iam delete-access-key \
    --user-name "$user_name" \
    --access-key-id "$access_key")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-access-key operation failed.\n$response"
    return 1
fi

iecho "delete-access-key response:$response"
iecho

return 0
}

#####
# function iam_delete_user
#
# This function deletes the specified IAM user.
#
# Parameters:
#     -u user_name -- The name of the user to create.
#
# Returns:
#     0 - If successful.
#     1 - If it fails.
#####
function iam_delete_user() {
    local user_name response

```

```

local option OPTARG # Required to use getopt command in a function.

# bashsupport disable=BP5008
function usage() {
    echo "function iam_delete_user"
    echo "Deletes an AWS Identity and Access Management (IAM) user. You must supply"
    echo "a username:"
    echo "  -u user_name      The name of the user."
    echo ""
}

# Retrieve the calling parameters.
while getopt "u:h" option; do
    case "${option}" in
        u) user_name="${OPTARG}" ;;
        h)
            usage
            return 0
            ;;
        \?)
            echo "Invalid parameter"
            usage
            return 1
            ;;
    esac
done
export OPTIND=1

if [[ -z "$user_name" ]]; then
    errecho "ERROR: You must provide a username with the -u parameter."
    usage
    return 1
fi

iecho "Parameters:\n"
iecho "  User name: $user_name"
iecho ""

# If the user does not exist, we don't want to try to delete it.
if (! iam_user_exists "$user_name"); then
    errecho "ERROR: A user with that name does not exist in the account."
    return 1
fi

response=$(aws iam delete-user \
    --user-name "$user_name")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-user operation failed.$response"
    return 1
fi

iecho "delete-user response:$response"
iecho

return 0
}

```

- For API details, see the following topics in *AWS CLI Command Reference*.
 - [AttachRolePolicy](#)

- [CreateAccessKey](#)
- [CreatePolicy](#)
- [CreateRole](#)
- [CreateUser](#)
- [DeleteAccessKey](#)
- [DeletePolicy](#)
- [DeleteRole](#)
- [DeleteUser](#)
- [DeleteUserPolicy](#)
- [DetachRolePolicy](#)
- [PutUserPolicy](#)

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
namespace AwsDoc {
    namespace IAM {

        //! Cleanup by deleting created entities.
        /*!
         * \sa DeleteCreatedEntities
         * \param client: IAM client.
         * \param role: IAM role.
         * \param user: IAM user.
         * \param policy: IAM policy.
        */
        static bool DeleteCreatedEntities(const Aws::IAM::IAMClient &client,
                                         const Aws::IAM::Model::Role &role,
                                         const Aws::IAM::Model::User &user,
                                         const Aws::IAM::Model::Policy &policy);
    }

    static const int LIST_BUCKETS_WAIT_SEC = 20;

    static const char ALLOCATION_TAG[] = "example_code";
}

//! Scenario to create an IAM user, create an IAM role, and apply the role to the
// user.
// "IAM access" permissions are needed to run this code.
// "STS assume role" permissions are needed to run this code. (Note: It might be
// necessary to
//     create a custom policy).
/*!
 * \sa iamCreateUserAssumeRoleScenario
 * \param clientConfig: Aws client configuration.
 * \return bool: Successful completion.
*/
bool AwsDoc::IAM::iamCreateUserAssumeRoleScenario(
    const Aws::Client::ClientConfiguration &clientConfig) {

    Aws::IAM::IAMClient client(clientConfig);
    Aws::IAM::Model::User user;
    Aws::IAM::Model::Role role;
```

```

Aws::IAM::Model::Policy policy;

// 1. Create a user.
{
    Aws::IAM::Model::CreateUserRequest request;
    Aws::String uuid = Aws::Utils::UUID::RandomUUID();
    Aws::String userName = "iam-demo-user-" +
        Aws::Utils::StringUtils::ToLower(uuid.c_str());
    request.SetUserName(userName);

    Aws::IAM::Model::CreateUserOutcome outcome = client.CreateUser(request);
    if (!outcome.IsSuccess()) {
        std::cout << "Error creating IAM user " << userName << ":" <<
            outcome.GetError().GetMessage() << std::endl;
        return false;
    }
    else {
        std::cout << "Successfully created IAM user " << userName << std::endl;
    }

    user = outcome.GetResult(). GetUser();
}

// 2. Create a role.
{
    // Get the IAM user for the current client in order to access its ARN.
    Aws::String iamUserArn;
    {
        Aws::IAM::Model:: GetUserRequest request;
        Aws::IAM::Model:: GetUserOutcome outcome = client.GetUser(request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Error getting Iam user. " <<
                outcome.GetError().GetMessage() << std::endl;

            DeleteCreatedEntities(client, role, user, policy);
            return false;
        }
        else {
            std::cout << "Successfully retrieved Iam user "
                << outcome.GetResult(). GetUser().GetUserName()
                << std::endl;
        }
    }

    iamUserArn = outcome.GetResult(). GetUser().GetArn();
}

Aws::IAM::Model::CreateRoleRequest request;

Aws::String uuid = Aws::Utils::UUID::RandomUUID();
Aws::String roleName = "iam-demo-role-" +
    Aws::Utils::StringUtils::ToLower(uuid.c_str());
request.SetRoleName(roleName);

// Build policy document for role.
Aws::Utils::Document jsonStatement;
jsonStatement.WithString("Effect", "Allow");

Aws::Utils::Document jsonPrincipal;
jsonPrincipal.WithString("AWS", iamUserArn);
jsonStatement.WithObject("Principal", jsonPrincipal);
jsonStatement.WithString("Action", "sts:AssumeRole");
jsonStatement.WithObject("Condition", Aws::Utils::Document());

Aws::Utils::Document policyDocument;
policyDocument.WithString("Version", "2012-10-17");

```

```

Aws::Utils::Array< Aws::Utils::Document> statements(1);
statements[0] = jsonStatement;
policyDocument.WithArray("Statement", statements);

std::cout << "Setting policy for role\n    "
             << policyDocument.View().WriteCompact() << std::endl;

// Set role policy document as JSON string.
request.SetAssumeRolePolicyDocument(policyDocument.View().WriteCompact());

Aws::IAM::Model::CreateRoleOutcome outcome = client.CreateRole(request);
if (!outcome.IsSuccess()) {
    std::cerr << "Error creating role. " <<
        outcome.GetError().GetMessage() << std::endl;

    DeleteCreatedEntities(client, role, user, policy);
    return false;
}
else {
    std::cout << "Successfully created a role with name " << roleName
             << std::endl;
}

role = outcome.GetResult().GetRole();
}

// 3. Create an IAM policy.
{
    Aws::IAM::Model::CreatePolicyRequest request;
    Aws::String uuid = Aws::Utils::UUID::RandomUUID();
    Aws::String policyName = "iam-demo-policy-" +
                                Aws::Utils::StringUtils::ToLower(uuid.c_str());
    request.SetPolicyName(policyName);

    // Build IAM policy document.
    Aws::Utils::Document jsonStatement;
    jsonStatement.WithString("Effect", "Allow");
    jsonStatement.WithString("Action", "s3>ListAllMyBuckets");
    jsonStatement.WithString("Resource", "arn:aws:s3:::*");

    Aws::Utils::Document policyDocument;
    policyDocument.WithString("Version", "2012-10-17");

    Aws::Utils::Array< Aws::Utils::Document> statements(1);
    statements[0] = jsonStatement;
    policyDocument.WithArray("Statement", statements);

    std::cout << "Creating a policy.\n    " <<
policyDocument.View().WriteCompact()
             << std::endl;

    // Set IAM policy document as JSON string.
    request.SetPolicyDocument(policyDocument.View().WriteCompact());

    Aws::IAM::Model::CreatePolicyOutcome outcome =
client.CreatePolicy(request);
    if (!outcome.IsSuccess()) {
        std::cerr << "Error creating policy. " <<
            outcome.GetError().GetMessage() << std::endl;

        DeleteCreatedEntities(client, role, user, policy);
        return false;
    }
    else {
        std::cout << "Successfully created a policy with name, " << policyName
        <<

```

```

        " ." << std::endl;
    }

    policy = outcome.GetResult().GetPolicy();
}

// 4. Assume the new role using the AWS Security Token Service (STS).
Aws::STS::Model::Credentials credentials;
{
    Aws::STS::STSClient stsClient(clientConfig);

    Aws::STS::Model::AssumeRoleRequest request;
    request.SetRoleArn(role.GetArn());
    Aws::String uuid = Aws::Utils::UUID::RandomUUID();
    Aws::String roleSessionName = "iam-demo-role-session-" +
        Aws::Utils::StringUtils::ToLower(uuid.c_str());
    request.SetRoleSessionName(roleSessionName);

    Aws::STS::Model::AssumeRoleOutcome assumeRoleOutcome;

    // Repeatedly call AssumeRole, because there is often a delay
    // before the role is available to be assumed.
    // Repeat at most 20 times when access is denied.
    int count = 0;
    while (true) {
        assumeRoleOutcome = stsClient.AssumeRole(request);
        if (!assumeRoleOutcome.IsSuccess()) {
            if (count > 20 || assumeRoleOutcome.GetError().GetErrorCode() !=
                Aws::STS::STSErrors::ACCESS_DENIED) {
                std::cerr << "Error assuming role after 20 tries. " <<
                    assumeRoleOutcome.GetError().GetMessage() <<
                std::endl;
                DeleteCreatedEntities(client, role, user, policy);
                return false;
            }
            std::this_thread::sleep_for(std::chrono::seconds(1));
        }
        else {
            std::cout << "Successfully assumed the role after " << count
                << " seconds." << std::endl;
            break;
        }
        count++;
    }
    credentials = assumeRoleOutcome.GetResult().GetCredentials();
}

// 5. List objects in the bucket (This should fail).
{
    Aws::S3::S3Client s3Client(
        Aws::Auth::AWSCredentials(credentials.GetAccessKeyId(),
            credentials.GetSecretAccessKey(),
            credentials.GetSessionToken()),
        Aws::MakeShared<Aws::S3::S3EndpointProvider>(ALLOCATION_TAG),
        clientConfig);
    Aws::S3::Model::ListBucketsOutcome listBucketsOutcome =
s3Client.ListBuckets();
    if (!listBucketsOutcome.IsSuccess()) {
        if (listBucketsOutcome.GetError().GetErrorCode() !=
            Aws::S3::S3Errors::ACCESS_DENIED) {
            std::cerr << "Could not lists buckets. " <<

```

```

        listBucketsOutcome.GetError().GetMessage() << std::endl;
    }
    else {
        std::cout
            << "Access to list buckets denied because privileges have
not been applied."
            << std::endl;
    }
    else {
        std::cerr
            << "Successfully retrieved bucket lists when this should not
happen."
            << std::endl;
    }
}

// 6. Attach the policy to the role.
{
    Aws::IAM::Model::AttachRolePolicyRequest request;
    request.SetRoleName(role.GetRoleName());
    request.WithPolicyArn(policy.GetArn());

    Aws::IAM::Model::AttachRolePolicyOutcome outcome = client.AttachRolePolicy(
        request);
    if (!outcome.IsSuccess()) {
        std::cerr << "Error creating policy. " <<
            outcome.GetError().GetMessage() << std::endl;

        DeleteCreatedEntities(client, role, user, policy);
        return false;
    }
    else {
        std::cout << "Successfully attached the policy with name, "
            << policy.GetPolicyName() <<
            ", to the role, " << role.GetRoleName() << "." << std::endl;
    }
}

int count = 0;
// 7. List objects in the bucket (this should succeed).
// Repeatedly call ListBuckets, because there is often a delay
// before the policy with ListBucket permissions has been applied to the role.
// Repeat at most LIST_BUCKETS_WAIT_SEC times when access is denied.
while (true) {
    Aws::S3::S3Client s3Client(
        Aws::Auth::AWSCredentials(credentials.GetAccessKeyId(),
            credentials.GetSecretAccessKey(),
            credentials.GetSessionToken()),
        Aws::MakeShared<Aws::S3::S3EndpointProvider>(ALLOCATION_TAG),
        clientConfig);
    Aws::S3::Model::ListBucketsOutcome listBucketsOutcome =
s3Client.ListBuckets();
    if (!listBucketsOutcome.IsSuccess()) {
        if ((count > LIST_BUCKETS_WAIT_SEC) ||
            listBucketsOutcome.GetError().GetErrorType() !=
            Aws::S3::S3Errors::ACCESS_DENIED) {
            std::cerr << "Could not lists buckets after " <<
LIST_BUCKETS_WAIT_SEC << " seconds. " <<
            listBucketsOutcome.GetError().GetMessage() << std::endl;
            DeleteCreatedEntities(client, role, user, policy);
            return false;
        }
        std::this_thread::sleep_for(std::chrono::seconds(1));
    }
}

```

```

        else {

            std::cout << "Successfully retrieved bucket lists after " << count
            << " seconds." << std::endl;
            break;
        }
        count++;
    }

    // 8. Delete all the created resources.
    return DeleteCreatedEntities(client, role, user, policy);
}

bool AwsDoc::IAM::DeleteCreatedEntities(const Aws::IAM::IAMClient &client,
                                         const Aws::IAM::Model::Role &role,
                                         const Aws::IAM::Model::User &user,
                                         const Aws::IAM::Model::Policy &policy) {
    bool result = true;
    if (policy.ArnHasBeenSet()) {
        // Detach the policy from the role.
        {
            Aws::IAM::Model::DetachRolePolicyRequest request;
            request.SetPolicyArn(policy.GetArn());
            request.SetRoleName(role.GetRoleName());

            Aws::IAM::Model::DetachRolePolicyOutcome outcome =
client.DetachRolePolicy(
                request);
            if (!outcome.IsSuccess()) {
                std::cerr << "Error Detaching policy from roles. " <<
                outcome.GetError().GetMessage() << std::endl;
                result = false;
            }
            else {
                std::cout << "Successfully detached the policy with arn "
                << policy.GetArn()
                << " from role " << role.GetRoleName() << "." <<
std::endl;
            }
        }

        // Delete the policy.
        {
            Aws::IAM::Model::DeletePolicyRequest request;
            request.WithPolicyArn(policy.GetArn());

            Aws::IAM::Model::DeletePolicyOutcome outcome =
client.DeletePolicy(request);
            if (!outcome.IsSuccess()) {
                std::cerr << "Error deleting policy. " <<
                outcome.GetError().GetMessage() << std::endl;
                result = false;
            }
            else {
                std::cout << "Successfully deleted the policy with arn "
                << policy.GetArn() << std::endl;
            }
        }

        if (role.RoleIdHasBeenSet()) {
            // Delete the role.
            Aws::IAM::Model::DeleteRoleRequest request;
            request.SetRoleName(role.GetRoleName());
        }
    }
}

```

```
Aws::IAM::Model::DeleteRoleOutcome outcome = client.DeleteRole(request);
if (!outcome.IsSuccess()) {
    std::cerr << "Error deleting role. " <<
        outcome.GetError().GetMessage() << std::endl;
    result = false;
}
else {
    std::cout << "Successfully deleted the role with name "
        << role.GetRoleName() << std::endl;
}
}

if (user.ArnHasBeenSet()) {
    // Delete the user.
    Aws::IAM::Model::DeleteUserRequest request;
    request.WithUserName(user.GetUserName());

    Aws::IAM::Model::DeleteUserOutcome outcome = client.DeleteUser(request);
    if (!outcome.IsSuccess()) {
        std::cerr << "Error deleting user. " <<
            outcome.GetError().GetMessage() << std::endl;
        result = false;
    }
    else {
        std::cout << "Successfully deleted the user with name "
            << user.GetUserName() << std::endl;
    }
}

return result;
}
```

- For API details, see the following topics in *AWS SDK for C++ API Reference*.
 - [AttachRolePolicy](#)
 - [CreateAccessKey](#)
 - [CreatePolicy](#)
 - [CreateRole](#)
 - [CreateUser](#)
 - [DeleteAccessKey](#)
 - [DeletePolicy](#)
 - [DeleteRole](#)
 - [DeleteUser](#)
 - [DeleteUserPolicy](#)
 - [DetachRolePolicy](#)
 - [PutUserPolicy](#)

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Run an interactive scenario at a command prompt.

```
// AssumeRoleScenario shows you how to use the AWS Identity and Access Management
// (IAM)
// service to perform the following actions:
//
// 1. Create a user who has no permissions.
// 2. Create a role that grants permission to list Amazon Simple Storage Service
//    (Amazon S3) buckets for the account.
// 3. Add a policy to let the user assume the role.
// 4. Try and fail to list buckets without permissions.
// 5. Assume the role and list S3 buckets using temporary credentials.
// 6. Delete the policy, role, and user.
type AssumeRoleScenario struct {
    sdkConfig aws.Config
    accountWrapper actions.AccountWrapper
    policyWrapper actions.PolicyWrapper
    roleWrapper actions.RoleWrapper
    userWrapper actions.UserWrapper
    questioner demotools.IQuestioner
    helper IScenarioHelper
    isTestRun bool
}

// NewAssumeRoleScenario constructs an AssumeRoleScenario instance from a
// configuration.
// It uses the specified config to get an IAM client and create wrappers for the
// actions
// used in the scenario.
func NewAssumeRoleScenario(sdkConfig aws.Config, questioner demotools.IQuestioner,
    helper IScenarioHelper) AssumeRoleScenario {
    iamClient := iam.NewFromConfig(sdkConfig)
    return AssumeRoleScenario{
        sdkConfig:   sdkConfig,
        accountWrapper: actions.AccountWrapper{IamClient: iamClient},
        policyWrapper: actions.PolicyWrapper{IamClient: iamClient},
        roleWrapper:   actions.RoleWrapper{IamClient: iamClient},
        userWrapper:   actions.UserWrapper{IamClient: iamClient},
        questioner:    questioner,
        helper:       helper,
    }
}

// addTestOptions appends the API options specified in the original configuration
// to
// another configuration. This is used to attach the middleware stubber to clients
// that are constructed during the scenario, which is needed for unit testing.
func (scenario AssumeRoleScenario) addTestOptions(scenarioConfig *aws.Config) {
    if scenario.isTestRun {
        scenarioConfig.APIOptions = append(scenarioConfig.APIOptions,
            scenario.sdkConfig.APIOptions...)
    }
}

// Run runs the interactive scenario.
func (scenario AssumeRoleScenario) Run() {
    defer func() {
        if r := recover(); r != nil {
            log.Printf("Something went wrong with the demo.\n")
            log.Println(r)
        }
    }()
}

log.Println(strings.Repeat("-", 88))
log.Println("Welcome to the AWS Identity and Access Management (IAM) assume role
demo.")
log.Println(strings.Repeat("-", 88))
```

```

user := scenario.CreateUser()
accessKey := scenario.CreateAccessKey(user)
role := scenario.CreateRoleAndPolicies(user)
noPermsConfig := scenario.ListBucketsWithoutPermissions(accessKey)
scenario.ListBucketsWithAssumedRole(noPermsConfig, role)
scenario.Cleanup(user, role)

log.Println(strings.Repeat("-", 88))
log.Println("Thanks for watching!")
log.Println(strings.Repeat("-", 88))
}

// CreateUser creates a new IAM user. This user has no permissions.
func (scenario AssumeRoleScenario) CreateUser() *types.User {
    log.Println("Let's create an example user with no permissions.")
    userName := scenario.questioner.Ask("Enter a name for the example user:",
        demotools.NotEmpty{})
    user, err := scenario.userWrapper.GetUser(userName)
    if err != nil {
        panic(err)
    }
    if user == nil {
        user, err = scenario.userWrapper.CreateUser(userName)
        if err != nil {
            panic(err)
        }
        log.Printf("Created user %v.\n", *user.UserName)
    } else {
        log.Printf("User %v already exists.\n", *user.UserName)
    }
    log.Println(strings.Repeat("-", 88))
    return user
}

// CreateAccessKey creates an access key for the user.
func (scenario AssumeRoleScenario) CreateAccessKey(user *types.User)
    *types.AccessKey {
    accessKey, err := scenario.userWrapper.CreateAccessKeyPair(*user.UserName)
    if err != nil {
        panic(err)
    }
    log.Printf("Created access key %v for your user.", *accessKey.AccessKeyId)
    log.Println("Waiting a few seconds for your user to be ready...")
    scenario.helper.Pause(10)
    log.Println(strings.Repeat("-", 88))
    return accessKey
}

// CreateRoleAndPolicies creates a policy that grants permission to list S3 buckets
for
// the current account and attaches the policy to a newly created role. It also
// adds an
// inline policy to the specified user that grants the user permission to assume
the role.
func (scenario AssumeRoleScenario) CreateRoleAndPolicies(user *types.User)
    *types.Role {
    log.Println("Let's create a role and policy that grant permission to list S3
buckets.")
    scenario.questioner.Ask("Press Enter when you're ready.")
    listBucketsRole, err := scenario.roleWrapper.CreateRole(scenario.helper.GetName(),
        *user.Arn)
    if err != nil {panic(err)}
    log.Printf("Created role %v.\n", *listBucketsRole.RoleName)
    listBucketsPolicy, err := scenario.policyWrapper.CreatePolicy(
        scenario.helper.GetName(), []string{"s3>ListAllMyBuckets"}, "arn:aws:s3:::*")
    if err != nil {panic(err)}
}

```

```

log.Printf("Created policy %v.\n", *listBucketsPolicy.PolicyName)
err = scenario.roleWrapper.AttachRolePolicy(*listBucketsPolicy.Arn,
*listBucketsRole.RoleName)
if err != nil {panic(err)}
log.Printf("Attached policy %v to role %v.\n", *listBucketsPolicy.PolicyName,
*listBucketsRole.RoleName)
err = scenario.userWrapper.CreateUserPolicy(*user.UserName,
scenario.helper.GetName(),
[]string{"sts:AssumeRole"}, *listBucketsRole.Arn)
if err != nil {panic(err)}
log.Printf("Created an inline policy for user %v that lets the user assume the
role.\n",
*user.UserName)
log.Println("Let's give AWS a few seconds to propagate these new resources and
connections...")
scenario.helper.Pause(10)
log.Println(strings.Repeat("-", 88))
return listBucketsRole
}

// ListBucketsWithoutPermissions creates an Amazon S3 client from the user's access
key
// credentials and tries to list buckets for the account. Because the user does not
have
// permission to perform this action, the action fails.
func (scenario AssumeRoleScenario) ListBucketsWithoutPermissions(accessKey
*types.AccessKey) *aws.Config {
log.Println("Let's try to list buckets without permissions. This should return an
AccessDenied error.")
scenario.questioner.Ask("Press Enter when you're ready.")
noPermsConfig, err := config.LoadDefaultConfig(context.TODO(),
config.WithCredentialsProvider(credentials.NewStaticCredentialsProvider(
*accessKey.AccessKeyId, *accessKey.SecretAccessKey, ""))
)
if err != nil {panic(err)}

// Add test options if this is a test run. This is needed only for testing
purposes.
scenario.addTestOptions(&noPermsConfig)

s3Client := s3.NewFromConfig(noPermsConfig)
_, err = s3Client.ListBuckets(context.TODO(), &s3.ListBucketsInput{})
if err != nil {
// The SDK for Go does not model the AccessDenied error, so check ErrorCode
directly.
var ae smithy.APIError
if errors.As(err, &ae) {
switch ae.ErrorCode() {
case "AccessDenied":
log.Println("Got AccessDenied error, which is the expected result because\n" +
"the ListBuckets call was made without permissions.")
default:
log.Println("Expected AccessDenied, got something else.")
panic(err)
}
}
} else {
log.Println("Expected AccessDenied error when calling ListBuckets without
permissions,\n" +
"but the call succeeded. Continuing the example anyway...")
}
log.Println(strings.Repeat("-", 88))
return &noPermsConfig
}

// ListBucketsWithAssumedRole performs the following actions:

```

```

// 1. Creates an AWS Security Token Service (AWS STS) client from the config
// created from
//   the user's access key credentials.
// 2. Gets temporary credentials by assuming the role that grants permission to
//   list the
//   buckets.
// 3. Creates an Amazon S3 client from the temporary credentials.
// 4. Lists buckets for the account. Because the temporary credentials are
//   generated by
//   assuming the role that grants permission, the action succeeds.
func (scenario AssumeRoleScenario) ListBucketsWithAssumedRole(noPermsConfig
    *aws.Config, role *types.Role) {
    log.Println("Let's assume the role that grants permission to list buckets and try
again.")
    scenario.questioner.Ask("Press Enter when you're ready.")
    stsClient := sts.NewFromConfig(*noPermsConfig)
    tempCredentials, err := stsClient.AssumeRole(context.TODO(), &sts.AssumeRoleInput{
        RoleArn:           role.Arn,
        RoleSessionName:  aws.String("AssumeRoleExampleSession"),
        DurationSeconds:   aws.Int32(900),
    })
    if err != nil {
        log.Printf("Couldn't assume role %v.\n", *role.RoleName)
        panic(err)
    }
    log.Printf("Assumed role %v, got temporary credentials.\n", *role.RoleName)
    assumeRoleConfig, err := config.LoadDefaultConfig(context.TODO(),
        config.WithCredentialsProvider(credentials.NewStaticCredentialsProvider(
            *tempCredentials.Credentials.AccessKeyId,
            *tempCredentials.Credentials.SecretAccessKey,
            *tempCredentials.Credentials.SessionToken),
    ),
)
    if err != nil {panic(err)}

    // Add test options if this is a test run. This is needed only for testing
    // purposes.
    scenario.addTestOptions(&assumeRoleConfig)

    s3Client := s3.NewFromConfig(assumeRoleConfig)
    result, err := s3Client.ListBuckets(context.TODO(), &s3.ListBucketsInput{})
    if err != nil {
        log.Println("Couldn't list buckets with assumed role credentials.")
        panic(err)
    }
    log.Println("Successfully called ListBuckets with assumed role credentials, \n" +
        "here are some of them:")
    for i := 0; i < len(result.Buckets) && i < 5; i++ {
        log.Printf("\t%v\n", *result.Buckets[i].Name)
    }
    log.Println(strings.Repeat("-", 88))
}

// Cleanup deletes all resources created for the scenario.
func (scenario AssumeRoleScenario) Cleanup(user *types.User, role *types.Role) {
    if scenario.questioner.AskBool(
        "Do you want to delete the resources created for this example? (y/n)", "y",
    ) {
        policies, err := scenario.roleWrapper.ListAttachedRolePolicies(*role.RoleName)
        if err != nil {panic(err)}
        for _, policy := range policies {
            err = scenario.roleWrapper.DetachRolePolicy(*role.RoleName, *policy.PolicyArn)
            if err != nil {panic(err)}
            err = scenario.policyWrapper.DeletePolicy(*policy.PolicyArn)
            if err != nil {panic(err)}
    }
}

```

```

        log.Printf("Detached policy %v from role %v and deleted the policy.\n",
                   *policy.PolicyName, *role.RoleName)
    }
    err = scenario.roleWrapper.DeleteRole(*role.RoleName)
    if err != nil {panic(err)}
    log.Printf("Deleted role %v.\n", *role.RoleName)

    userPols, err := scenario.userWrapper.ListUserPolicies(*user.UserName)
    if err != nil {panic(err)}
    for _, userPol := range userPols {
        err = scenario.userWrapper.DeleteUserPolicy(*user.UserName, userPol)
        if err != nil {panic(err)}
        log.Printf("Deleted policy %v from user %v.\n", userPol, *user.UserName)
    }
    keys, err := scenario.userWrapper.ListAccessKeys(*user.UserName)
    if err != nil {panic(err)}
    for _, key := range keys {
        err = scenario.userWrapper.DeleteAccessKey(*user.UserName, *key.AccessKeyId)
        if err != nil {panic(err)}
        log.Printf("Deleted access key %v from user %v.\n", *key.AccessKeyId,
                  *user.UserName)
    }
    err = scenario.userWrapper.DeleteUser(*user.UserName)
    if err != nil {panic(err)}
    log.Printf("Deleted user %v.\n", *user.UserName)
    log.Println(strings.Repeat("-", 88))
}
}

```

Define a struct that wraps account actions.

```

// AccountWrapper encapsulates AWS Identity and Access Management (IAM) account
// actions
// used in the examples.
// It contains an IAM service client that is used to perform account actions.
type AccountWrapper struct {
    IamClient *iam.Client
}

// GetAccountPasswordPolicy gets the account password policy for the current
// account.
// If no policy has been set, a NoSuchEntityException is error is returned.
func (wrapper AccountWrapper) GetAccountPasswordPolicy() (*types.PasswordPolicy,
    error) {
    var pwPolicy *types.PasswordPolicy
    result, err := wrapper.IamClient.GetAccountPasswordPolicy(context.TODO(),
        &iam.GetAccountPasswordPolicyInput{})
    if err != nil {
        log.Printf("Couldn't get account password policy. Here's why: %v\n", err)
    } else {
        pwPolicy = result.PasswordPolicy
    }
    return pwPolicy, err
}

// ListSAMLProviders gets the SAML providers for the account.

```

```
func (wrapper AccountWrapper) ListSAMLProviders() ([]types.SAMLProviderListEntry,
    error) {
    var providers []types.SAMLProviderListEntry
    result, err := wrapper.IamClient.ListSAMLProviders(context.TODO(),
    &iام.ListSAMLProvidersInput{})
    if err != nil {
        log.Printf("Couldn't list SAML providers. Here's why: %v\n", err)
    } else {
        providers = result.SAMLProviderList
    }
    return providers, err
}
```

Define a struct that wraps policy actions.

```
// PolicyDocument defines a policy document as a Go struct that can be serialized
// to JSON.
type PolicyDocument struct {
    Version string
    Statement []PolicyStatement
}

// PolicyStatement defines a statement in a policy document.
type PolicyStatement struct {
    Effect string
    Action []string
    Principal map[string]string `json:"",omitempty"`
    Resource *string `json:"",omitempty"`
}

// PolicyWrapper encapsulates AWS Identity and Access Management (IAM) policy
// actions
// used in the examples.
// It contains an IAM service client that is used to perform policy actions.
type PolicyWrapper struct {
    IamClient *iam.Client
}

// ListPolicies gets up to maxPolicies policies.
func (wrapper PolicyWrapper) ListPolicies(maxPolicies int32) ([]types.Policy,
    error) {
    var policies []types.Policy
    result, err := wrapper.IamClient.ListPolicies(context.TODO(),
    &iام.ListPoliciesInput{
        MaxItems: aws.Int32(maxPolicies),
    })
    if err != nil {
        log.Printf("Couldn't list policies. Here's why: %v\n", err)
    } else {
        policies = result.Policies
    }
    return policies, err
}

// CreatePolicy creates a policy that grants a list of actions to the specified
// resource.
```

```
// PolicyDocument shows how to work with a policy document as a data structure and
// serialize it to JSON by using Go's JSON marshaler.
func (wrapper PolicyWrapper) CreatePolicy(policyName string, actions []string,
    resourceArn string) (*types.Policy, error) {
    var policy *types.Policy
    policyDoc := PolicyDocument{
        Version: "2012-10-17",
        Statement: []PolicyStatement{
            Effect: "Allow",
            Action: actions,
            Resource: aws.String(resourceArn),
        },
    }
    policyBytes, err := json.Marshal(policyDoc)
    if err != nil {
        log.Printf("Couldn't create policy document for %v. Here's why: %v\n",
            resourceArn, err)
        return nil, err
    }
    result, err := wrapper.IamClient.CreatePolicy(context.TODO(),
        &iam.CreatePolicyInput{
            PolicyDocument: aws.String(string(policyBytes)),
            PolicyName:     aws.String(policyName),
        })
    if err != nil {
        log.Printf("Couldn't create policy %v. Here's why: %v\n", policyName, err)
    } else {
        policy = result.Policy
    }
    return policy, err
}

// GetPolicy gets data about a policy.
func (wrapper PolicyWrapper) GetPolicy(policyArn string) (*types.Policy, error) {
    var policy *types.Policy
    result, err := wrapper.IamClient.GetPolicy(context.TODO(), &iam.GetPolicyInput{
        PolicyArn: aws.String(policyArn),
    })
    if err != nil {
        log.Printf("Couldn't get policy %v. Here's why: %v\n", policyArn, err)
    } else {
        policy = result.Policy
    }
    return policy, err
}

// DeletePolicy deletes a policy.
func (wrapper PolicyWrapper) DeletePolicy(policyArn string) error {
    _, err := wrapper.IamClient.DeletePolicy(context.TODO(), &iam.DeletePolicyInput{
        PolicyArn: aws.String(policyArn),
    })
    if err != nil {
        log.Printf("Couldn't delete policy %v. Here's why: %v\n", policyArn, err)
    }
    return err
}
```

Define a struct that wraps role actions.

```
// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
// It contains an IAM service client that is used to perform role actions.
type RoleWrapper struct {
    IamClient *iam.Client
}

// ListRoles gets up to maxRoles roles.
func (wrapper RoleWrapper) ListRoles(maxRoles int32) ([]types.Role, error) {
    var roles []types.Role
    result, err := wrapper.IamClient.ListRoles(context.TODO(),
        &iam.ListRolesInput{MaxItems: aws.Int32(maxRoles)},
    )
    if err != nil {
        log.Printf("Couldn't list roles. Here's why: %v\n", err)
    } else {
        roles = result.Roles
    }
    return roles, err
}

// CreateRole creates a role that trusts a specified user. The trusted user can
// assume
// the role to acquire its permissions.
// PolicyDocument shows how to work with a policy document as a data structure and
// serialize it to JSON by using Go's JSON marshaler.
func (wrapper RoleWrapper) CreateRole(roleName string, trustedUserArn string) (*types.Role, error) {
    var role *types.Role
    trustPolicy := PolicyDocument{
        Version: "2012-10-17",
        Statement: []PolicyStatement{
            Effect: "Allow",
            Principal: map[string]string{"AWS": trustedUserArn},
            Action: []string{"sts:AssumeRole"},
        },
    }
    policyBytes, err := json.Marshal(trustPolicy)
    if err != nil {
        log.Printf("Couldn't create trust policy for %v. Here's why: %v\n",
            trustedUserArn, err)
        return nil, err
    }
    result, err := wrapper.IamClient.CreateRole(context.TODO(), &iam.CreateRoleInput{
        AssumeRolePolicyDocument: aws.String(string(policyBytes)),
        RoleName:                 aws.String(roleName),
    })
    if err != nil {
        log.Printf("Couldn't create role %v. Here's why: %v\n", roleName, err)
    } else {
        role = result.Role
    }
    return role, err
}

// GetRole gets data about a role.
func (wrapper RoleWrapper) GetRole(roleName string) (*types.Role, error) {
    var role *types.Role
```

```

result, err := wrapper.IamClient.GetRole(context.TODO(),
    &iam.GetRoleInput{RoleName: aws.String(roleName)})
if err != nil {
    log.Printf("Couldn't get role %v. Here's why: %v\n", roleName, err)
} else {
    role = result.Role
}
return role, err
}

// CreateServiceLinkedRole creates a service-linked role that is owned by the
// specified service.
func (wrapper RoleWrapper) CreateServiceLinkedRole(serviceName string, description
string) (*types.Role, error) {
var role *types.Role
result, err := wrapper.IamClient.CreateServiceLinkedRole(context.TODO(),
&iam.CreateServiceLinkedRoleInput{
    AWSServiceName: aws.String(serviceName),
    Description:   aws.String(description),
})
if err != nil {
    log.Printf("Couldn't create service-linked role %v. Here's why: %v\n",
serviceName, err)
} else {
    role = result.Role
}
return role, err
}

// DeleteServiceLinkedRole deletes a service-linked role.
func (wrapper RoleWrapper) DeleteServiceLinkedRole(roleName string) error {
_, err := wrapper.IamClient.DeleteServiceLinkedRole(context.TODO(),
&iam.DeleteServiceLinkedRoleInput{
    RoleName: aws.String(roleName)},
)
if err != nil {
    log.Printf("Couldn't delete service-linked role %v. Here's why: %v\n",
err)
}
return err
}

// AttachRolePolicy attaches a policy to a role.
func (wrapper RoleWrapper) AttachRolePolicy(policyArn string, roleName string)
error {
_, err := wrapper.IamClient.AttachRolePolicy(context.TODO(),
&iam.AttachRolePolicyInput{
    PolicyArn: aws.String(policyArn),
    RoleName:  aws.String(roleName),
})
if err != nil {
    log.Printf("Couldn't attach policy %v to role %v. Here's why: %v\n",
policyArn,
roleName, err)
}
return err
}

```

```
// ListAttachedRolePolicies lists the policies that are attached to the specified role.
func (wrapper RoleWrapper) ListAttachedRolePolicies(roleName string) ([]types.AttachedPolicy, error) {
    var policies []types.AttachedPolicy
    result, err := wrapper.IamClient.ListAttachedRolePolicies(context.TODO(),
        &iam.ListAttachedRolePoliciesInput{
            RoleName: aws.String(roleName),
        })
    if err != nil {
        log.Printf("Couldn't list attached policies for role %v. Here's why: %v\n",
            roleName, err)
    } else {
        policies = result.AttachedPolicies
    }
    return policies, err
}

// DetachRolePolicy detaches a policy from a role.
func (wrapper RoleWrapper) DetachRolePolicy(roleName string, policyArn string) error {
    _, err := wrapper.IamClient.DetachRolePolicy(context.TODO(),
        &iam.DetachRolePolicyInput{
            PolicyArn: aws.String(policyArn),
            RoleName: aws.String(roleName),
        })
    if err != nil {
        log.Printf("Couldn't detach policy from role %v. Here's why: %v\n", roleName,
            err)
    }
    return err
}

// ListRolePolicies lists the inline policies for a role.
func (wrapper RoleWrapper) ListRolePolicies(roleName string) ([]string, error) {
    var policies []string
    result, err := wrapper.IamClient.ListRolePolicies(context.TODO(),
        &iam.ListRolePoliciesInput{
            RoleName: aws.String(roleName),
        })
    if err != nil {
        log.Printf("Couldn't list policies for role %v. Here's why: %v\n", roleName, err)
    } else {
        policies = result.PolicyNames
    }
    return policies, err
}

// DeleteRole deletes a role. All attached policies must be detached before a
// role can be deleted.
func (wrapper RoleWrapper) DeleteRole(roleName string) error {
    _, err := wrapper.IamClient.DeleteRole(context.TODO(), &iam.DeleteRoleInput{
        RoleName: aws.String(roleName),
    })
    if err != nil {
        log.Printf("Couldn't delete role %v. Here's why: %v\n", roleName, err)
    }
    return err
}
```

Define a struct that wraps user actions.

```
// UserWrapper encapsulates user actions used in the examples.  
// It contains an IAM service client that is used to perform user actions.  
type UserWrapper struct {  
    IamClient *iam.Client  
}  
  
  
// ListUsers gets up to maxUsers number of users.  
func (wrapper UserWrapper) ListUsers(maxUsers int32) ([]types.User, error) {  
    var users []types.User  
    result, err := wrapper.IamClient.ListUsers(context.TODO(), &iam.ListUsersInput{  
        MaxItems: aws.Int32(maxUsers),  
    })  
    if err != nil {  
        log.Printf("Couldn't list users. Here's why: %v\n", err)  
    } else {  
        users = result.Users  
    }  
    return users, err  
}  
  
  
// GetUser gets data about a user.  
func (wrapper UserWrapper) GetUser(userName string) (*types.User, error) {  
    var user *types.User  
    result, err := wrapper.IamClient.GetUser(context.TODO(), &iam.GetUserInput{  
        UserName: aws.String(userName),  
    })  
    if err != nil {  
        var apiError smithy.APIError  
        if errors.As(err, &apiError) {  
            switch apiError.(type) {  
            case *types.NoSuchEntityException:  
                log.Printf("User %v does not exist.\n", userName)  
                err = nil  
            default:  
                log.Printf("Couldn't get user %v. Here's why: %v\n", userName, err)  
            }  
        }  
    } else {  
        user = result.User  
    }  
    return user, err  
}  
  
  
// CreateUser creates a new user with the specified name.  
func (wrapper UserWrapper) CreateUser(userName string) (*types.User, error) {  
    var user *types.User  
    result, err := wrapper.IamClient.CreateUser(context.TODO(), &iam.CreateUserInput{  
        UserName: aws.String(userName),  
    })  
    if err != nil {  
        log.Printf("Couldn't create user %v. Here's why: %v\n", userName, err)  
    } else {  
        user = result.User  
    }  
    return user, err  
}
```

```

    }

    return user, err
}

// CreateUserPolicy adds an inline policy to a user. This example creates a policy
// that grants a list of actions on a specified role.
// PolicyDocument shows how to work with a policy document as a data structure and
// serialize it to JSON by using Go's JSON marshaler.
func (wrapper UserWrapper) CreateUserPolicy(userName string, policyName string,
    actions []string,
    roleArn string) error {
    policyDoc := PolicyDocument{
        Version: "2012-10-17",
        Statement: []PolicyStatement{
            {
                Effect: "Allow",
                Action: actions,
                Resource: aws.String(roleArn),
            },
        },
    }
    policyBytes, err := json.Marshal(policyDoc)
    if err != nil {
        log.Printf("Couldn't create policy document for %v. Here's why: %v\n", roleArn, err)
        return err
    }
    _, err = wrapper.IamClient.PutUserPolicy(context.TODO(), &iam.PutUserPolicyInput{
        PolicyDocument: aws.String(string(policyBytes)),
        PolicyName:     aws.String(policyName),
        UserName:       aws.String(userName),
    })
    if err != nil {
        log.Printf("Couldn't create policy for user %v. Here's why: %v\n", userName, err)
    }
    return err
}

// ListUserPolicies lists the inline policies for the specified user.
func (wrapper UserWrapper) ListUserPolicies(userName string) ([]string, error) {
    var policies []string
    result, err := wrapper.IamClient.ListUserPolicies(context.TODO(),
        &iam.ListUserPoliciesInput{
            UserName: aws.String(userName),
        })
    if err != nil {
        log.Printf("Couldn't list policies for user %v. Here's why: %v\n", userName, err)
    } else {
        policies = result.PolicyNames
    }
    return policies, err
}

// DeleteUserPolicy deletes an inline policy from a user.
func (wrapper UserWrapper) DeleteUserPolicy(userName string, policyName string) error {
    _, err := wrapper.IamClient.DeleteUserPolicy(context.TODO(),
        &iam.DeleteUserPolicyInput{
            PolicyName: aws.String(policyName),
            UserName:   aws.String(userName),
        })
}

```

```

if err != nil {
    log.Printf("Couldn't delete policy from user %v. Here's why: %v\n", userName,
    err)
}
return err
}

// DeleteUser deletes a user.
func (wrapper UserWrapper) DeleteUser(userName string) error {
_, err := wrapper.IamClient.DeleteUser(context.TODO(), &iam.DeleteUserInput{
    UserName: aws.String(userName),
})
if err != nil {
    log.Printf("Couldn't delete user %v. Here's why: %v\n", userName, err)
}
return err
}

// CreateAccessKeyPair creates an access key for a user. The returned access key
// contains
// the ID and secret credentials needed to use the key.
func (wrapper UserWrapper) CreateAccessKeyPair(userName string) (*types.AccessKey,
    error) {
var key *types.AccessKey
result, err := wrapper.IamClient.CreateAccessKey(context.TODO(),
    &iam.CreateAccessKeyInput{
        UserName: aws.String(userName)})
if err != nil {
    log.Printf("Couldn't create access key pair for user %v. Here's why: %v\n",
    userName, err)
} else {
    key = result.AccessKey
}
return key, err
}

// DeleteAccessKey deletes an access key from a user.
func (wrapper UserWrapper) DeleteAccessKey(userName string, keyId string) error {
_, err := wrapper.IamClient.DeleteAccessKey(context.TODO(),
    &iam.DeleteAccessKeyInput{
        AccessKeyId: aws.String(keyId),
        UserName:    aws.String(userName),
})
if err != nil {
    log.Printf("Couldn't delete access key %v. Here's why: %v\n", keyId, err)
}
return err
}

// ListAccessKeys lists the access keys for the specified user.
func (wrapper UserWrapper) ListAccessKeys(userName string) ([]types.AccessKeyMetadata, error) {
var keys []types.AccessKeyMetadata
result, err := wrapper.IamClient.ListAccessKeys(context.TODO(),
    &iam.ListAccessKeysInput{
        UserName: aws.String(userName),
})
if err != nil {

```

```
    log.Printf("Couldn't list access keys for user %v. Here's why: %v\n", userName,
    err)
} else {
    keys = result.AccessKeyMetadata
}
return keys, err
}
```

- For API details, see the following topics in *AWS SDK for Go API Reference*.
 - [AttachRolePolicy](#)
 - [CreateAccessKey](#)
 - [CreatePolicy](#)
 - [CreateRole](#)
 - [CreateUser](#)
 - [DeleteAccessKey](#)
 - [DeletePolicy](#)
 - [DeleteRole](#)
 - [DeleteUser](#)
 - [DeleteUserPolicy](#)
 - [DetachRolePolicy](#)
 - [PutUserPolicy](#)

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Create functions that wrap IAM user actions.

```
/*
 To run this Java V2 code example, set up your development environment, including
 your credentials.

 For information, see this documentation topic:

 https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html

 This example performs these operations:

 1. Creates a user that has no permissions.
 2. Creates a role and policy that grants Amazon S3 permissions.
 3. Creates a role.
 4. Grants the user permissions.
 5. Gets temporary credentials by assuming the role. Creates an Amazon S3 Service
 client object with the temporary credentials.
 6. Deletes the resources.
 */

public class IAMScenario {
    public static final String DASHES = new String(new char[80]).replace("\0",
    "-");
    public static final String PolicyDocument =
        "{" +
```

```

    " \\"Version\": \\"2012-10-17\\", " +
    " \\"Statement\": [ " +
    "   { " +
    "     \\"Effect\\": \\"Allow\\", " +
    "     \\"Action\\": [ " +
    "       \\"s3:*\\\" " +
    "     ], " +
    "     \\"Resource\\": \\"*\\\" " +
    "   } " +
    " ] " +
    " }";

public static String userArn;
public static void main(String[] args) throws Exception {

    final String usage = "\n" +
        "Usage:\n" +
        "      <username> <policyName> <roleName> <roleSessionName> <bucketName>
\n\n" +
        "Where:\n" +
        "      username - The name of the IAM user to create. \n\n" +
        "      policyName - The name of the policy to create. \n\n" +
        "      roleName - The name of the role to create. \n\n" +
        "      roleSessionName - The name of the session required for the
assumeRole operation. \n\n" +
        "      bucketName - The name of the Amazon S3 bucket from which objects
are read. \n\n";

    if (args.length != 5) {
        System.out.println(usage);
        System.exit(1);
    }

    String userName = args[0];
    String policyName = args[1];
    String roleName = args[2];
    String roleSessionName = args[3];
    String bucketName = args[4];

    Region region = Region.AWS_GLOBAL;
    IamClient iam = IamClient.builder()
        .region(region)
        .credentialsProvider(ProfileCredentialsProvider.create())
        .build();

    System.out.println(DASHES);
    System.out.println("Welcome to the AWS IAM example scenario.");
    System.out.println(DASHES);

    System.out.println(DASHES);
    System.out.println(" 1. Create the IAM user.");
    User createUser = createIAMUser(iam, userName);

    System.out.println(DASHES);
    userArn = createUser.arn();

    AccessKey myKey = createIAMAccessKey(iam, userName);
    String accessKey = myKey.accessKeyId();
    String secretKey = myKey.secretAccessKey();
    String assumeRolePolicyDocument = "{" +
        "\\"Version\\": \\"2012-10-17\\", " +
        "\\"Statement\\": [{" +
        "\\"Effect\\": \\"Allow\\", " +
        "\\"Principal\\": {" +
        " \\"AWS\\": \" + userArn + "\"" +
        "}, " +

```

```

        "\"Action\": \"sts:AssumeRole\""
    "}]"
    "}";

System.out.println(assumeRolePolicyDocument);
System.out.println(userName + " was successfully created.");
System.out.println(DASHES);
System.out.println("2. Creates a policy.");
String polArn = createIAMPolicy(iam, policyName);
System.out.println("The policy " + polArn + " was successfully created.");
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("3. Creates a role.");
TimeUnit.SECONDS.sleep(30);
String roleArn = createIAMRole(iam, roleName, assumeRolePolicyDocument);
System.out.println(roleArn + " was successfully created.");
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("4. Grants the user permissions.");
attachIAMRolePolicy(iam, roleName, polArn);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("*** Wait for 30 secs so the resource is available");
TimeUnit.SECONDS.sleep(30);
System.out.println("5. Gets temporary credentials by assuming the role.");
System.out.println("Perform an Amazon S3 Service operation using the
temporary credentials.");
assumeRole(roleArn, roleSessionName, bucketName, accessKey, secretKey);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("6 Getting ready to delete the AWS resources");
deleteKey(iam, userName, accessKey );
deleteRole(iam, roleName, polArn);
deleteIAMUser(iam, userName);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("This IAM Scenario has successfully completed");
System.out.println(DASHES);
}

public static AccessKey createIAMAccessKey(IamClient iam, String user) {
    try {
        CreateAccessKeyRequest request = CreateAccessKeyRequest.builder()
            .userName(user)
            .build();

        CreateAccessKeyResponse response = iam.createAccessKey(request);
        return response.accessKey();

    } catch (IamException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
    return null;
}

public static User createIAMUser(IamClient iam, String username) {
    try {
        // Create an IamWaiter object
        IamWaiter iamWaiter = iam.waiter();
        CreateUserRequest request = CreateUserRequest.builder()

```

```

        .userName(username)
        .build();

        // Wait until the user is created.
        CreateUserResponse response = iam.createUser(request);
        GetUserRequest userRequest = GetUserRequest.builder()
            .userName(response.user().userName())
            .build();

        WaiterResponse< GetUserResponse> waitUntilUserExists =
        iamWaiter.waitUntilUserExists(userRequest);

        waitUntilUserExists.matched().response().ifPresent(System.out::println);
        return response.user();

    } catch (IamException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
    return null;
}

public static String createIAMRole(IamClient iam, String rolename, String json)
{
    try {
        CreateRoleRequest request = CreateRoleRequest.builder()
            .roleName(rolename)
            .assumeRolePolicyDocument(json)
            .description("Created using the AWS SDK for Java")
            .build();

        CreateRoleResponse response = iam.createRole(request);
        System.out.println("The ARN of the role is " + response.role().arn());
        return response.role().arn();

    } catch (IamException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
    return "";
}

public static String createIAMPolicy(IamClient iam, String policyName) {
    try {
        // Create an IamWaiter object.
        IamWaiter iamWaiter = iam.waiter();
        CreatePolicyRequest request = CreatePolicyRequest.builder()
            .policyName(policyName)
            .policyDocument(PolicyDocument).build();

        CreatePolicyResponse response = iam.createPolicy(request);
        GetPolicyRequest polRequest = GetPolicyRequest.builder()
            .policyArn(response.policy().arn())
            .build();

        WaiterResponse< GetPolicyResponse> waitUntilPolicyExists =
        iamWaiter.waitUntilPolicyExists(polRequest);

        waitUntilPolicyExists.matched().response().ifPresent(System.out::println);
        return response.policy().arn();

    } catch (IamException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}

```

```

        return "";
    }

    public static void attachIAMRolePolicy(IamClient iam, String roleName, String
policyArn) {
        try {
            ListAttachedRolePoliciesRequest request =
ListAttachedRolePoliciesRequest.builder()
                .roleName(roleName)
                .build();

            ListAttachedRolePoliciesResponse response =
iam.listAttachedRolePolicies(request);
            List<AttachedPolicy> attachedPolicies = response.attachedPolicies();
            String polArn;
            for (AttachedPolicy policy : attachedPolicies) {
                polArn = policy.policyArn();
                if (polArn.compareTo(policyArn) == 0) {
                    System.out.println(roleName + " policy is already attached to
this role.");
                }
            }
        }

        AttachRolePolicyRequest attachRequest =
AttachRolePolicyRequest.builder()
            .roleName(roleName)
            .policyArn(policyArn)
            .build();

        iam.attachRolePolicy(attachRequest);
        System.out.println("Successfully attached policy " + policyArn + " to
role " + roleName);

    } catch (IamException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}

// Invoke an Amazon S3 operation using the Assumed Role.
public static void assumeRole(String roleArn, String roleSessionName, String
bucketName, String keyVal, String keySecret) {

    // Use the creds of the new IAM user that was created in this code example.
    AwsBasicCredentials credentials = AwsBasicCredentials.create(keyVal,
keySecret);
    StsClient stsClient = StsClient.builder()
        .region(Region.US_EAST_1)
        .credentialsProvider(StaticCredentialsProvider.create(credentials))
        .build();

    try {
        AssumeRoleRequest roleRequest = AssumeRoleRequest.builder()
            .roleArn(roleArn)
            .roleSessionName(roleSessionName)
            .build();

        AssumeRoleResponse roleResponse = stsClient.assumeRole(roleRequest);
        Credentials myCreds = roleResponse.credentials();
        String key = myCreds.accessKeyId();
        String secKey = myCreds.secretAccessKey();
        String secToken = myCreds.sessionToken();

        // List all objects in an Amazon S3 bucket using the temp creds
retrieved by invoking assumeRole.
    }
}

```

```

Region region = Region.US_EAST_1;
S3Client s3 = S3Client.builder()

.credentialsProvider(StaticCredentialsProvider.create(AwsSessionCredentials.create(key,
secKey, secToken)))
.region(region)
.build();

System.out.println("Created a S3Client using temp credentials.");
System.out.println("Listing objects in " + bucketName);
ListObjectsRequest listObjects = ListObjectsRequest.builder()
.bucket(bucketName)
.build();

ListObjectsResponse res = s3.listObjects(listObjects);
List<S3Object> objects = res.contents();
for (S3Object myValue : objects) {
    System.out.println("The name of the key is " + myValue.key());
    System.out.println("The owner is " + myValue.owner());
}

} catch (StsException e) {
    System.err.println(e.getMessage());
    System.exit(1);
}
}
public static void deleteRole(IamClient iam, String roleName, String polArn) {

try {
    // First the policy needs to be detached.
    DetachRolePolicyRequest rolePolicyRequest =
DetachRolePolicyRequest.builder()
    .policyArn(polArn)
    .roleName(roleName)
    .build();

    iam.detachRolePolicy(rolePolicyRequest);

    // Delete the policy.
    DeletePolicyRequest request = DeletePolicyRequest.builder()
    .policyArn(polArn)
    .build();

    iam.deletePolicy(request);
    System.out.println("*** Successfully deleted " + polArn);

    // Delete the role.
    DeleteRoleRequest roleRequest = DeleteRoleRequest.builder()
    .roleName(roleName)
    .build();

    iam.deleteRole(roleRequest);
    System.out.println("*** Successfully deleted " + roleName);

} catch (IamException e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}
}

public static void deleteKey(IamClient iam ,String username, String accessKey )
{
try {
    DeleteAccessKeyRequest request = DeleteAccessKeyRequest.builder()
    .accessKeyId(accessKey)
    .userName(username)
}
}

```

```

        .build();

    iam.deleteAccessKey(request);
    System.out.println("Successfully deleted access key " + accessKey +
        " from user " + username);

} catch (IamException e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}

public static void deleteIAMUser(IamClient iam, String userName) {
    try {
        DeleteUserRequest request = DeleteUserRequest.builder()
            .userName(userName)
            .build();

        iam.deleteUser(request);
        System.out.println("**** Successfully deleted " + userName);

    } catch (IamException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
}

```

- For API details, see the following topics in *AWS SDK for Java 2.x API Reference*.
 - [AttachRolePolicy](#)
 - [CreateAccessKey](#)
 - [CreatePolicy](#)
 - [CreateRole](#)
 - [CreateUser](#)
 - [DeleteAccessKey](#)
 - [DeletePolicy](#)
 - [DeleteRole](#)
 - [DeleteUser](#)
 - [DeleteUserPolicy](#)
 - [DetachRolePolicy](#)
 - [PutUserPolicy](#)

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Create an IAM user and a role that grants permission to list Amazon S3 buckets. The user has rights only to assume the role. After assuming the role, use temporary credentials to list buckets for the account.

```

import {
  CreateUserCommand,
  CreateAccessKeyCommand,
}

```

```

CreatePolicyCommand,
CreateRoleCommand,
AttachRolePolicyCommand,
DeleteAccessKeyCommand,
DeleteUserCommand,
DeleteRoleCommand,
DeletePolicyCommand,
DetachRolePolicyCommand,
IAMClient,
} from "@aws-sdk/client-iam";
import { ListBucketsCommand, S3Client } from "@aws-sdk/client-s3";
import { AssumeRoleCommand, STSClient } from "@aws-sdk/client-sts";
import { retry } from "libs/utils/util-timers.js";

// Set the parameters.
const iamClient = new IAMClient({});
const userName = "test_name";
const policyName = "test_policy";
const roleName = "test_role";

export const main = async () => {
    // Create a user. The user has no permissions by default.
    const { User } = await iamClient.send(
        new CreateUserCommand({ UserName: userName })
    );

    if (!User) {
        throw new Error("User not created");
    }

    // Create an access key. This key is used to authenticate the new user to
    // Amazon Simple Storage Service (Amazon S3) and AWS Security Token Service (AWS
    // STS).
    // It's not best practice to use access keys. For more information, see https://aws.amazon.com/iam/resources/best-practices/.
    const createAccessKeyResponse = await iamClient.send(
        new CreateAccessKeyCommand({ UserName: userName })
    );

    if (
        !createAccessKeyResponse.AccessKey?.AccessKeyId ||
        !createAccessKeyResponse.AccessKey?.SecretAccessKey
    ) {
        throw new Error("Access key not created");
    }

    const {
        AccessKey: { AccessKeyId, SecretAccessKey },
    } = createAccessKeyResponse;

    let s3Client = new S3Client({
        credentials: {
            accessKeyId: AccessKeyId,
            secretAccessKey: SecretAccessKey,
        },
    });

    // Retry the list buckets operation until it succeeds. InvalidAccessKeyId is
    // thrown while the user and access keys are still stabilizing.
    await retry({ intervalInMs: 1000, maxRetries: 300 }, async () => {
        try {
            return await listBuckets(s3Client);
        } catch (err) {
            if (err instanceof Error && err.name === "InvalidAccessKeyId") {
                throw err;
            }
        }
    });
}

```

```

    });

    // Retry the create role operation until it succeeds. A MalformedPolicyDocument
error
// is thrown while the user and access keys are still stabilizing.
const { Role } = await retry(
{
    intervalInMs: 2000,
    maxRetries: 60,
},
() =>
iamClient.send(
    new CreateRoleCommand({
        AssumeRolePolicyDocument: JSON.stringify({
            Version: "2012-10-17",
            Statement: [
                {
                    Effect: "Allow",
                    Principal: {
                        // Allow the previously created user to assume this role.
                        AWS: User.Arn,
                    },
                    Action: "sts:AssumeRole",
                },
                ],
            ],
        RoleName: roleName,
    })
);
}

if (!Role) {
    throw new Error("Role not created");
}

// Create a policy that allows the user to list S3 buckets.
const { Policy: listBucketPolicy } = await iamClient.send(
    new CreatePolicyCommand({
        PolicyDocument: JSON.stringify({
            Version: "2012-10-17",
            Statement: [
                {
                    Effect: "Allow",
                    Action: ["s3>ListAllMyBuckets"],
                    Resource: "*",
                },
                ],
            ],
        PolicyName: policyName,
    })
);

if (!listBucketPolicy) {
    throw new Error("Policy not created");
}

// Attach the policy granting the 's3>ListAllMyBuckets' action to the role.
await iamClient.send(
    new AttachRolePolicyCommand({
        PolicyArn: listBucketPolicy.Arn,
        RoleName: Role.RoleName,
    })
);

// Assume the role.

```

```

const stsClient = new STSClient({
  credentials: {
    accessKeyId: AccessKeyId,
    secretAccessKey: SecretAccessKey,
  },
});

// Retry the assume role operation until it succeeds.
const { Credentials } = await retry(
  { intervalInMs: 2000, maxRetries: 60 },
  () =>
    stsClient.send(
      new AssumeRoleCommand({
        RoleArn: Role.Arn,
        RoleSessionName: `iamBasicScenarioSession-${Math.floor(
          Math.random() * 1000000
        )}`,
        DurationSeconds: 900,
      })
    )
);

if (!Credentials?.AccessKeyId || !Credentials?.SecretAccessKey) {
  throw new Error("Credentials not created");
}

s3Client = new S3Client({
  credentials: {
    accessKeyId: Credentials.AccessKeyId,
    secretAccessKey: Credentials.SecretAccessKey,
    sessionToken: Credentials.SessionToken,
  },
});

// List the S3 buckets again.
// Retry the list buckets operation until it succeeds. AccessDenied might
// be thrown while the role policy is still stabilizing.
await retry({ intervalInMs: 2000, maxRetries: 60 }, () =>
  listBuckets(s3Client)
);

// Clean up.
await iamClient.send(
  new DetachRolePolicyCommand({
    PolicyArn: listBucketPolicy.Arn,
    RoleName: Role.RoleName,
  })
);

await iamClient.send(
  new DeletePolicyCommand({
    PolicyArn: listBucketPolicy.Arn,
  })
);

await iamClient.send(
  new DeleteRoleCommand({
    RoleName: Role.RoleName,
  })
);

await iamClient.send(
  new DeleteAccessKeyCommand({
    UserName: userName,
    AccessKeyId,
  })
);

```

```

);
await iamClient.send(
  new DeleteUserCommand({
    UserName: userName,
  })
);
};

/**
 *
 * @param {S3Client} s3Client
 */
const listBuckets = async (s3Client) => {
  const { Buckets } = await s3Client.send(new ListBucketsCommand({}));

  if (!Buckets) {
    throw new Error("Buckets not listed");
  }

  console.log(Buckets.map((bucket) => bucket.Name).join("\n"));
};

```

- For API details, see the following topics in *AWS SDK for JavaScript API Reference*.
 - [AttachRolePolicy](#)
 - [CreateAccessKey](#)
 - [CreatePolicy](#)
 - [CreateRole](#)
 - [CreateUser](#)
 - [DeleteAccessKey](#)
 - [DeletePolicy](#)
 - [DeleteRole](#)
 - [DeleteUser](#)
 - [DeleteUserPolicy](#)
 - [DetachRolePolicy](#)
 - [PutUserPolicy](#)

Kotlin

SDK for Kotlin

Note

This is prerelease documentation for a feature in preview release. It is subject to change.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Create functions that wrap IAM user actions.

```

suspend fun main(args: Array<String>) {

  val usage = """
  Usage:
    <username> <policyName> <roleName> <roleSessionName> <fileLocation>
    <bucketName>
  
```

Where:

```

username - The name of the IAM user to create.
policyName - The name of the policy to create.
roleName - The name of the role to create.
roleSessionName - The name of the session required for the assumeRole
operation.
fileLocation - The file location to the JSON required to create the role
(see Readme).
bucketName - The name of the Amazon S3 bucket from which objects are read.
"""

if (args.size != 6) {
    println(usage)
    exitProcess(1)
}

val userName = args[0]
val policyName = args[1]
val roleName = args[2]
val roleSessionName = args[3]
val fileLocation = args[4]
val bucketName = args[5]

createUser(userName)
println("$userName was successfully created.")

val polArn = createPolicy(policyName)
println("The policy $polArn was successfully created.")

val roleArn = createRole(roleName, fileLocation)
println("$roleArn was successfully created.")
attachRolePolicy(roleName, polArn)

println("**** Wait for 1 MIN so the resource is available.")
delay(60000)
assumeGivenRole(roleArn, roleSessionName, bucketName)

println("**** Getting ready to delete the AWS resources.")
deleteRole(roleName, polArn)
deleteUser(userName)
println("This IAM Scenario has successfully completed.")
}

suspend fun createUser(usernameVal: String?): String? {

    val request = CreateUserRequest {
        userName = usernameVal
    }

    IamClient { region = "AWS_GLOBAL" }.use { iamClient ->
        val response = iamClient.createUser(request)
        return response.user?.userName
    }
}

suspend fun createPolicy(policyNameVal: String?): String {

    val policyDocumentValue: String = "{" +
        "  \"Version\": \"2012-10-17\", " +
        "  \"Statement\": [ " +
        "    { " +
        "      \"Effect\": \"Allow\", " +
        "      \"Action\": [ " +
        "        \"s3:*\" " +
        "      ], " +
        "      \"Resource\": \"arn:aws:s3::: ${bucketName}/*\" "
    }
}

```

```

        "      \\"Resource\\": \\"*\\" +  

        "    }" +  

        "  ]" +  

        "}" +  

    val request = CreatePolicyRequest {  

        policyName = policyNameVal  

        policyDocument = policyDocumentValue  

    }  

    IamClient { region = "AWS_GLOBAL" }.use { iamClient ->  

        val response = iamClient.createPolicy(request)  

        return response.policy?.arn.toString()  

    }  

}  

suspend fun createRole(rolenameVal: String?, fileLocation: String?): String? {  

    val jsonObject = fileLocation?.let { readJsonSimpleDemo(it) } as JSONObject  

    val request = CreateRoleRequest {  

        roleName = rolenameVal  

        assumeRolePolicyDocument = jsonObject.toJSONString()  

        description = "Created using the AWS SDK for Kotlin"  

    }  

    IamClient { region = "AWS_GLOBAL" }.use { iamClient ->  

        val response = iamClient.createRole(request)  

        return response.role?.arn  

    }  

}  

suspend fun attachRolePolicy(roleNameVal: String, policyArnVal: String) {  

    val request = ListAttachedRolePoliciesRequest {  

        roleName = roleNameVal  

    }  

    IamClient { region = "AWS_GLOBAL" }.use { iamClient ->  

        val response = iamClient.listAttachedRolePolicies(request)  

        val attachedPolicies = response.attachedPolicies  

        // Ensure that the policy is not attached to this role.  

        val checkStatus: Int  

        if (attachedPolicies != null) {  

            checkStatus = checkMyList(attachedPolicies, policyArnVal)  

            if (checkStatus == -1)  

                return
        }
  

        val policyRequest = AttachRolePolicyRequest {  

            roleName = roleNameVal  

            policyArn = policyArnVal
        }
        iamClient.attachRolePolicy(policyRequest)
        println("Successfully attached policy $policyArnVal to role $roleNameVal")
    }
}  

fun checkMyList(attachedPolicies: List<AttachedPolicy>, policyArnVal: String): Int  

{  

    for (policy in attachedPolicies) {  

        val polArn = policy.policyArn.toString()  

        if (polArn.compareTo(policyArnVal) == 0) {

```

```

        println("The policy is already attached to this role.")
        return -1
    }
}
return 0
}

suspend fun assumeGivenRole(roleArnVal: String?, roleSessionNameVal: String?,
                           bucketName: String) {

    val stsClient = StsClient {
        region = "us-east-1"
    }

    val roleRequest = AssumeRoleRequest {
        roleArn = roleArnVal
        roleSessionName = roleSessionNameVal
    }

    val roleResponse = stsClient.assumeRole(roleRequest)
    val myCreds = roleResponse.credentials
    val key = myCreds?.accessKeyId
    val secKey = myCreds?.secretAccessKey
    val secToken = myCreds?.sessionToken

    val staticCredentials = StaticCredentialsProvider {
        accessKeyId = key
        secretAccessKey = secKey
        sessionToken = secToken
    }

    // List all objects in an Amazon S3 bucket using the temp creds.
    val s3 = S3Client {
        credentialsProvider = staticCredentials
        region = "us-east-1"
    }

    println("Created a S3Client using temp credentials.")
    println("Listing objects in $bucketName")

    val listObjects = ListObjectsRequest {
        bucket = bucketName
    }

    val response = s3.listObjects(listObjects)
    response.contents?.forEach { myObject ->
        println("The name of the key is ${myObject.key}")
        println("The owner is ${myObject.owner}")
    }
}

suspend fun deleteRole(roleNameVal: String, polArn: String) {

    val iam = IamClient { region = "AWS_GLOBAL" }

    // First the policy needs to be detached.
    val rolePolicyRequest = DetachRolePolicyRequest {
        policyArn = polArn
        roleName = roleNameVal
    }

    iam.detachRolePolicy(rolePolicyRequest)

    // Delete the policy.
    val request = DeletePolicyRequest {
        policyArn = polArn
    }
}

```

```

    }

    iam.deletePolicy(request)
    println("**** Successfully deleted $polArn")

    // Delete the role.
    val roleRequest = DeleteRoleRequest {
        roleName = roleNameVal
    }

    iam.deleteRole(roleRequest)
    println("**** Successfully deleted $roleNameVal")
}

suspend fun deleteUser(userNameVal: String) {
    val iam = IamClient { region = "AWS_GLOBAL" }
    val request = DeleteUserRequest {
        userName = userNameVal
    }

    iam.deleteUser(request)
    println("**** Successfully deleted $userNameVal")
}

@Throws(java.lang.Exception::class)
fun readJsonSimpleDemo(filename: String): Any? {
    val reader = FileReader(filename)
    val jsonParser = JSONParser()
    return jsonParser.parse(reader)
}

```

- For API details, see the following topics in *AWS SDK for Kotlin API reference*.
 - [AttachRolePolicy](#)
 - [CreateAccessKey](#)
 - [CreatePolicy](#)
 - [CreateRole](#)
 - [CreateUser](#)
 - [DeleteAccessKey](#)
 - [DeletePolicy](#)
 - [DeleteRole](#)
 - [DeleteUser](#)
 - [DeleteUserPolicy](#)
 - [DetachRolePolicy](#)
 - [PutUserPolicy](#)

PHP

SDK for PHP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```

namespace Iam\Basics;

require 'vendor/autoload.php';

```

```

use Aws\Credentials\Credentials;
use Aws\S3\Exception\S3Exception;
use Aws\S3\S3Client;
use Aws\Sts\StsClient;
use Iam\IAMService;

echo("\n");
echo("-----\n");
print("Welcome to the IAM getting started demo using PHP!\n");
echo("-----\n");

$uuid = uniqid();
$service = new IAMService();

$user = $service->createUser("iam_demo_user_$uuid");
echo "Created user with the arn: {$user['Arn']}\n";

$key = $service->createAccessKey($user['UserName']);
$assumeRolePolicyDocument = "{$
    \\"Version\\": \\"2012-10-17\",
    \\"Statement\\": [
        {
            \\"Effect\\": \\"Allow\",
            \\"Principal\\": {\\\"AWS\\": \"{$user['Arn']}\"},
            \\"Action\\": \\"sts:AssumeRole\"
        }
    ]
}";
$assumeRoleRole = $service->createRole("iam_demo_role_$uuid",
    $assumeRolePolicyDocument);
echo "Created role: {$assumeRoleRole['RoleName']}\n";

$listAllBucketsPolicyDocument = "{$
    \\"Version\\": \\"2012-10-17\",
    \\"Statement\\": [
        {
            \\"Effect\\": \\"Allow\",
            \\"Action\\": \\"s3>ListAllMyBuckets\",
            \\"Resource\\": \"arn:aws:s3:::*\"
        }
    ]
}";
$listAllBucketsPolicy = $service->createPolicy("iam_demo_policy_$uuid",
    $listAllBucketsPolicyDocument);
echo "Created policy: {$listAllBucketsPolicy['PolicyName']}\n";

$service->attachRolePolicy($assumeRoleRole['RoleName'],
    $listAllBucketsPolicy['Arn']);

$inlinePolicyDocument = "{$
    \\"Version\\": \\"2012-10-17\",
    \\"Statement\\": [
        {
            \\"Effect\\": \\"Allow\",
            \\"Action\\": \\"sts:AssumeRole\",
            \\"Resource\\": \"{$assumeRoleRole['Arn']}\"\"
        }
    ]
}";
$inlinePolicy = $service->createUserPolicy("iam_demo_inline_policy_$uuid",
    $inlinePolicyDocument, $user['UserName']);
//First, fail to list the buckets with the user
$credentials = new Credentials($key['AccessKeyId'], $key['SecretAccessKey']);
$s3Client = new S3Client(['region' => 'us-west-2', 'version' => 'latest',
    'credentials' => $credentials]);
try {
    $s3Client->listBuckets([
    ]);
    echo "this should not run";
} catch (S3Exception $exception) {
    echo "successfully failed!\n";
}

```

```
$stsClient = new StsClient(['region' => 'us-west-2', 'version' => 'latest',
    'credentials' => $credentials]);
sleep(10);
$assumedRole = $stsClient->assumeRole([
    'RoleArn' => $assumeRoleRole['Arn'],
    'RoleSessionName' => "DemoAssumeRoleSession_{$uuid}",
]);
$assumedCredentials = [
    'key' => $assumedRole['Credentials']['AccessKeyId'],
    'secret' => $assumedRole['Credentials']['SecretAccessKey'],
    'token' => $assumedRole['Credentials']['SessionToken'],
];
$s3Client = new S3Client(['region' => 'us-west-2', 'version' => 'latest',
    'credentials' => $assumedCredentials]);
try {
    $s3Client->listBuckets();
    echo "this should now run!\n";
} catch (S3Exception $exception) {
    echo "this should now not fail\n";
}

$service->detachRolePolicy($assumeRoleRole['RoleName'],
    $listAllBucketsPolicy['Arn']);
$deletePolicy = $service->deletePolicy($listAllBucketsPolicy['Arn']);
echo "Delete policy: {$listAllBucketsPolicy['PolicyName']}\n";
$deletedRole = $service->deleteRole($assumeRoleRole['Arn']);
echo "Deleted role: {$assumeRoleRole['RoleName']}\n";
$deletedKey = $service->deleteAccessKey($key['AccessKeyId'], $user['UserName']);
$deletedUser = $service->deleteUser($user['UserName']);
echo "Delete user: {$user['UserName']}\n";
```

- For API details, see the following topics in *AWS SDK for PHP API Reference*.

- [AttachRolePolicy](#)
- [CreateAccessKey](#)
- [CreatePolicy](#)
- [CreateRole](#)
- [CreateUser](#)
- [DeleteAccessKey](#)
- [DeletePolicy](#)
- [DeleteRole](#)
- [DeleteUser](#)
- [DeleteUserPolicy](#)
- [DetachRolePolicy](#)
- [PutUserPolicy](#)

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Create an IAM user and a role that grants permission to list Amazon S3 buckets. The user has rights only to assume the role. After assuming the role, use temporary credentials to list buckets for the account.

```

import json
import sys
import time
from uuid import uuid4

import boto3
from botocore.exceptions import ClientError


def progress_bar(seconds):
    """Shows a simple progress bar in the command window."""
    for _ in range(seconds):
        time.sleep(1)
        print('.', end='')
        sys.stdout.flush()
    print()


def setup(iam_resource):
    """
    Creates a new user with no permissions.
    Creates an access key pair for the user.
    Creates a role with a policy that lets the user assume the role.
    Creates a policy that allows listing Amazon S3 buckets.
    Attaches the policy to the role.
    Creates an inline policy for the user that lets the user assume the role.

    :param iam_resource: A Boto3 AWS Identity and Access Management (IAM) resource
                         that has permissions to create users, roles, and policies
                         in the account.
    :return: The newly created user, user key, and role.
    """
    try:
        user = iam_resource.create_user(UserName=f'demo-user-{uuid4()}')
        print(f"Created user {user.name}.")
    except ClientError as error:
        print(f"Couldn't create a user for the demo. Here's why: "
              f"{error.response['Error']['Message']}")
        raise

    try:
        user_key = user.create_access_key_pair()
        print(f"Created access key pair for user {user.name}.")
    except ClientError as error:
        print(f"Couldn't create access keys for user {user.name}. Here's why: "
              f"{error.response['Error']['Message']}")
        raise

    print(f"Wait for user to be ready.", end='')
    progress_bar(10)

    try:
        role = iam_resource.create_role(
            RoleName=f'demo-role-{uuid4()}',
            AssumeRolePolicyDocument=json.dumps({
                'Version': '2012-10-17',
                'Statement': [
                    {
                        'Effect': 'Allow',
                        'Principal': {'AWS': user.arn},
                        'Action': 'sts:AssumeRole'
                    }
                ]
            })
        print(f"Created role {role.name}.")
    except ClientError as error:
        print(f"Couldn't create a role for the demo. Here's why: "
              f"{error.response['Error']['Message']}")
        raise

```

```

try:
    policy = iam_resource.create_policy(
        PolicyName=f'demo-policy-{uuid4()}',
        PolicyDocument=json.dumps({
            'Version': '2012-10-17',
            'Statement': [{
                'Effect': 'Allow',
                'Action': 's3>ListAllMyBuckets',
                'Resource': 'arn:aws:s3:::*'}]}))
    role.attach_policy(PolicyArn=policy.arn)
    print(f"Created policy {policy.policy_name} and attached it to the role.")
except ClientError as error:
    print(f"Couldn't create a policy and attach it to role {role.name}. Here's why: "
          f"{error.response['Error']['Message']}")
    raise

try:
    user.create_policy(
        PolicyName=f'demo-user-policy-{uuid4()}',
        PolicyDocument=json.dumps({
            'Version': '2012-10-17',
            'Statement': [{
                'Effect': 'Allow',
                'Action': 'sts:AssumeRole',
                'Resource': role.arn}]}))
    print(f"Created an inline policy for {user.name} that lets the user assume "
          f"the role.")
except ClientError as error:
    print(f"Couldn't create an inline policy for user {user.name}. Here's why: "
          f"{error.response['Error']['Message']}")
    raise

print("Give AWS time to propagate these new resources and connections.", end='')
progress_bar(10)

return user, user_key, role

def show_access_denied_without_role(user_key):
    """
    Shows that listing buckets without first assuming the role is not allowed.

    :param user_key: The key of the user created during setup. This user does not
                     have permission to list buckets in the account.
    """
    print(f"Try to list buckets without first assuming the role.")
    s3_denied_resource = boto3.resource(
        's3', aws_access_key_id=user_key.id, aws_secret_access_key=user_key.secret)
    try:
        for bucket in s3_denied_resource.buckets.all():
            print(bucket.name)
        raise RuntimeError("Expected to get AccessDenied error when listing buckets!")
    except ClientError as error:
        if error.response['Error']['Code'] == 'AccessDenied':
            print("Attempt to list buckets with no permissions: AccessDenied.")
        else:
            raise

def list_buckets_from_assumed_role(user_key, assume_role_arn, session_name):

```

```

"""
Assumes a role that grants permission to list the Amazon S3 buckets in the
account.
Uses the temporary credentials from the role to list the buckets that are owned
by the assumed role's account.

:param user_key: The access key of a user that has permission to assume the
role.
:param assume_role_arn: The Amazon Resource Name (ARN) of the role that
                        grants access to list the other account's buckets.
:param session_name: The name of the STS session.
"""

sts_client = boto3.client(
    'sts', aws_access_key_id=user_key.id,
    aws_secret_access_key=user_key.secret)
try:
    response = sts_clientassume_role(
        RoleArn=assume_role_arn, RoleSessionName=session_name)
    temp_credentials = response['Credentials']
    print(f"Assumed role {assume_role_arn} and got temporary credentials.")
except ClientError as error:
    print(f"Couldn't assume role {assume_role_arn}. Here's why: "
          f"{error.response['Error']['Message']}")
    raise

# Create an S3 resource that can access the account with the temporary
# credentials.
s3_resource = boto3.resource(
    's3',
    aws_access_key_id=temp_credentials['AccessKeyId'],
    aws_secret_access_key=temp_credentials['SecretAccessKey'],
    aws_session_token=temp_credentials['SessionToken'])
print(f"Listing buckets for the assumed role's account:")
try:
    for bucket in s3_resource.buckets.all():
        print(bucket.name)
except ClientError as error:
    print(f"Couldn't list buckets for the account. Here's why: "
          f"{error.response['Error']['Message']}")
    raise

def teardown(user, role):
    """
    Removes all resources created during setup.

    :param user: The demo user.
    :param role: The demo role.
    """

    try:
        for attached in role.attached_policies.all():
            policy_name = attached.policy_name
            role.detach_policy(PolicyArn=attached.arn)
            attached.delete()
            print(f"Detached and deleted {policy_name}.")
        role.delete()
        print(f"Deleted {role.name}.")
    except ClientError as error:
        print("Couldn't detach policy, delete policy, or delete role. Here's why: "
              f"{error.response['Error']['Message']}")
        raise

    try:
        for user_pol in user.policies.all():
            user_pol.delete()
            print("Deleted inline user policy.")
    
```

```
for key in user.access_keys.all():
    key.delete()
    print("Deleted user's access key.")
user.delete()
print(f"Deleted {user.name}.")
except ClientError as error:
    print("Couldn't delete user policy or delete user. Here's why: "
          f"{error.response['Error']['Message']}")

def usage_demo():
    """Drives the demonstration."""
    print('-'*88)
    print(f"Welcome to the IAM create user and assume role demo.")
    print('-'*88)
    iam_resource = boto3.resource('iam')
    user = None
    role = None
    try:
        user, user_key, role = setup(iam_resource)
        print(f"Created {user.name} and {role.name}.")
        show_access_denied_without_role(user_key)
        list_buckets_from_assumed_role(user_key, role.arn, 'AssumeRoleDemoSession')
    except Exception:
        print("Something went wrong!")
    finally:
        if user is not None and role is not None:
            teardown(user, role)
        print("Thanks for watching!")

if __name__ == '__main__':
    usage_demo()
```

- For API details, see the following topics in *AWS SDK for Python (Boto3) API Reference*.
 - [AttachRolePolicy](#)
 - [CreateAccessKey](#)
 - [CreatePolicy](#)
 - [CreateRole](#)
 - [CreateUser](#)
 - [DeleteAccessKey](#)
 - [DeletePolicy](#)
 - [DeleteRole](#)
 - [DeleteUser](#)
 - [DeleteUserPolicy](#)
 - [DetachRolePolicy](#)
 - [PutUserPolicy](#)

Ruby

SDK for Ruby

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
require "aws-sdk-iam"
```

```

require "aws-sdk-s3"

# Wraps the scenario actions.
class ScenarioCreateUserAssumeRole
  attr_reader :iam_resource

  # @param iam_resource [Aws::IAM::Resource] An AWS IAM resource.
  def initialize(iam_resource)
    @iam_resource = iam_resource
  end

  # Waits for the specified number of seconds.
  #
  # @param duration [Integer] The number of seconds to wait.
  def wait(duration)
    puts("Give AWS time to propagate resources...")
    sleep(duration)
  end

  # Creates a user.
  #
  # @param user_name [String] The name to give the user.
  # @return [Aws::IAM::User] The newly created user.
  def create_user(user_name)
    user = @iam_resource.create_user(user_name: user_name)
    puts("Created demo user named #{user.name}.")
    rescue Aws::Errors::ServiceError => e
      puts("Tried and failed to create demo user.")
      puts("t#{e.code}: #{e.message}")
      puts("\nCan't continue the demo without a user!")
      raise
    else
      user
    end

    # Creates an access key for a user.
    #
    # @param user [Aws::IAM::User] The user that owns the key.
    # @return [Aws::IAM::AccessKeyPair] The newly created access key.
    def create_access_key_pair(user)
      user_key = user.create_access_key_pair
      puts("Created access key pair for user.")
      rescue Aws::Errors::ServiceError => e
        puts("Couldn't create access keys for user #{user.name}.")
        puts("t#{e.code}: #{e.message}")
        raise
      else
        user_key
      end

      # Creates a role that can be assumed by a user.
      #
      # @param role_name [String] The name to give the role.
      # @param user [Aws::IAM::User] The user who is granted permission to assume the
      # role.
      # @return [Aws::IAM::Role] The newly created role.
      def create_role(role_name, user)
        role = @iam_resource.create_role(
          role_name: role_name,
          assume_role_policy_document: {
            Version: "2012-10-17",
            Statement: [
              {
                Effect: "Allow",
                Principal: {'AWS': user.arn},
                Action: "sts:AssumeRole"
              }
            ]
          }
        )
      end
    end
  end
end

```

```

        }.to_json)
    puts("Created role #{role.name}.")
rescue Aws::Errors::ServiceError => e
    puts("Couldn't create a role for the demo. Here's why: ")
    puts("\t#{e.code}: #{e.message}")
    raise
else
    role
end

# Creates a policy that grants permission to list S3 buckets in the account, and
# then attaches the policy to a role.
#
# @param policy_name [String] The name to give the policy.
# @param role [Aws::IAM::Role] The role that the policy is attached to.
# @return [Aws::IAM::Policy] The newly created policy.
def create_and_attach_role_policy(policy_name, role)
    policy = @iam_resource.create_policy(
        policy_name: policy_name,
        policy_document: {
            Version: "2012-10-17",
            Statement: [
                {
                    Effect: "Allow",
                    Action: "s3>ListAllMyBuckets",
                    Resource: "arn:aws:s3::*"
                }
            ].to_json)
    role.attach_policy(policy_arn: policy.arn)
    puts("Created policy #{policy.policy_name} and attached it to role
#{role.name}.")
    rescue Aws::Errors::ServiceError => e
        puts("Couldn't create a policy and attach it to role #{role.name}. Here's why:
")
        puts("\t#{e.code}: #{e.message}")
        raise
    else
        policy
    end

# Creates an inline policy for a user that lets the user assume a role.
#
# @param policy_name [String] The name to give the policy.
# @param user [Aws::IAM::User] The user that owns the policy.
# @param role [Aws::IAM::Role] The role that can be assumed.
# @return [Aws::IAM::UserPolicy] The newly created policy.
def create_user_policy(policy_name, user, role)
    policy = user.create_policy(
        policy_name: policy_name,
        policy_document: {
            Version: "2012-10-17",
            Statement: [
                {
                    Effect: "Allow",
                    Action: "sts:AssumeRole",
                    Resource: role.arn
                }
            ].to_json)
    puts("Created an inline policy for #{user.name} that lets the user assume role
#{role.name}.")
    rescue Aws::Errors::ServiceError => e
        puts("Couldn't create an inline policy for user #{user.name}. Here's why: ")
        puts("\t#{e.code}: #{e.message}")
        raise
    else
        policy
    end

```

```

# Creates an Amazon S3 resource with specified credentials. This is separated
into a
# factory function so that it can be mocked for unit testing.
#
# @param credentials [Aws::Credentials] The credentials used by the Amazon S3
resource.
def create_s3_resource(credentials)
  Aws::S3::Resource.new(client: Aws::S3::Client.new(credentials: credentials))
end

# Lists the S3 buckets for the account, using the specified Amazon S3 resource.
# Because the resource uses credentials with limited access, it may not be able
to
# list the S3 buckets.
#
# @param s3_resource [Aws::S3::Resource] An Amazon S3 resource.
def list_buckets(s3_resource)
  count = 10
  s3_resource.buckets.each do |bucket|
    puts "\t#{bucket.name}"
    count -= 1
    break if count.zero?
  end
rescue Aws::Errors::ServiceError => e
  if e.code == "AccessDenied"
    puts("Attempt to list buckets with no permissions: AccessDenied.")
  else
    puts("Couldn't list buckets for the account. Here's why: ")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
end

# Creates an AWS Security Token Service (AWS STS) client with specified
credentials.
# This is separated into a factory function so that it can be mocked for unit
testing.
#
# @param key_id [String] The ID of the access key used by the STS client.
# @param key_secret [String] The secret part of the access key used by the STS
client.
def create_sts_client(key_id, key_secret)
  Aws::STS::Client.new(access_key_id: key_id, secret_access_key: key_secret)
end

# Gets temporary credentials that can be used to assume a role.
#
# @param role_arn [String] The ARN of the role that is assumed when these
credentials
#           are used.
# @param sts_client [AWS::STS::Client] An AWS STS client.
# @return [Aws::AssumeRoleCredentials] The credentials that can be used to assume
the role.
def assume_role(role_arn, sts_client)
  credentials = Aws::AssumeRoleCredentials.new(
    client: sts_client,
    role_arn: role_arn,
    role_session_name: "create-use-assume-role-scenario"
  )
  puts("Assumed role '#{role_arn}', got temporary credentials.")
  credentials
end

# Deletes a role. If the role has policies attached, they are detached and
# deleted before the role is deleted.
#

```

```

# @param role [Aws::IAM::Role] The role to delete.
def delete_role(role)
  role.attached_policies.each do |policy|
    name = policy.policy_name
    policy.detach_role(role_name: role.name)
    policy.delete
    puts("Deleted policy #{name}.")
  end
  name = role.name
  role.delete
  puts("Deleted role #{name}.")
rescue Aws::Errors::ServiceError => e
  puts("Couldn't detach policies and delete role #{role.name}. Here's why:")
  puts("\t#{e.code}: #{e.message}")
  raise
end

# Deletes a user. If the user has inline policies or access keys, they are
# deleted
# before the user is deleted.
#
# @param user [Aws::IAM::User] The user to delete.
def delete_user(user)
  user.policies.each do |policy|
    name = policy.name
    policy.delete
    puts("Deleted user policy #{name}.")
  end
  user.access_keys.each do |key|
    key.delete
    puts("Deleted access key for user #{user.name}.")
  end
  name = user.name
  user.delete
  puts("Deleted user #{name}.")
rescue Aws::Errors::ServiceError => e
  puts("Couldn't detach policies and delete user #{user.name}. Here's why:")
  puts("\t#{e.code}: #{e.message}")
end
end

# Runs the IAM create a user and assume a role scenario.
def run_scenario(scenario)
  puts("-" * 88)
  puts("Welcome to the IAM create a user and assume a role demo!")
  puts("-" * 88)

  user = scenario.create_user("doc-example-user-#{Random.uuid}")
  user_key = scenario.create_access_key_pair(user)
  scenario.wait(10)
  role = scenario.create_role("doc-example-role-#{Random.uuid}", user)
  scenario.create_and_attach_role_policy("doc-example-role-policy-#{Random.uuid}", role)
  scenario.create_user_policy("doc-example-user-policy-#{Random.uuid}", user, role)
  scenario.wait(10)
  puts("Try to list buckets with credentials for a user who has no permissions.")
  puts("Expect AccessDenied from this call.")
  scenario.list_buckets(
    scenario.create_s3_resource(Aws::Credentials.new(user_key.id,
    user_key.secret)))
  puts("Now, assume the role that grants permission.")
  temp_credentials = scenario.assume_role(
    role.arn, scenario.create_sts_client(user_key.id, user_key.secret))
  puts("Here are your buckets:")
  scenario.list_buckets(scenario.create_s3_resource(temp_credentials))
  puts("Deleting role '#{role.name}' and attached policies.")
end

```

```

scenario.delete_role(role)
puts("Deleting user '#{user.name}', policies, and keys.")
scenario.delete_user(user)

puts("Thanks for watching!")
puts("-" * 88)
rescue Aws::Errors::ServiceError => e
  puts("Something went wrong with the demo.")
  puts("\t#{e.code}: #{e.message}")
end

run_scenario(ScenarioCreateUserAssumeRole.new(Aws::IAM::Resource.new)) if
$PROGRAM_NAME == __FILE__

```

- For API details, see the following topics in *AWS SDK for Ruby API Reference*.
 - [AttachRolePolicy](#)
 - [CreateAccessKey](#)
 - [CreatePolicy](#)
 - [CreateRole](#)
 - [CreateUser](#)
 - [DeleteAccessKey](#)
 - [DeletePolicy](#)
 - [DeleteRole](#)
 - [DeleteUser](#)
 - [DeleteUserPolicy](#)
 - [DetachRolePolicy](#)
 - [PutUserPolicy](#)

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```

use aws_config::meta::region::RegionProviderChain;
use aws_sdk_iam::Error as iamError;
use aws_sdk_iam::{config::Credentials as iamCredentials, config::Region, Client as iamClient};
use aws_sdk_s3::Client as s3Client;
use aws_sdk_sts::Client as stsClient;
use std::borrow::Borrow;
use tokio::time::{sleep, Duration};
use uuid::Uuid;

#[tokio::main]
async fn main() -> Result<(), iamError> {
    let (client, uuid, list_all_buckets_policy_document, inline_policy_document) =
        initialize_variables().await;

```

```

        if let Err(e) = run_iam_operations(
            client,
            uuid,
            list_all_buckets_policy_document,
            inline_policy_document,
        )
        .await
    {
        println!("{}:?", e);
    };

    Ok(())
}

async fn initialize_variables() -> (iamClient, String, String, String) {
    let region_provider = RegionProviderChain::first_try(Region::new("us-west-2"));

    let shared_config =
        aws_config::from_env().region(region_provider).load().await;
    let client = iamClient::new(&shared_config);
    let uuid = Uuid::new_v4().to_string();

    let list_all_buckets_policy_document = "{\n        \"Version\": \"2012-10-17\",\n        \"Statement\": [{\n            \"Effect\": \"Allow\",\n            \"Action\": \"s3>ListAllMyBuckets\",\n            \"Resource\": \"arn:aws:s3:::*\"\n        }]\n    }.to_string();
    let inline_policy_document = "{\n        \"Version\": \"2012-10-17\",\n        \"Statement\": [{\n            \"Effect\": \"Allow\",\n            \"Action\": \"sts:AssumeRole\",\n            \"Resource\": \"{}\"\n        }]\n    }.to_string();

    (
        client,
        uuid,
        list_all_buckets_policy_document,
        inline_policy_document,
    )
}

async fn run_iam_operations(
    client: iamClient,
    uuid: String,
    list_all_buckets_policy_document: String,
    inline_policy_document: String,
) -> Result<(), iamError> {
    let user = iam_service::create_user(&client, &format!("{}{}", "iam_demo_user_", uuid)).await?;
    println!(
        "Created the user with the name: {}",
        user.user_name.as_ref().unwrap()
    );
    let key = iam_service::create_access_key(&client,
    user.user_name.as_ref().unwrap()).await?;

    let assume_role_policy_document = "{\n        \"Version\": \"2012-10-17\",\n        \"Statement\": [{\n            \"Effect\": \"Allow\"}

```

```

        \"Principal\": {\"AWS\": \"{}\"},
        \"Action\": \"sts:AssumeRole\"
    }]
}
.to_string()
.replace("{}", user.arn.as_ref().unwrap());

let assume_role_role = iam_service::create_role(
    &client,
    &format!("{}{}", "iam_demo_role_", uuid),
    &assume_role_policy_document,
)
.await?;
println!(
    "Created the role with the ARN: {}",
    assume_role_role.arn.as_ref().unwrap()
);

let list_all_buckets_policy = iam_service::create_policy(
    &client,
    &format!("{}{}", "iam_demo_policy_", uuid),
    &list_all_buckets_policy_document,
)
.await?;
println!(
    "Created policy: {}",
    list_all_buckets_policy.policy_name.as_ref().unwrap()
);

let attach_role_policy_result =
    iam_service::attach_role_policy(&client, &assume_role_role,
&list_all_buckets_policy)
    .await?;
println!(
    "Attached the policy to the role: {:?}", attach_role_policy_result
);

let inline_policy_name = format!("{}{}", "iam_demo_inline_policy_", uuid);
let inline_policy_document =
    inline_policy_document.replace("{}",
assume_role_role.arn.as_ref().unwrap());
    iam_service::create_user_policy(&client, &user, &inline_policy_name,
&inline_policy_document)
    .await?;
println!("Created inline policy.");

//First, fail to list the buckets with the user.
let creds = iamCredentials::from_keys(
    key.access_key_id.as_ref().unwrap(),
    key.secret_access_key.as_ref().unwrap(),
    None,
);
let fail_config = aws_config::from_env()
    .credentials_provider(creds.clone())
    .load()
    .await;
println!("Fail config: {:?}", fail_config);
let fail_client: s3Client = s3Client::new(&fail_config);
match fail_client.list_buckets().send().await {
    Ok(e) => {
        println!("This should not run. {:?}", e);
    }
    Err(e) => {
        println!("Successfully failed with error: {:?}", e)
    }
}

```

```

}

let sts_config = aws_config::from_env()
    .credentials_provider(creds.clone())
    .load()
    .await;
let sts_client: stsClient = stsClient::new(&sts_config);
sleep(Duration::from_secs(10)).await;
let assumed_role = sts_client
    .assume_role()
    .role_arn(assume_role.role.arn.as_ref().unwrap())
    .role_session_name(&format!("{}{}", "iam_demo_assumerole_session_", uuid))
    .send()
    .await;
println!("Assumed role: {:?}", assumed_role);
sleep(Duration::from_secs(10)).await;

let assumed_credentials = iamCredentials::from_keys(
    assumed_role
        .as_ref()
        .unwrap()
        .credentials
        .as_ref()
        .unwrap()
        .access_key_id
        .as_ref()
        .unwrap(),
    assumed_role
        .as_ref()
        .unwrap()
        .credentials
        .as_ref()
        .unwrap()
        .secret_access_key
        .as_ref()
        .unwrap(),
    assumed_role
        .as_ref()
        .unwrap()
        .credentials
        .as_ref()
        .unwrap()
        .session_token
        .borrow()
        .clone(),
);
;

let succeed_config = aws_config::from_env()
    .credentials_provider(assumed_credentials)
    .load()
    .await;
println!("succeed config: {:?}", succeed_config);
let succeed_client: s3Client = s3Client::new(&succeed_config);
sleep(Duration::from_secs(10)).await;
match succeed_client.list_buckets().send().await {
    Ok(_) => {
        println!("This should now run successfully.")
    }
    Err(e) => {
        println!("This should not run. {:?}", e);
        panic!()
    }
}
;

//Clean up.
iam_service::detach_role_policy(

```

```
    &client,
    assume_role_role.role_name.as_ref().unwrap(),
    list_all_buckets_policy.arn.as_ref().unwrap(),
)
.await?;
iam_service::delete_policy(&client, list_all_buckets_policy).await?;
iam_service::delete_role(&client, &assume_role_role).await?;
println!(
    "Deleted role {}",
    assume_role_role.role_name.as_ref().unwrap()
);
iam_service::delete_access_key(&client, &user, &key).await?;
println!("Deleted key for {}", key.user_name.as_ref().unwrap());
iam_service::delete_user_policy(&client, &user, &inline_policy_name).await?;
println!("Deleted inline user policy: {}", inline_policy_name);
iam_service::delete_user(&client, &user).await?;
println!("Deleted user {}", user.user_name.as_ref().unwrap());

Ok(())
}
```

- For API details, see the following topics in *AWS SDK for Rust API reference*.

- [AttachRolePolicy](#)
- [CreateAccessKey](#)
- [CreatePolicy](#)
- [CreateRole](#)
- [CreateUser](#)
- [DeleteAccessKey](#)
- [DeletePolicy](#)
- [DeleteRole](#)
- [DeleteUser](#)
- [DeleteUserPolicy](#)
- [DetachRolePolicy](#)
- [PutUserPolicy](#)

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Create read-only and read-write IAM users using an AWS SDK

The following code example shows how to create users and attach policies to them.

Warning

To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as [AWS IAM Identity Center \(successor to AWS Single Sign-On\)](#).

- Create two IAM users.
- Attach a policy for one user to get and put objects in an Amazon S3 bucket.
- Attach a policy for the second user to get objects from the bucket.
- Get different permissions to the bucket based on user credentials.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Create functions that wrap IAM user actions.

```
import logging
import time

import boto3
from botocore.exceptions import ClientError

import access_key_wrapper
import policy_wrapper

logger = logging.getLogger(__name__)
iam = boto3.resource('iam')

def create_user(user_name):
    """
    Creates a user. By default, a user has no permissions or access keys.

    :param user_name: The name of the user.
    :return: The newly created user.
    """
    try:
        user = iam.create_user(UserName=user_name)
        logger.info("Created user %s.", user.name)
    except ClientError:
        logger.exception("Couldn't create user %s.", user_name)
        raise
    else:
        return user

def update_user(user_name, new_user_name):
    """
    Updates a user's name.

    :param user_name: The current name of the user to update.
    :param new_user_name: The new name to assign to the user.
    :return: The updated user.
    """
    try:
        user = iam.User(user_name)
        user.update(NewUserName=new_user_name)
        logger.info("Renamed %s to %s.", user_name, new_user_name)
    except ClientError:
        logger.exception("Couldn't update name for user %s.", user_name)
        raise
    return user

def list_users():
    """
    Lists the users in the current account.

    :return: The list of users.
    """
    try:
        users = list(iam.users.all())
        logger.info("Got %s users.", len(users))
    except ClientError:
```

```

        logger.exception("Couldn't get users.")
        raise
    else:
        return users

def delete_user(user_name):
    """
    Deletes a user. Before a user can be deleted, all associated resources,
    such as access keys and policies, must be deleted or detached.

    :param user_name: The name of the user.
    """
    try:
        iam.User(user_name).delete()
        logger.info("Deleted user %s.", user_name)
    except ClientError:
        logger.exception("Couldn't delete user %s.", user_name)
        raise

def attach_policy(user_name, policy_arn):
    """
    Attaches a policy to a user.

    :param user_name: The name of the user.
    :param policy_arn: The Amazon Resource Name (ARN) of the policy.
    """
    try:
        iam.User(user_name).attach_policy(PolicyArn=policy_arn)
        logger.info("Attached policy %s to user %s.", policy_arn, user_name)
    except ClientError:
        logger.exception("Couldn't attach policy %s to user %s.", policy_arn,
                         user_name)
        raise

def detach_policy(user_name, policy_arn):
    """
    Detaches a policy from a user.

    :param user_name: The name of the user.
    :param policy_arn: The Amazon Resource Name (ARN) of the policy.
    """
    try:
        iam.User(user_name).detach_policy(PolicyArn=policy_arn)
        logger.info("Detached policy %s from user %s.", policy_arn, user_name)
    except ClientError:
        logger.exception(
            "Couldn't detach policy %s from user %s.", policy_arn, user_name)
        raise

```

Create functions that wrap IAM policy actions.

```

import json
import logging
import operator
import pprint
import time

import boto3
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)
iam = boto3.resource('iam')

def create_policy(name, description, actions, resource_arn):

```

```

"""
Creates a policy that contains a single statement.

:param name: The name of the policy to create.
:param description: The description of the policy.
:param actions: The actions allowed by the policy. These typically take the
    form of service:action, such as s3:PutObject.
:param resource_arn: The Amazon Resource Name (ARN) of the resource this policy
    applies to. This ARN can contain wildcards, such as
    'arn:aws:s3:::my-bucket/*' to allow actions on all objects
    in the bucket named 'my-bucket'.
:return: The newly created policy.
"""

policy_doc = {
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": actions,
            "Resource": resource_arn
        }
    ]
}
try:
    policy = iam.create_policy(
        PolicyName=name, Description=description,
        PolicyDocument=json.dumps(policy_doc))
    logger.info("Created policy %s.", policy.arn)
except ClientError:
    logger.exception("Couldn't create policy %s.", name)
    raise
else:
    return policy

def delete_policy(policy_arn):
    """
    Deletes a policy.

    :param policy_arn: The ARN of the policy to delete.
    """
    try:
        iam.Policy(policy_arn).delete()
        logger.info("Deleted policy %s.", policy_arn)
    except ClientError:
        logger.exception("Couldn't delete policy %s.", policy_arn)
        raise

```

Create functions that wrap IAM access key actions.

```

import logging
import boto3
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)

iam = boto3.resource('iam')

def create_key(user_name):
    """
    Creates an access key for the specified user. Each user can have a
    maximum of two keys.

    :param user_name: The name of the user.
    :return: The created access key.
    """

```

```

"""
try:
    key_pair = iam.User(user_name).create_access_key_pair()
    logger.info(
        "Created access key pair for %s. Key ID is %s.",
        key_pair.user_name, key_pair.id)
except ClientError:
    logger.exception("Couldn't create access key pair for %s.", user_name)
    raise
else:
    return key_pair

def delete_key(user_name, key_id):
    """
    Deletes a user's access key.

    :param user_name: The user that owns the key.
    :param key_id: The ID of the key to delete.
    """

    try:
        key = iam.AccessKey(user_name, key_id)
        key.delete()
        logger.info(
            "Deleted access key %s for %s.", key.id, key.user_name)
    except ClientError:
        logger.exception("Couldn't delete key %s for %s", key_id, user_name)
        raise

```

Use the wrapper functions to create users with differing policies and use their credentials to access an Amazon S3 bucket.

```

def usage_demo():
    """
    Shows how to manage users, keys, and policies.
    This demonstration creates two users: one user who can put and get objects in
    an
    Amazon S3 bucket, and another user who can only get objects from the bucket.
    The demo then shows how the users can perform only the actions they are
    permitted
    to perform.
    """
    logging.basicConfig(level=logging.INFO, format='%(levelname)s: %(message)s')
    print('*'*88)
    print("Welcome to the AWS Identity and Account Management user demo.")
    print('*'*88)
    print("Users can have policies and roles attached to grant them specific "
          "permissions.")
    s3 = boto3.resource('s3')
    bucket = s3.create_bucket(
        Bucket=f'demo-iam-bucket-{time.time_ns()}',
        CreateBucketConfiguration={
            'LocationConstraint': s3.meta.client.meta.region_name
        }
    )
    print(f"Created an Amazon S3 bucket named {bucket.name}.")
    user_read_writer = create_user('demo-iam-read-writer')
    user_reader = create_user('demo-iam-reader')
    print(f"Created two IAM users: {user_read_writer.name} and {user_reader.name}")
    update_user(user_read_writer.name, 'demo-iam-creator')
    update_user(user_reader.name, 'demo-iam-getter')
    users = list_users()

```

```

user_read_writer = next(user for user in users if user.user_id ==
user_read_writer.user_id)
user_reader = next(user for user in users if user.user_id ==
user_reader.user_id)
print(f"Changed the names of the users to {user_read_writer.name} "
      f"and {user_reader.name}.")

read_write_policy = policy_wrapper.create_policy(
    'demo-iam-read-write-policy',
    'Grants rights to create and get an object in the demo bucket.',
    ['s3:PutObject', 's3:GetObject'],
    f'arn:aws:s3:::{bucket.name}/*')
print(f"Created policy {read_write_policy.policy_name} with ARN:
{read_write_policy.arn}")
print(read_write_policy.description)
read_policy = policy_wrapper.create_policy(
    'demo-iam-read-policy',
    'Grants rights to get an object from the demo bucket.',
    's3:GetObject',
    f'arn:aws:s3:::{bucket.name}/*')
print(f"Created policy {read_policy.policy_name} with ARN: {read_policy.arn}")
print(read_policy.description)
attach_policy(user_read_writer.name, read_write_policy.arn)
print(f"Attached {read_write_policy.policy_name} to {user_read_writer.name}.")
attach_policy(user_reader.name, read_policy.arn)
print(f"Attached {read_policy.policy_name} to {user_reader.name}.")

user_read_writer_key = access_key_wrapper.create_key(user_read_writer.name)
print(f"Created access key pair for {user_read_writer.name}.")
user_reader_key = access_key_wrapper.create_key(user_reader.name)
print(f"Created access key pair for {user_reader.name}.")

s3_read_writer_resource = boto3.resource(
    's3',
    aws_access_key_id=user_read_writer_key.id,
    aws_secret_access_key=user_read_writer_key.secret)
demo_object_key = f'object-{time.time_ns()}''
demo_object = None
while demo_object is None:
    try:
        demo_object = s3_read_writer_resource.Bucket(bucket.name).put_object(
            Key=demo_object_key, Body=b'AWS IAM demo object content!')
    except ClientError as error:
        if error.response['Error']['Code'] == 'InvalidAccessKeyId':
            print("Access key not yet available. Waiting...")
            time.sleep(1)
        else:
            raise
print(f"Put {demo_object_key} into {bucket.name} using "
      f"{user_read_writer.name}'s credentials.")

read_writer_object = s3_read_writer_resource.Bucket(
    bucket.name).Object(demo_object_key)
read_writer_content = read_writer_object.get()['Body'].read()
print(f"Got object {read_writer_object.key} using read-writer user's
credentials.")
print(f"Object content: {read_writer_content}")

s3_reader_resource = boto3.resource(
    's3',
    aws_access_key_id=user_reader_key.id,
    aws_secret_access_key=user_reader_key.secret)
demo_content = None
while demo_content is None:
    try:

```

```

demo_object =
s3_reader_resource.Bucket(bucket.name).Object(demo_object_key)
demo_content = demo_object.get()['Body'].read()
print(f"Got object {demo_object.key} using reader user's credentials.")
print(f"Object content: {demo_content}")
except ClientError as error:
    if error.response['Error']['Code'] == 'InvalidAccessKeyId':
        print("Access key not yet available. Waiting...")
        time.sleep(1)
    else:
        raise

try:
    demo_object.delete()
except ClientError as error:
    if error.response['Error']['Code'] == 'AccessDenied':
        print('*'*88)
        print("Tried to delete the object using the reader user's credentials.
"
        "Got expected AccessDenied error because the reader is not "
        "allowed to delete objects.")
        print('*'*88)

access_key_wrapper.delete_key(user_reader.name, user_reader_key.id)
detach_policy(user_reader.name, read_policy.arn)
policy_wrapper.delete_policy(read_policy.arn)
delete_user(user_reader.name)
print(f"Deleted keys, detached and deleted policy, and deleted
{user_reader.name}.")

access_key_wrapper.delete_key(user_read_writer.name, user_read_writer_key.id)
detach_policy(user_read_writer.name, read_write_policy.arn)
policy_wrapper.delete_policy(read_write_policy.arn)
delete_user(user_read_writer.name)
print(f"Deleted keys, detached and deleted policy, and deleted
{user_read_writer.name}.")

bucket.objects.delete()
bucket.delete()
print(f"Eemptied and deleted {bucket.name}.")
print("Thanks for watching!")

```

- For API details, see the following topics in *AWS SDK for Python (Boto3) API Reference*.
 - [AttachUserPolicy](#)
 - [CreateAccessKey](#)
 - [CreatePolicy](#)
 - [CreateUser](#)
 - [DeleteAccessKey](#)
 - [DeletePolicy](#)
 - [DeleteUser](#)
 - [DetachUserPolicy](#)
 - [ListUsers](#)
 - [UpdateUser](#)

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Manage IAM access keys using an AWS SDK

The following code example shows how to manage access keys.

Warning

To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as [AWS IAM Identity Center \(successor to AWS Single Sign-On\)](#).

- Create and list access keys.
- Find out when and how an access key was last used.
- Update and delete access keys.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Create functions that wrap IAM access key actions.

```
import logging
import boto3
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)

iam = boto3.resource('iam')

def list_keys(user_name):
    """
    Lists the keys owned by the specified user.

    :param user_name: The name of the user.
    :return: The list of keys owned by the user.
    """
    try:
        keys = list(iam.User(user_name).access_keys.all())
        logger.info("Got %s access keys for %s.", len(keys), user_name)
    except ClientError:
        logger.exception("Couldn't get access keys for %s.", user_name)
        raise
    else:
        return keys

def create_key(user_name):
    """
    Creates an access key for the specified user. Each user can have a
    maximum of two keys.

    :param user_name: The name of the user.
    :return: The created access key.
    """
    try:
        key_pair = iam.User(user_name).create_access_key_pair()
        logger.info(
            "Created access key pair for %s. Key ID is %s.",
            key_pair.user_name, key_pair.id)
    except ClientError:
        logger.exception("Couldn't create access key pair for %s.", user_name)
```

```

        raise
    else:
        return key_pair

def get_last_use(key_id):
    """
    Gets information about when and how a key was last used.

    :param key_id: The ID of the key to look up.
    :return: Information about the key's last use.
    """

    try:
        response = iam.meta.client.get_access_key_last_used(AccessKeyId=key_id)
        last_used_date = response['AccessKeyLastUsed'].get('LastUsedDate', None)
        last_service = response['AccessKeyLastUsed'].get('ServiceName', None)
        logger.info(
            "Key %s was last used by %s on %s to access %s.", key_id,
            response['UserName'], last_used_date, last_service)
    except ClientError:
        logger.exception("Couldn't get last use of key %s.", key_id)
        raise
    else:
        return response

def update_key(user_name, key_id, activate):
    """
    Updates the status of a key.

    :param user_name: The user that owns the key.
    :param key_id: The ID of the key to update.
    :param activate: When True, the key is activated. Otherwise, the key is
    deactivated.
    """

    try:
        key = iam.User(user_name).AccessKey(key_id)
        if activate:
            key.activate()
        else:
            key.deactivate()
        logger.info("%s key %s.", 'Activated' if activate else 'Deactivated',
key_id)
    except ClientError:
        logger.exception(
            "Couldn't %s key %s.", 'Activate' if activate else 'Deactivate',
key_id)
        raise

def delete_key(user_name, key_id):
    """
    Deletes a user's access key.

    :param user_name: The user that owns the key.
    :param key_id: The ID of the key to delete.
    """

    try:
        key = iam.AccessKey(user_name, key_id)
        key.delete()
        logger.info(
            "Deleted access key %s for %s.", key.id, key.user_name)
    except ClientError:
        logger.exception("Couldn't delete key %s for %s", key_id, user_name)
        raise

```

Use the wrapper functions to perform access key actions for the current user.

```
def usage_demo():
    """Shows how to create and manage access keys."""
    def print_keys():
        """Gets and prints the current keys for a user."""
        current_keys = list_keys(current_user_name)
        print("The current user's keys are now:")
        print(*[f"{key.id}: {key.status}" for key in current_keys], sep='\n')

    logging.basicConfig(level=logging.INFO, format='%(levelname)s: %(message)s')
    print('*'*88)
    print("Welcome to the AWS Identity and Account Management access key demo.")
    print('*'*88)
    current_user_name = iam.CurrentUser().user_name
    print(f"This demo creates an access key for the current user "
          f"\n{current_user_name}, manipulates the key in a few ways, and then "
          f"\ndeletes it.")
    all_keys = list_keys(current_user_name)
    if len(all_keys) == 2:
        print("The current user already has the maximum of 2 access keys. To run "
              "this demo, either delete one of the access keys or use a user "
              "that has only 1 access key.")
    else:
        new_key = create_key(current_user_name)
        print(f"Created a new key with id {new_key.id} and secret "
              f"\n{new_key.secret}.")
        print_keys()
        existing_key = next(key for key in all_keys if key != new_key)
        last_use = get_last_use(existing_key.id)['AccessKeyLastUsed']
        print(f"Key {all_keys[0].id} was last used to access "
              f"\n{last_use['ServiceName']} "
              f"\non {last_use['LastUsedDate']}")
        update_key(current_user_name, new_key.id, False)
        print(f"Key {new_key.id} is now deactivated.")
        print_keys()
        delete_key(current_user_name, new_key.id)
        print_keys()
        print("Thanks for watching!")
```

- For API details, see the following topics in *AWS SDK for Python (Boto3) API Reference*.
 - [CreateAccessKey](#)
 - [DeleteAccessKey](#)
 - [GetAccessKeyLastUsed](#)
 - [ListAccessKeys](#)
 - [UpdateAccessKey](#)

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Manage IAM policies using an AWS SDK

The following code example shows how to:

- Create and list policies.
- Create and get policy versions.
- Roll back a policy to a previous version.

- Delete policies.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Create functions that wrap IAM policy actions.

```
import json
import logging
import operator
import pprint
import time

import boto3
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)
iam = boto3.resource('iam')

def create_policy(name, description, actions, resource_arn):
    """
    Creates a policy that contains a single statement.

    :param name: The name of the policy to create.
    :param description: The description of the policy.
    :param actions: The actions allowed by the policy. These typically take the
                    form of service:action, such as s3:PutObject.
    :param resource_arn: The Amazon Resource Name (ARN) of the resource this policy
                        applies to. This ARN can contain wildcards, such as
                        'arn:aws:s3:::my-bucket/*' to allow actions on all objects
                        in the bucket named 'my-bucket'.
    :return: The newly created policy.
    """

    policy_doc = {
        "Version": "2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Action": actions,
                "Resource": resource_arn
            }
        ]
    }
    try:
        policy = iam.create_policy(
            PolicyName=name, Description:description,
            PolicyDocument=json.dumps(policy_doc))
        logger.info("Created policy %s.", policy.arn)
    except ClientError:
        logger.exception("Couldn't create policy %s.", name)
        raise
    else:
        return policy

def list_policies(scope):
    """
    Lists the policies in the current account.

    :param scope: Limits the kinds of policies that are returned. For example,
    
```

```

        'Local' specifies that only locally managed policies are
returned.
:return: The list of policies.
"""
try:
    policies = list(iam.policies.filter(Scope=scope))
    logger.info("Got %s policies in scope '%s'.", len(policies), scope)
except ClientError:
    logger.exception("Couldn't get policies for scope '%s'.", scope)
    raise
else:
    return policies

def create_policy_version(policy_arn, actions, resource_arn, set_as_default):
"""
Creates a policy version. Policies can have up to five versions. The default
version is the one that is used for all resources that reference the policy.

:param policy_arn: The ARN of the policy.
:param actions: The actions to allow in the policy version.
:param resource_arn: The ARN of the resource this policy version applies to.
:param set_as_default: When True, this policy version is set as the default
                        version for the policy. Otherwise, the default
                        is not changed.
:return: The newly created policy version.
"""
policy_doc = {
    'Version': '2012-10-17',
    'Statement': [
        {
            'Effect': 'Allow',
            'Action': actions,
            'Resource': resource_arn
        }
    ]
}
try:
    policy = iam.Policy(policy_arn)
    policy_version = policy.create_version(
        PolicyDocument=json.dumps(policy_doc), SetAsDefault=set_as_default)
    logger.info(
        "Created policy version %s for policy %s.",
        policy_version.version_id, policy_version.arn)
except ClientError:
    logger.exception("Couldn't create a policy version for %s.", policy_arn)
    raise
else:
    return policy_version

def get_default_policy_statement(policy_arn):
"""
Gets the statement of the default version of the specified policy.

:param policy_arn: The ARN of the policy to look up.
:return: The statement of the default policy version.
"""
try:
    policy = iam.Policy(policy_arn)
    # To get an attribute of a policy, the SDK first calls get_policy.
    policy_doc = policy.default_version.document
    policy_statement = policy_doc.get('Statement', None)
    logger.info("Got default policy doc for %s.", policy.policy_name)
    logger.info(policy_doc)
except ClientError:
    logger.exception("Couldn't get default policy statement for %s.",
                     policy_arn)

```

```

        raise
    else:
        return policy_statement

def rollback_policy_version(policy_arn):
    """
    Rolls back to the previous default policy, if it exists.

    1. Gets the list of policy versions in order by date.
    2. Finds the default.
    3. Makes the previous policy the default.
    4. Deletes the old default version.

    :param policy_arn: The ARN of the policy to roll back.
    :return: The default version of the policy after the rollback.
    """
    try:
        policy_versions = sorted(
            iam.Policy(policy_arn).versions.all(),
            key=operator.attrgetter('create_date'))
        logger.info("Got %s versions for %s.", len(policy_versions), policy_arn)
    except ClientError:
        logger.exception("Couldn't get versions for %s.", policy_arn)
        raise

    default_version = None
    rollback_version = None
    try:
        while default_version is None:
            ver = policy_versions.pop()
            if ver.is_default_version:
                default_version = ver
        rollback_version = policy_versions.pop()
        rollback_version.set_as_default()
        logger.info("Set %s as the default version.", rollback_version.version_id)
        default_version.delete()
        logger.info("Deleted original default version %s.",
                   default_version.version_id)
    except IndexError:
        if default_version is None:
            logger.warning("No default version found for %s.", policy_arn)
        elif rollback_version is None:
            logger.warning(
                "Default version %s found for %s, but no previous version exists,
so "
                "nothing to roll back to.", default_version.version_id, policy_arn)
    except ClientError:
        logger.exception("Couldn't roll back version for %s.", policy_arn)
        raise
    else:
        return rollback_version

def delete_policy(policy_arn):
    """
    Deletes a policy.

    :param policy_arn: The ARN of the policy to delete.
    """
    try:
        iam.Policy(policy_arn).delete()
        logger.info("Deleted policy %s.", policy_arn)
    except ClientError:
        logger.exception("Couldn't delete policy %s.", policy_arn)
        raise

```

Use the wrapper functions to create policies, update versions, and get information about them.

```
def usage_demo():
    """Shows how to use the policy functions."""
    logging.basicConfig(level=logging.INFO, format='%(levelname)s: %(message)s')
    print('*'*88)
    print("Welcome to the AWS Identity and Account Management policy demo.")
    print('*'*88)
    print("Policies let you define sets of permissions that can be attached to "
          "other IAM resources, like users and roles.")
    bucket_arn = f'arn:aws:s3:::made-up-bucket-name'
    policy = create_policy(
        'demo-iam-policy', 'Policy for IAM demonstration.',
        ['s3>ListObjects'], bucket_arn)
    print(f"Created policy {policy.policy_name}.")
    policies = list_policies('Local')
    print(f"Your account has {len(policies)} managed policies:")
    print(*[pol.policy_name for pol in policies], sep=', ')
    time.sleep(1)
    policy_version = create_policy_version(
        policy.arn, ['s3:PutObject'], bucket_arn, True)
    print(f"Added policy version {policy_version.version_id} to policy "
          f"{policy.policy_name}.")
    default_statement = get_default_policy_statement(policy.arn)
    print(f"The default policy statement for {policy.policy_name} is:")
    pprint.pprint(default_statement)
    rollback_version = rollback_policy_version(policy.arn)
    print(f"Rolled back to version {rollback_version.version_id} for "
          f"{policy.policy_name}.")
    default_statement = get_default_policy_statement(policy.arn)
    print(f"The default policy statement for {policy.policy_name} is now:")
    pprint.pprint(default_statement)
    delete_policy(policy.arn)
    print(f"Deleted policy {policy.policy_name}.")
    print("Thanks for watching!")
```

- For API details, see the following topics in *AWS SDK for Python (Boto3) API Reference*.
 - [CreatePolicy](#)
 - [CreatePolicyVersion](#)
 - [DeletePolicy](#)
 - [DeletePolicyVersion](#)
 - [GetPolicyVersion](#)
 - [ListPolicies](#)
 - [ListPolicyVersions](#)
 - [SetDefaultPolicyVersion](#)

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Manage IAM roles using an AWS SDK

The following code example shows how to:

- Create an IAM role.
- Attach and detach policies for a role.
- Delete a role.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Create functions that wrap IAM role actions.

```
import json
import logging
import pprint

import boto3
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)
iam = boto3.resource('iam')

def create_role(role_name, allowed_services):
    """
    Creates a role that lets a list of specified services assume the role.

    :param role_name: The name of the role.
    :param allowed_services: The services that can assume the role.
    :return: The newly created role.
    """
    trust_policy = {
        'Version': '2012-10-17',
        'Statement': [
            {
                'Effect': 'Allow',
                'Principal': {'Service': service},
                'Action': 'sts:AssumeRole'
            } for service in allowed_services
        ]
    }

    try:
        role = iam.create_role(
            RoleName=role_name,
            AssumeRolePolicyDocument=json.dumps(trust_policy))
        logger.info("Created role %s.", role.name)
    except ClientError:
        logger.exception("Couldn't create role %s.", role_name)
        raise
    else:
        return role

def attach_policy(role_name, policy_arn):
    """
    Attaches a policy to a role.

    :param role_name: The name of the role. **Note** this is the name, not the ARN.
    :param policy_arn: The ARN of the policy.
    """
    try:
        iam.Role(role_name).attach_policy(PolicyArn=policy_arn)
        logger.info("Attached policy %s to role %s.", policy_arn, role_name)
    except ClientError:
        logger.exception("Couldn't attach policy %s to role %s.", policy_arn,
                         role_name)
        raise

def detach_policy(role_name, policy_arn):
```

```
"""
Detaches a policy from a role.

:param role_name: The name of the role. **Note** this is the name, not the ARN.
:param policy_arn: The ARN of the policy.
"""
try:
    iam.Role(role_name).detach_policy(PolicyArn=policy_arn)
    logger.info("Detached policy %s from role %s.", policy_arn, role_name)
except ClientError:
    logger.exception(
        "Couldn't detach policy %s from role %s.", policy_arn, role_name)
    raise

def delete_role(role_name):
    """
    Deletes a role.

    :param role_name: The name of the role to delete.
    """
    try:
        iam.Role(role_name).delete()
        logger.info("Deleted role %s.", role_name)
    except ClientError:
        logger.exception("Couldn't delete role %s.", role_name)
        raise
```

Use the wrapper functions to create a role, then attach and detach a policy.

```
def usage_demo():
    """Shows how to use the role functions."""
    logging.basicConfig(level=logging.INFO, format='%(levelname)s: %(message)s')
    print('*'*88)
    print("Welcome to the AWS Identity and Account Management role demo.")
    print('*'*88)
    print("Roles let you define sets of permissions and can be assumed by "
          "other entities, like users and services.")
    print("The first 10 roles currently in your account are:")
    roles = list_roles(10)
    print(f"The inline policies for role {roles[0].name} are:")
    list_policies(roles[0].name)
    role = create_role(
        'demo-iam-role',
        ['lambda.amazonaws.com', 'batchoperations.s3.amazonaws.com'])
    print(f"Created role {role.name}, with trust policy:")
    pprint.pprint(role.assume_role_policy_document)
    policy_arn = 'arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess'
    attach_policy(role.name, policy_arn)
    print(f"Attached policy {policy_arn} to {role.name}.")
    print(f"Policies attached to role {role.name} are:")
    list_attached_policies(role.name)
    detach_policy(role.name, policy_arn)
    print(f"Detached policy {policy_arn} from {role.name}.")
    delete_role(role.name)
    print(f"Deleted {role.name}.")
    print("Thanks for watching!")
```

- For API details, see the following topics in *AWS SDK for Python (Boto3) API Reference*.
 - [AttachRolePolicy](#)
 - [CreateRole](#)
 - [DeleteRole](#)

- [DetachRolePolicy](#)

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Manage your IAM account using an AWS SDK

The following code example shows how to:

- Get and update the account alias.
- Generate a report of users and credentials.
- Get a summary of account usage.
- Get details for all users, groups, roles, and policies in your account, including their relationships to each other.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Create functions that wrap IAM account actions.

```
import logging
import pprint
import sys
import time
import boto3
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)
iam = boto3.resource('iam')

def list_aliases():
    """
    Gets the list of aliases for the current account. An account has at most one
    alias.

    :return: The list of aliases for the account.
    """
    try:
        response = iam.meta.client.list_account_aliases()
        aliases = response['AccountAliases']
        if len(aliases) > 0:
            logger.info("Got aliases for your account: %s.", ','.join(aliases))
        else:
            logger.info("Got no aliases for your account.")
    except ClientError:
        logger.exception("Couldn't list aliases for your account.")
        raise
    else:
        return response['AccountAliases']

def create_alias(alias):
    """
    Creates an alias for the current account. The alias can be used in place of the
    account name when assuming roles or using access keys.
```

```

account ID in the sign-in URL. An account can have only one alias. When a new
alias is created, it replaces any existing alias.

:param alias: The alias to assign to the account.
"""

try:
    iam.create_account_alias(AccountAlias=alias)
    logger.info("Created an alias '%s' for your account.", alias)
except ClientError:
    logger.exception("Couldn't create alias '%s' for your account.", alias)
    raise

def delete_alias(alias):
"""
Removes the alias from the current account.

:param alias: The alias to remove.
"""

try:
    iam.meta.client.delete_account_alias(AccountAlias=alias)
    logger.info("Removed alias '%s' from your account.", alias)
except ClientError:
    logger.exception("Couldn't remove alias '%s' from your account.", alias)
    raise

def generate_credential_report():
"""
Starts generation of a credentials report about the current account. After
calling this function to generate the report, call get_credential_report
to get the latest report. A new report can be generated a minimum of four hours
after the last one was generated.
"""

try:
    response = iam.meta.client.generate_credential_report()
    logger.info("Generating credentials report for your account. "
                "Current state is %s.", response['State'])
except ClientError:
    logger.exception("Couldn't generate a credentials report for your
account.")
    raise
else:
    return response

def get_credential_report():
"""
Gets the most recently generated credentials report about the current account.

:return: The credentials report.
"""

try:
    response = iam.meta.client.get_credential_report()
    logger.debug(response['Content'])
except ClientError:
    logger.exception("Couldn't get credentials report.")
    raise
else:
    return response['Content']

def get_summary():
"""
Gets a summary of account usage.

:return: The summary of account usage.
"""

try:

```

```

        summary = iam.AccountSummary()
        logger.debug(summary.summary_map)
    except ClientError:
        logger.exception("Couldn't get a summary for your account.")
        raise
    else:
        return summary.summary_map

def get_authorization_details(response_filter):
    """
    Gets an authorization detail report for the current account.

    :param response_filter: A list of resource types to include in the report, such
                           as users or roles. When not specified, all resources
                           are included.
    :return: The authorization detail report.
    """
    try:
        account_details = iam.meta.client.get_account_authorization_details(
            Filter=response_filter
        )
        logger.debug(account_details)
    except ClientError:
        logger.exception("Couldn't get details for your account.")
        raise
    else:
        return account_details

```

Call wrapper functions to change the account alias and to get reports about the account.

```

def usage_demo():
    """
    Shows how to use the account functions.
    """
    logging.basicConfig(level=logging.INFO, format='%(levelname)s: %(message)s')
    print('*'*88)
    print("Welcome to the AWS Identity and Account Management account demo.")
    print('*'*88)
    print("Setting an account alias lets you use the alias in your sign-in URL "
          "instead of your account number.")
    old_aliases = list_aliases()
    if len(old_aliases) > 0:
        print(f"Your account currently uses '{old_aliases[0]}' as its alias.")
    else:
        print("Your account currently has no alias.")
    for index in range(1, 3):
        new_alias = f'alias-{index}-{time.time_ns()}''
        print(f"Setting your account alias to {new_alias}")
        create_alias(new_alias)
    current_aliases = list_aliases()
    print(f"Your account alias is now {current_aliases}.")
    delete_alias(current_aliases[0])
    print(f"Your account now has no alias.")
    if len(old_aliases) > 0:
        print(f"Restoring your original alias back to {old_aliases[0]}...")
        create_alias(old_aliases[0])

    print('*'*88)
    print("You can get various reports about your account.")
    print("Let's generate a credentials report...")
    report_state = None
    while report_state != 'COMPLETE':
        cred_report_response = generate_credential_report()
        old_report_state = report_state
        report_state = cred_report_response['State']
        if report_state != old_report_state:

```

```

        print(report_state, sep='')
    else:
        print('.', sep='')
    sys.stdout.flush()
    time.sleep(1)
print()
cred_report = get_credential_report()
col_count = 3
print(f"Got credentials report. Showing only the first {col_count} columns.")
cred_lines = [line.split(',')[:col_count] for line
             in cred_report.decode('utf-8').split('\n')]
col_width = max([len(item) for line in cred_lines for item in line]) + 2
for line in cred_report.decode('utf-8').split('\n'):
    print(''.join(element.ljust(col_width)
                  for element in line.split(',')[:col_count]))

print('*'*88)
print("Let's get an account summary.")
summary = get_summary()
print("Here's your summary:")
pprint.pprint(summary)

print('*'*88)
print("Let's get authorization details!")
details = get_authorization_details([])
see_details = input("These are pretty long, do you want to see them (y/n)? ")
if see_details.lower() == 'y':
    pprint.pprint(details)

print('*'*88)
pw_policy_created = None
see_pw_policy = input("Want to see the password policy for the account (y/n)? ")
if see_pw_policy.lower() == 'y':
    while True:
        if print_password_policy():
            break
        else:
            answer = input("Do you want to create a default password policy (y/n)? ")
            if answer.lower() == 'y':
                pw_policy_created = iam.create_account_password_policy()
            else:
                break
    if pw_policy_created is not None:
        answer = input("Do you want to delete the password policy (y/n)? ")
        if answer.lower() == 'y':
            pw_policy_created.delete()
            print("Password policy deleted.")

print("The SAML providers for your account are:")
list_saml_providers(10)

print('*'*88)
print("Thanks for watching.")

```

- For API details, see the following topics in *AWS SDK for Python (Boto3) API Reference*.
 - [CreateAccountAlias](#)
 - [DeleteAccountAlias](#)
 - [GenerateCredentialReport](#)
 - [GetAccountAuthorizationDetails](#)
 - [GetAccountSummary](#)

- [GetCredentialReport](#)
- [ListAccountAliases](#)

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Roll back an IAM policy version using an AWS SDK

The following code example shows how to:

- Get the list of policy versions in order by date.
- Find the default policy version.
- Make the previous policy version the default.
- Delete the old default version.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
def rollback_policy_version(policy_arn):
    """
    Rolls back to the previous default policy, if it exists.

    1. Gets the list of policy versions in order by date.
    2. Finds the default.
    3. Makes the previous policy the default.
    4. Deletes the old default version.

    :param policy_arn: The ARN of the policy to roll back.
    :return: The default version of the policy after the rollback.
    """
    try:
        policy_versions = sorted(
            iam.Policy(policy_arn).versions.all(),
            key=operator.attrgetter('create_date'))
        logger.info("Got %s versions for %s.", len(policy_versions), policy_arn)
    except ClientError:
        logger.exception("Couldn't get versions for %s.", policy_arn)
        raise

    default_version = None
    rollback_version = None
    try:
        while default_version is None:
            ver = policy_versions.pop()
            if ver.is_default_version:
                default_version = ver
        rollback_version = policy_versions.pop()
        rollback_version.set_as_default()
        logger.info("Set %s as the default version.", rollback_version.version_id)
        default_version.delete()
        logger.info("Deleted original default version %s.",
                   default_version.version_id)
    except IndexError:
```

```
        if default_version is None:
            logger.warning("No default version found for %s.", policy_arn)
        elif rollback_version is None:
            logger.warning(
                "Default version %s found for %s, but no previous version exists,
so "
                "nothing to roll back to.", default_version.version_id, policy_arn)
        except ClientError:
            logger.exception("Couldn't roll back version for %s.", policy_arn)
            raise
        else:
            return rollback_version
```

- For API details, see the following topics in *AWS SDK for Python (Boto3) API Reference*.
 - [DeletePolicyVersion](#)
 - [ListPolicyVersions](#)
 - [SetDefaultPolicyVersion](#)

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Work with the IAM Policy Builder API using an AWS SDK

The following code example shows how to:

- Create IAM policies by using the object-oriented API.
- Use the IAM Policy Builder API with the IAM service.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

The examples use the following imports.

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.awssdk.policybuilder.iam.IamConditionOperator;
import software.amazon.awssdk.policybuilder.iam.IamEffect;
import software.amazon.awssdk.policybuilder.iam.IamPolicy;
import software.amazon.awssdk.policybuilder.iam.IamPolicyWriter;
import software.amazon.awssdk.policybuilder.iam.IamPrincipal;
import software.amazon.awssdk.policybuilder.iam.IamPrincipalType;
import software.amazon.awssdk.policybuilder.iam.IamResource;
import software.amazon.awssdk.policybuilder.iam.IamStatement;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iam.IamClient;
import software.amazon.awssdk.services.iam.model.GetPolicyResponse;
import software.amazon.awssdk.services.iam.model.GetPolicyVersionResponse;
import software.amazon.awssdk.services.sts.StsClient;

import java.net.URLDecoder;
import java.nio.charset.StandardCharsets;
import java.util.Arrays;
```

```
import java.util.List;
```

Create a time-based policy.

```
public String timeBasedPolicyExample() {
    IamPolicy policy = IamPolicy.builder()
        .addStatement(b -> b
            .effect(IamEffect.ALLOW)
            .addAction("dynamodb:GetItem")
            .addResource(IamResource.ALL)
            .addCondition(b1 -> b1
                .operator(IamConditionOperator.DATE_GREATER_THAN)
                .key("aws:CurrentTime")
                .value("2020-04-01T00:00:00Z"))
            .addCondition(b1 -> b1
                .operator(IamConditionOperator.DATE_LESS_THAN)
                .key("aws:CurrentTime")
                .value("2020-06-30T23:59:59Z")))
        .build();

    // Use an IamPolicyWriter to write out the JSON string to a more readable
    // format.
    return policy.toJson(IamPolicyWriter.builder()
        .prettyPrint(true)
        .build());
}
```

Create a policy with multiple conditions.

```
public String multipleConditionsExample() {
    IamPolicy policy = IamPolicy.builder()
        .addStatement(b -> b
            .effect(IamEffect.ALLOW)
            .addAction("dynamodb:GetItem")
            .addAction("dynamodb:BatchGetItem")
            .addAction("dynamodb:Query")
            .addAction("dynamodb:PutItem")
            .addAction("dynamodb:UpdateItem")
            .addAction("dynamodb:DeleteItem")
            .addAction("dynamodb:BatchWriteItem")
            .addResource("arn:aws:dynamodb:*:*:table/table-name")

        .addConditions(IamConditionOperator.STRING_EQUALS.addPrefix("ForAllValues:"),  

                      "dynamodb:Attributes",
                      List.of("column-name1", "column-name2", "column-  

name3"))
            .addCondition(b1 ->
                b1.operator(IamConditionOperator.STRING_EQUALS.addSuffix("IfExists"))
                    .key("dynamodb:Select")
                    .value("SPECIFIC_ATTRIBUTES")))
        .build();

    return policy.toJson(IamPolicyWriter.builder()
        .prettyPrint(true).build());
}
```

Use principals in a policy.

```
public String specifyPrincipalsExample() {
```

```
IamPolicy policy = IamPolicy.builder()
    .addStatement(b -> b
        .effect(IamEffect.DENY)
        .addAction("s3:*")
        .addPrincipal(IamPrincipal.ALL)
        .addResource("arn:aws:s3:::BUCKETNAME/*")
        .addResource("arn:aws:s3:::BUCKETNAME")
        .addCondition(b1 -> b1
            .operator(IamConditionOperator.ARN_NOT_EQUALS)
            .key("aws:PrincipalArn")
            .value("arn:aws:iam::444455556666:user/user-
name")))
    .build();
return policy.toJson(IamPolicyWriter.builder()
    .prettyPrint(true).build());
}
```

Allow cross-account access.

```
public String allowCrossAccountAccessExample() {
    IamPolicy policy = IamPolicy.builder()
        .addStatement(b -> b
            .effect(IamEffect.ALLOW)
            .addPrincipal(IamPrincipalType.AWS, "111122223333")
            .addAction("s3:PutObject")
            .addResource("arn:aws:s3:::DOC-EXAMPLE-BUCKET/*")
            .addCondition(b1 -> b1
                .operator(IamConditionOperator.STRING_EQUALS)
                .key("s3:x-amz-acl")
                .value("bucket-owner-full-control")))
        .build();
    return policy.toJson(IamPolicyWriter.builder()
        .prettyPrint(true).build());
}
```

Build and upload an IamPolicy.

```
public String createAndUploadPolicyExample(IamClient iam, String accountID,
String policyName) {
    // Build the policy.
    IamPolicy policy =
        IamPolicy.builder() // 'version' defaults to "2012-10-17".
            .addStatement(IamStatement.builder()
                .effect(IamEffect.ALLOW)
                .addAction("dynamodb:PutItem")
                .addResource("arn:aws:dynamodb:us-east-1:" +
accountID + ":table/exampleTableName")
                .build())
            .build();
    // Upload the policy.
    iam.createPolicy(r ->
r.policyName(policyName).policyDocument(policy.toJson()));
    return policy.toJson(IamPolicyWriter.builder().prettyPrint(true).build());
}
```

Download and work with an IamPolicy.

```
public String createNewBasedOnExistingPolicyExample(IamClient iam, String
accountID, String policyName, String newPolicyName) {
```

```

        String policyArn = "arn:aws:iam://" + accountId + ":policy/" + policyName;
        GetPolicyResponse getPolicyResponse = iam.getPolicy(r ->
r.policyArn(policyArn));

        String policyVersion = getPolicyResponse.policy().defaultVersionId();
        GetPolicyVersionResponse getPolicyVersionResponse =
            iam.getPolicyVersion(r ->
r.policyArn(policyArn).versionId(policyVersion));

        // Create an IamPolicy instance from the JSON string returned from IAM.
        String decodedPolicy =
URLDecoder.decode(getPolicyVersionResponse.policyVersion().document(),
StandardCharsets.UTF_8);
        IamPolicy policy = IamPolicy.fromJson(decodedPolicy);

        /*
         All IamPolicy components are immutable, so use the copy method that
         creates a new instance that
         can be altered in the same method call.

         Add the ability to get an item from DynamoDB as an additional action.
        */
        IamStatement newStatement = policy.statements().get(0).copy(s ->
s.addAction("dynamodb:GetItem"));

        // Create a new statement that replaces the original statement.
        IamPolicy newPolicy = policy.copy(p ->
p.statements(Arrays.asList(newStatement)));

        // Upload the new policy. IAM now has both policies.
        iam.createPolicy(r -> r.policyName(newPolicyName)
            .policyDocument(newPolicy.toJson()));

        return
newPolicy.toJson(IamPolicyWriter.builder().prettyPrint(true).build());
    }
}

```

- For more information, see [AWS SDK for Java 2.x Developer Guide](#).
- For API details, see the following topics in *AWS SDK for Java 2.x API Reference*.
 - [CreatePolicy](#)
 - [GetPolicy](#)
 - [GetPolicyVersion](#)

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Code examples for AWS STS using AWS SDKs

The following code examples show how to use AWS STS with an AWS software development kit (SDK).

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Code examples

- [Actions for AWS STS using AWS SDKs \(p. 998\)](#)
 - [Assume a role with AWS STS using an AWS SDK \(p. 998\)](#)
 - [Get a session token with AWS STS using an AWS SDK \(p. 1007\)](#)
- [Scenarios for AWS STS using AWS SDKs \(p. 1008\)](#)
 - [Assume an IAM role that requires an MFA token with AWS STS using an AWS SDK \(p. 1008\)](#)
 - [Construct a URL with AWS STS for federated users using an AWS SDK \(p. 1012\)](#)
 - [Get a session token that requires an MFA token with AWS STS using an AWS SDK \(p. 1015\)](#)

Actions for AWS STS using AWS SDKs

The following code examples demonstrate how to perform individual AWS STS actions with AWS SDKs. These excerpts call the AWS STS API and are code excerpts from larger programs that must be run in context. Each example includes a link to GitHub, where you can find instructions for setting up and running the code.

The following examples include only the most commonly used actions. For a complete list, see the [AWS Security Token Service \(AWS STS\) API Reference](#).

Examples

- [Assume a role with AWS STS using an AWS SDK \(p. 998\)](#)
- [Get a session token with AWS STS using an AWS SDK \(p. 1007\)](#)

Assume a role with AWS STS using an AWS SDK

The following code examples show how to assume a role with AWS STS.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- [Assume an IAM role that requires an MFA token \(p. 1008\)](#)
- [Construct a URL for federated users \(p. 1012\)](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
using Amazon;
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;
using System;
using System.Threading.Tasks;
```

```

namespace AssumeRoleExample
{
    class AssumeRole
    {
        ///<summary>
        /// This example shows how to use the AWS Security Token
        /// Service (AWS STS) to assume an IAM role.
        ///
        /// NOTE: It is important that the role that will be assumed has a
        /// trust relationship with the account that will assume the role.
        ///
        /// Before you run the example, you need to create the role you want to
        /// assume and have it trust the IAM account that will assume that role.
        ///
        /// See https://docs.aws.amazon.com/IAM/latest/UserGuide/id\_roles\_create.html
        /// for help in working with roles.
        /// </summary>

        private static readonly RegionEndpoint REGION = RegionEndpoint.USWest2;

        static async Task Main()
        {
            // Create the SecurityToken client and then display the identity of the
            // default user.
            var roleArnToAssume = "arn:aws:iam::123456789012:role/testAssumeRole";

            var client = new
Amazon.SecurityToken.AmazonSecurityTokenServiceClient(REGION);

            // Get and display the information about the identity of the default
user.
            var callerIdRequest = new GetCallerIdentityRequest();
            var caller = await client.GetCallerIdentityAsync(callerIdRequest);
            Console.WriteLine($"Original Caller: {caller.Arн}");

            // Create the request to use with the AssumeRoleAsync call.
            var assumeRoleReq = new AssumeRoleRequest()
            {
                DurationSeconds = 1600,
                RoleSessionName = "Session1",
                RoleArn = roleArnToAssume
            };

            var assumeRoleRes = await client.AssumeRoleAsync(assumeRoleReq);

            // Now create a new client based on the credentials of the caller
            assuming the role.
            var client2 = new AmazonSecurityTokenServiceClient(credentials:
assumeRoleRes.Credentials);

            // Get and display information about the caller that has assumed the
defined role.
            var caller2 = await client2.GetCallerIdentityAsync(callerIdRequest);
            Console.WriteLine($"AssumedRole Caller: {caller2.Arн}");
        }
    }
}

```

- For API details, see [AssumeRole](#) in *AWS SDK for .NET API Reference*.

Bash

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
#####
function iecho() {
    if [[ $VERBOSE == true ]]; then
        echo "$@"
    fi
}

#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
    printf "%s\n" "$*" 2>&1
}

#####
# function sts_assume_role
#
# This function assumes a role in the AWS account and returns the temporary
# credentials.
#
# Parameters:
#     -n role_session_name -- The name of the session.
#     -r role_arn -- The ARN of the role to assume.
#
# Returns:
#     [access_key_id, secret_access_key, session_token]
#     And:
#         0 - If successful.
#         1 - If an error occurred.
#####
function sts_assume_role() {
    local role_session_name role_arn response
    local option OPTARG # Required to use getopts command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function sts_assume_role"
        echo "Assumes a role in the AWS account and returns the temporary credentials:"
        echo "  -n role_session_name -- The name of the session."
        echo "  -r role_arn -- The ARN of the role to assume."
        echo ""
    }

    while getopts n:r:h option; do
        case "${option}" in
            n) role_session_name=${OPTARG} ;;
            r) role_arn=${OPTARG} ;;
            h)
                usage
                ;;
        esac
    done
}
```

```

        return 0
    ;;
\?)
    ech o"Invalid parameter"
    usage
    return 1
;;
esac
done

response=$(aws sts assume-role \
--role-session-name "$role_session_name" \
--role-arn "$role_arn" \
--output text \
--query "Credentials.[AccessKeyId, SecretAccessKey, SessionToken]")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports create-role operation failed.\n$response"
    return 1
fi

echo "$response"

return 0
}

```

- For API details, see [AssumeRole](#) in *AWS CLI Command Reference*.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```

bool AwsDoc::STS::assumeRole(const Aws::String &roleArn,
                             const Aws::String &roleSessionName,
                             const Aws::String &externalId,
                             Aws::Auth::AWSCredentials &credentials,
                             const Aws::Client::ClientConfiguration &clientConfig)
{
    Aws::STS::STSCient sts(clientConfig);
    Aws::STS::Model::AssumeRoleRequest sts_req;

    sts_req.SetRoleArn(roleArn);
    sts_req.SetRoleSessionName(roleSessionName);
    sts_req.SetExternalId(externalId);

    const Aws::STS::Model::AssumeRoleOutcome outcome = sts.AssumeRole(sts_req);

    if (!outcome.IsSuccess()) {
        std::cerr << "Error assuming IAM role. " <<
                    outcome.GetError().GetMessage() << std::endl;
    }
    else {
        std::cout << "Credentials successfully retrieved." << std::endl;
        const Aws::STS::Model::AssumeRoleResult result = outcome.GetResult();
    }
}

```

```
    const Aws::STS::Model::Credentials &temp_credentials =
result.GetCredentials();

    // Store temporary credentials in return argument.
    // Note: The credentials object returned by assumeRole differs
    // from the AWS Credentials object used in most situations.
    credentials.SetAWSAccessKeyId(temp_credentials.GetAccessKeyId());
    credentials.SetAWSSecretKey(temp_credentials.GetSecretAccessKey());
    credentials.SetSessionToken(temp_credentials.GetSessionToken());
}

return outcome.IsSuccess();
}
```

- For API details, see [AssumeRole](#) in *AWS SDK for C++ API Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public static void assumeGivenRole(StsClient stsClient, String roleArn, String
roleSessionName) {

    try {
        AssumeRoleRequest roleRequest = AssumeRoleRequest.builder()
            .roleArn(roleArn)
            .roleSessionName(roleSessionName)
            .build();

        AssumeRoleResponse roleResponse = stsClient.assumeRole(roleRequest);
        Credentials myCreds = roleResponse.credentials();

        // Display the time when the temp creds expire.
        Instant exTime = myCreds.expiration();
        String tokenInfo = myCreds.sessionToken();

        // Convert the Instant to readable date.
        DateTimeFormatter formatter =
            DateTimeFormatter.ofLocalizedDateTime(FormatStyle.SHORT)
                .withLocale(Locale.US)
                .withZone(ZoneId.systemDefault());

        formatter.format(exTime);
        System.out.println("The token " + tokenInfo + " expires on " + exTime);

    } catch (StsException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}
```

- For API details, see [AssumeRole](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Create the client.

```
import { STSClient } from "@aws-sdk/client-sts";
// Set the AWS Region.
const REGION = "us-east-1";
// Create an AWS STS service client object.
export const client = new STSClient({ region: REGION });
```

Assume the IAM role.

```
import { AssumeRoleCommand } from "@aws-sdk/client-sts";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Returns a set of temporary security credentials that you can use to
    // access Amazon Web Services resources that you might not normally
    // have access to.
    const command = new AssumeRoleCommand({
      // The Amazon Resource Name (ARN) of the role to assume.
      RoleArn: "ROLE_ARN",
      // An identifier for the assumed role session.
      RoleSessionName: "session1",
      // The duration, in seconds, of the role session. The value specified
      // can range from 900 seconds (15 minutes) up to the maximum session
      // duration set for the role.
      DurationSeconds: 900,
    });
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- For API details, see [AssumeRole in AWS SDK for JavaScript API Reference](#).

SDK for JavaScript (v2)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// Load the AWS SDK for Node.js
const AWS = require('aws-sdk');
// Set the region
AWS.config.update({region: 'REGION'});

var roleToAssume = {RoleArn: 'arn:aws:iam::123456789012:role/RoleName',
  RoleSessionName: 'session1',
```

```

        DurationSeconds: 900,};

var roleCreds;

// Create the STS service object
var sts = new AWS.STS({apiVersion: '2011-06-15'});

//Assume Role
sts.assumeRole(roleToAssume, function(err, data) {
    if (err) console.log(err, err.stack);
    else{
        roleCreds = {accessKeyId: data.Credentials.AccessKeyId,
                     secretAccessKey: data.Credentials.SecretAccessKey,
                     sessionToken: data.Credentials.SessionToken};
        stsGetCallerIdentity(roleCreds);
    }
});

//Get Arn of current identity
function stsGetCallerIdentity(creds) {
    var stsParams = {credentials: creds };
    // Create STS service object
    var sts = new AWS.STS(stsParams);

    sts.getCallerIdentity({}, function(err, data) {
        if (err) {
            console.log(err, err.stack);
        }
        else {
            console.log(data.ArN);
        }
    });
}

```

- For API details, see [AssumeRole](#) in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Assume an IAM role that requires an MFA token and use temporary credentials to list Amazon S3 buckets for the account.

```

def list_buckets_from_assumed_role_with_mfa(
    assume_role_arn, session_name, mfa_serial_number, mfa_totp, sts_client):
    """
    Assumes a role from another account and uses the temporary credentials from
    that role to list the Amazon S3 buckets that are owned by the other account.
    Requires an MFA device serial number and token.

    The assumed role must grant permission to list the buckets in the other
    account.

    :param assume_role_arn: The Amazon Resource Name (ARN) of the role that
                           grants access to list the other account's buckets.
    :param session_name: The name of the STS session.
    :param mfa_serial_number: The serial number of the MFA device. For a virtual
                             MFA
                                         device, this is an ARN.

```

```
:param mfa_totp: A time-based, one-time password issued by the MFA device.  
:param sts_client: A Boto3 STS instance that has permission to assume the role.  
"""  
response = sts_client.assume_role(  
    RoleArn=assume_role_arn,  
    RoleSessionName=session_name,  
    SerialNumber=mfa_serial_number,  
    TokenCode=mfa_totp)  
temp_credentials = response['Credentials']  
print(f"Assumed role {assume_role_arn} and got temporary credentials.")  
  
s3_resource = boto3.resource(  
    's3',  
    aws_access_key_id=temp_credentials['AccessKeyId'],  
    aws_secret_access_key=temp_credentials['SecretAccessKey'],  
    aws_session_token=temp_credentials['SessionToken'])  
  
print(f"Listing buckets for the assumed role's account:")  
for bucket in s3_resource.buckets.all():  
    print(bucket.name)
```

- For API details, see [AssumeRole in AWS SDK for Python \(Boto3\) API Reference](#).

Ruby

SDK for Ruby

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
# Creates an AWS Security Token Service (AWS STS) client with specified  
credentials.  
# This is separated into a factory function so that it can be mocked for unit  
testing.  
#  
# @param key_id [String] The ID of the access key used by the STS client.  
# @param key_secret [String] The secret part of the access key used by the STS  
client.  
def create_sts_client(key_id, key_secret)  
  Aws::STS::Client.new(access_key_id: key_id, secret_access_key: key_secret)  
end  
  
# Gets temporary credentials that can be used to assume a role.  
#  
# @param role_arn [String] The ARN of the role that is assumed when these  
credentials  
#           are used.  
# @param sts_client [AWS::STS::Client] An AWS STS client.  
# @return [Aws::AssumeRoleCredentials] The credentials that can be used to assume  
the role.  
def assume_role(role_arn, sts_client)  
  credentials = Aws::AssumeRoleCredentials.new(  
    client: sts_client,  
    role_arn: role_arn,  
    role_session_name: "create-use-assume-role-scenario"  
  )  
  puts("Assumed role '#{role_arn}', got temporary credentials.")  
  credentials  
end
```

- For API details, see [AssumeRole](#) in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
async fn assume_role(
    config: &SdkConfig,
    region: Region,
    role_name: String,
    session_name: Option<String>,
) -> Result<(), Error> {
    match config.credentials_provider() {
        Some(credential) => {
            let provider = aws_config::sts::AssumeRoleProvider::builder(role_name)
                .region(region)
                .session_name(session_name.unwrap_or_else(|| String::from("rust-
assume-role")))
                .build(credential.clone());
            let local_config = aws_config::from_env()
                .credentials_provider(provider)
                .load()
                .await;
            let client = Client::new(&local_config);
            let req = client.get_caller_identity();
            let resp = req.send().await;
            match resp {
                Ok(e) => {
                    println!("UserID : {}", e.user_id().unwrap_or_default());
                    println!("Account: {}", e.account().unwrap_or_default());
                    println!("Arn : {}", e.arn().unwrap_or_default());
                }
                Err(e) => println!("{}:{}", e),
            }
        }
        None => {
            println!("No config provided");
        }
    }
    Ok(())
}
```

- For API details, see [AssumeRole](#) in *AWS SDK for Rust API reference*.

Swift

SDK for Swift

Note

This is prerelease documentation for an SDK in preview release. It is subject to change.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public func assumeRole(role: IAMClientTypes.Role, sessionName: String)
    async throws -> STSClientTypes.Credentials {
    let input = AssumeRoleInput(
        roleArn: role.arn,
        roleSessionName: sessionName
    )
    do {
        let output = try await stsClient.assumeRole(input: input)

        guard let credentials = output.credentials else {
            throw ServiceHandlerError.authError
        }

        return credentials
    } catch {
        throw error
    }
}
```

- For API details, see [AssumeRole](#) in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Get a session token with AWS STS using an AWS SDK

The following code example shows how to get a session token with AWS STS and use it to perform a service action that requires an MFA token.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Get a session token that requires an MFA token \(p. 1015\)](#)

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Get a session token by passing an MFA token and use it to list Amazon S3 buckets for the account.

```
def list_buckets_with_session_token_with_mfa(mfa_serial_number, mfa_totp,
    sts_client):
    """
    Gets a session token with MFA credentials and uses the temporary session
    credentials to list Amazon S3 buckets.

    Requires an MFA device serial number and token.

```

```
:param mfa_serial_number: The serial number of the MFA device. For a virtual
MFA
    device, this is an Amazon Resource Name (ARN).
:param mfa_totp: A time-based, one-time password issued by the MFA device.
:param sts_client: A Boto3 STS instance that has permission to assume the role.
"""
if mfa_serial_number is not None:
    response = sts_client.get_session_token(
        SerialNumber=mfa_serial_number, TokenCode=mfa_totp)
else:
    response = sts_client.get_session_token()
temp_credentials = response['Credentials']

s3_resource = boto3.resource(
    's3',
    aws_access_key_id=temp_credentials['AccessKeyId'],
    aws_secret_access_key=temp_credentials['SecretAccessKey'],
    aws_session_token=temp_credentials['SessionToken'])

print(f"Buckets for the account:")
for bucket in s3_resource.buckets.all():
    print(bucket.name)
```

- For API details, see [GetSessionToken](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Scenarios for AWS STS using AWS SDKs

The following code examples show you how to implement common scenarios in AWS STS with AWS SDKs. These scenarios show you how to accomplish specific tasks by calling multiple functions within AWS STS. Each scenario includes a link to GitHub, where you can find instructions on how to set up and run the code.

Examples

- [Assume an IAM role that requires an MFA token with AWS STS using an AWS SDK \(p. 1008\)](#)
- [Construct a URL with AWS STS for federated users using an AWS SDK \(p. 1012\)](#)
- [Get a session token that requires an MFA token with AWS STS using an AWS SDK \(p. 1015\)](#)

Assume an IAM role that requires an MFA token with AWS STS using an AWS SDK

The following code example shows how to assume a role that requires an MFA token.

Warning

To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as [AWS IAM Identity Center \(successor to AWS Single Sign-On\)](#).

- Create an IAM role that grants permission to list Amazon S3 buckets.
- Create an IAM user that has permission to assume the role only when MFA credentials are provided.
- Register an MFA device for the user.
- Assume the role and use temporary credentials to list S3 buckets.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Create an IAM user, register an MFA device, and create a role that grants permission to list S3 buckets. The user has rights only to assume the role.

```
def setup(iam_resource):
    """
    Creates a new user with no permissions.
    Creates a new virtual MFA device.
    Displays the QR code to seed the device.
    Asks for two codes from the MFA device.
    Registers the MFA device for the user.
    Creates an access key pair for the user.
    Creates a role with a policy that lets the user assume the role and requires
    MFA.
    Creates a policy that allows listing Amazon S3 buckets.
    Attaches the policy to the role.
    Creates an inline policy for the user that lets the user assume the role.

    For demonstration purposes, the user is created in the same account as the
    role,
    but in practice the user would likely be from another account.

    Any MFA device that can scan a QR code will work with this demonstration.
    Common choices are mobile apps like LastPass Authenticator,
    Microsoft Authenticator, or Google Authenticator.

:param iam_resource: A Boto3 AWS Identity and Access Management (IAM) resource
                     that has permissions to create users, roles, and policies
                     in the account.
:return: The newly created user, user key, virtual MFA device, and role.
"""
user = iam_resource.create_user(UserName=unique_name('user'))
print(f"Created user {user.name}.")

virtual_mfa_device = iam_resource.create_virtual_mfa_device(
    VirtualMFADeviceName=unique_name('mfa'))
print(f"Created virtual MFA device {virtual_mfa_device.serial_number}")

print(f"Showing the QR code for the device. Scan this in the MFA app of your "
      f"choice.")
with open('qr.png', 'wb') as qr_file:
    qr_file.write(virtual_mfa_device.qr_code_png)
webbrowser.open(qr_file.name)

print(f"Enter two consecutive code from your MFA device.")
mfa_code_1 = input("Enter the first code: ")
mfa_code_2 = input("Enter the second code: ")
user.enable_mfa(
    SerialNumber=virtual_mfa_device.serial_number,
    AuthenticationCode1=mfa_code_1,
    AuthenticationCode2=mfa_code_2)
os.remove(qr_file.name)
print(f"MFA device is registered with the user.")

user_key = user.create_access_key_pair()
print(f"Created access key pair for user.")

print(f"Wait for user to be ready.", end='')
```

```

progress_bar(10)

role = iam_resource.create_role(
    RoleName=unique_name('role'),
    AssumeRolePolicyDocument=json.dumps({
        'Version': '2012-10-17',
        'Statement': [
            {
                'Effect': 'Allow',
                'Principal': {'AWS': user.arn},
                'Action': 'sts:AssumeRole',
                'Condition': {'Bool': {'aws:MultiFactorAuthPresent': True}}
            }
        ]
    })
print(f"Created role {role.name} that requires MFA.")

policy = iam_resource.create_policy(
    PolicyName=unique_name('policy'),
    PolicyDocument=json.dumps({
        'Version': '2012-10-17',
        'Statement': [
            {
                'Effect': 'Allow',
                'Action': 's3>ListAllMyBuckets',
                'Resource': 'arn:aws:s3:::/*'
            }
        ]
    })
)
role.attach_policy(PolicyArn=policy.arn)
print(f"Created policy {policy.policy_name} and attached it to the role.")

user.create_policy(
    PolicyName=unique_name('user-policy'),
    PolicyDocument=json.dumps({
        'Version': '2012-10-17',
        'Statement': [
            {
                'Effect': 'Allow',
                'Action': 'sts:AssumeRole',
                'Resource': role.arn
            }
        ]
    })
)
print(f"Created an inline policy for {user.name} that lets the user assume "
      f"the role.")

print("Give AWS time to propagate these new resources and connections.",
      end='')
progress_bar(10)

return user, user_key, virtual_mfa_device, role

```

Show that assuming the role without an MFA token is not allowed.

```

def try_to_assume_role_without_mfa(assume_role_arn, session_name, sts_client):
    """
    Shows that attempting to assume the role without sending MFA credentials
    results
    in an AccessDenied error.

```

```
:param assume_role_arn: The Amazon Resource Name (ARN) of the role to assume.
:param session_name: The name of the STS session.
:param sts_client: A Boto3 STS instance that has permission to assume the role.
"""
print(f"Trying to assume the role without sending MFA credentials...")
try:
    sts_client.assume_role(RoleArn=assume_role_arn,
                           RoleSessionName=session_name)
    raise RuntimeError("Expected AccessDenied error.")
except ClientError as error:
    if error.response['Error']['Code'] == 'AccessDenied':
        print("Got AccessDenied.")
    else:
        raise
```

Assume the role that grants permission to list S3 buckets, passing the required MFA token, and show that buckets can be listed.

```
def list_buckets_from_assumed_role_with_mfa(
    assume_role_arn, session_name, mfa_serial_number, mfa_totp, sts_client):
    """
    Assumes a role from another account and uses the temporary credentials from
    that role to list the Amazon S3 buckets that are owned by the other account.
    Requires an MFA device serial number and token.

    The assumed role must grant permission to list the buckets in the other
    account.

    :param assume_role_arn: The Amazon Resource Name (ARN) of the role that
                           grants access to list the other account's buckets.
    :param session_name: The name of the STS session.
    :param mfa_serial_number: The serial number of the MFA device. For a virtual
                             MFA
                                         device, this is an ARN.
    :param mfa_totp: A time-based, one-time password issued by the MFA device.
    :param sts_client: A Boto3 STS instance that has permission to assume the role.
    """
    response = sts_client.assume_role(
        RoleArn=assume_role_arn,
        RoleSessionName=session_name,
        SerialNumber=mfa_serial_number,
        TokenCode=mfa_totp)
    temp_credentials = response['Credentials']
    print(f"Assumed role {assume_role_arn} and got temporary credentials.")

    s3_resource = boto3.resource(
        's3',
        aws_access_key_id=temp_credentials['AccessKeyId'],
        aws_secret_access_key=temp_credentials['SecretAccessKey'],
        aws_session_token=temp_credentials['SessionToken'])

    print(f"Listing buckets for the assumed role's account:")
    for bucket in s3_resource.buckets.all():
        print(bucket.name)
```

Destroy the resources created for the demo.

```
def teardown(user, virtual_mfa_device, role):
    """
    Removes all resources created during setup.
```

```
:param user: The demo user.
:param role: The demo role.
"""
for attached in role.attached_policies.all():
    policy_name = attached.policy_name
    role.detach_policy(PolicyArn=attached.arn)
    attached.delete()
    print(f"Detached and deleted {policy_name}.")
role.delete()
print(f"Deleted {role.name}.")
for user_pol in user.policies.all():
    user_pol.delete()
    print("Deleted inline user policy.")
for key in user.access_keys.all():
    key.delete()
    print("Deleted user's access key.")
for mfa in user.mfa_devices.all():
    mfa.disassociate()
    virtual_mfa_device.delete()
user.delete()
print(f"Deleted {user.name}.")
```

Run this scenario by using the previously defined functions.

```
def usage_demo():
    """Drives the demonstration."""
    print('*'*88)
    print(f'Welcome to the AWS Security Token Service assume role demo, '
          f"starring multi-factor authentication (MFA)!")
    print('*'*88)
    iam_resource = boto3.resource('iam')
    user, user_key, virtual_mfa_device, role = setup(iam_resource)
    print(f"Created {user.name} and {role.name}.")
    try:
        sts_client = boto3.client(
            'sts', aws_access_key_id=user_key.id,
            aws_secret_access_key=user_key.secret)
        try_to_assume_role_without_mfa(role.arn, 'demo-sts-session', sts_client)
        mfa_totp = input('Enter the code from your registered MFA device: ')
        list_buckets_from_assumed_role_with_mfa(
            role.arn, 'demo-sts-session', virtual_mfa_device.serial_number,
            mfa_totp, sts_client)
    finally:
        teardown(user, virtual_mfa_device, role)
    print("Thanks for watching!")
```

- For API details, see [AssumeRole](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Construct a URL with AWS STS for federated users using an AWS SDK

The following code example shows how to:

- Create an IAM role that grants read-only access to the current account's Amazon S3 resources.

- Get a security token from the AWS federation endpoint.
- Construct a URL that can be used to access the console with federated credentials.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Create a role that grants read-only access to the current account's S3 resources.

```
def setup(iam_resource):
    """
    Creates a role that can be assumed by the current user.
    Attaches a policy that allows only Amazon S3 read-only access.

    :param iam_resource: A Boto3 AWS Identity and Access Management (IAM) instance
                          that has the permission to create a role.
    :return: The newly created role.
    """

    role = iam_resource.create_role(
        RoleName=unique_name('role'),
        AssumeRolePolicyDocument=json.dumps([
            'Version': '2012-10-17',
            'Statement': [
                {
                    'Effect': 'Allow',
                    'Principal': {'AWS': iam_resource.CurrentUser().arn},
                    'Action': 'sts:AssumeRole'
                }
            ]
        })
    role.attach_policy(PolicyArn='arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess')
    print(f"Created role {role.name}.")

    print("Give AWS time to propagate these new resources and connections.",
          end='')
    progress_bar(10)

    return role
```

Get a security token from the AWS federation endpoint and construct a URL that can be used to access the console with federated credentials.

```
def construct_federated_url(assume_role_arn, session_name, issuer, sts_client):
    """
    Constructs a URL that gives federated users direct access to the AWS Management
    Console.

    1. Acquires temporary credentials from AWS Security Token Service (AWS STS)
       that
           can be used to assume a role with limited permissions.
    2. Uses the temporary credentials to request a sign-in token from the
       AWS federation endpoint.
    3. Builds a URL that can be used in a browser to navigate to the AWS federation
       endpoint, includes the sign-in token for authentication, and redirects to
       the AWS Management Console with permissions defined by the role that was
       specified in step 1.
    """

    # Step 1: Acquire temporary credentials
    temp_credentials = sts_client.get_federation_token(
        DurationSeconds=3600,
        FederateUser=session_name,
        SourceIdentity=assume_role_arn
    )
```

```

:param assume_role_arn: The role that specifies the permissions that are
granted.
The current user must have permission to assume the
role.
:param session_name: The name for the STS session.
:param issuer: The organization that issues the URL.
:param sts_client: A Boto3 STS instance that can assume the role.
:return: The federated URL.
"""
response = sts_clientassume_role(
    RoleArn=assume_role_arn, RoleSessionName=session_name)
temp_credentials = response['Credentials']
print(f"Assumed role {assume_role_arn} and got temporary credentials.")

session_data = {
    'sessionId': temp_credentials['AccessKeyId'],
    'sessionKey': temp_credentials['SecretAccessKey'],
    'sessionToken': temp_credentials['SessionToken']
}
aws_federated_signin_endpoint = 'https://signin.aws.amazon.com/federation'

# Make a request to the AWS federation endpoint to get a sign-in token.
# The requests.get function URL-encodes the parameters and builds the query
string
# before making the request.
response = requests.get(
    aws_federated_signin_endpoint,
    params={
        'Action': 'getSigninToken',
        'SessionDuration': str(datetime.timedelta(hours=12).seconds),
        'Session': json.dumps(session_data)
    })
signin_token = json.loads(response.text)
print(f"Got a sign-in token from the AWS sign-in federation endpoint.")

# Make a federated URL that can be used to sign into the AWS Management
Console.
query_string = urllib.parse.urlencode({
    'Action': 'login',
    'Issuer': issuer,
    'Destination': 'https://console.aws.amazon.com/',
    'SigninToken': signin_token['SigninToken']
})
federated_url = f'{aws_federated_signin_endpoint}?{query_string}'
return federated_url

```

Destroy the resources created for the demo.

```

def teardown(role):
    """
    Removes all resources created during setup.

:param role: The demo role.
"""
for attached in role.attached_policies.all():
    role.detach_policy(PolicyArn=attached.arn)
    print(f"Detached {attached.policy_name}.")
role.delete()
print(f"Deleted {role.name}.")

```

Run this scenario by using the previously defined functions.

```
def usage_demo():
    """Drives the demonstration."""
    print('*'*88)
    print(f"Welcome to the AWS Security Token Service federated URL demo.")
    print('*'*88)
    iam_resource = boto3.resource('iam')
    role = setup(iam_resource)
    sts_client = boto3.client('sts')
    try:
        federated_url = construct_federated_url(
            role.arn, 'AssumeRoleDemoSession', 'example.org', sts_client)
        print("Constructed a federated URL that can be used to connect to the "
              "AWS Management Console with role-defined permissions:")
        print('*'*88)
        print(federated_url)
        print('*'*88)
        _ = input("Copy and paste the above URL into a browser to open the AWS "
                  "Management Console with limited permissions. When done, press "
                  "'Enter' to clean up and complete this demo.")
    finally:
        teardown(role)
        print("Thanks for watching!")
```

- For API details, see [AssumeRole in AWS SDK for Python \(Boto3\) API Reference](#).

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

Get a session token that requires an MFA token with AWS STS using an AWS SDK

The following code example shows how to get a session token that requires an MFA token.

Warning

To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as [AWS IAM Identity Center \(successor to AWS Single Sign-On\)](#).

- Create an IAM role that grants permission to list Amazon S3 buckets.
- Create an IAM user that has permission to assume the role only when MFA credentials are provided.
- Register an MFA device for the user.
- Provide MFA credentials to get a session token and use temporary credentials to list S3 buckets.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Create an IAM user, register an MFA device, and create a role that grants permission to let the user list S3 buckets only when MFA credentials are used.

```
def setup(iam_resource):
```

```
"""
Creates a new user with no permissions.
Creates a new virtual multi-factor authentication (MFA) device.
Displays the QR code to seed the device.
Asks for two codes from the MFA device.
Registers the MFA device for the user.
Creates an access key pair for the user.
Creates an inline policy for the user that lets the user list Amazon S3
buckets,
but only when MFA credentials are used.

Any MFA device that can scan a QR code will work with this demonstration.
Common choices are mobile apps like LastPass Authenticator,
Microsoft Authenticator, or Google Authenticator.

:param iam_resource: A Boto3 AWS Identity and Access Management (IAM) resource
                     that has permissions to create users, MFA devices, and
                     policies in the account.
:return: The newly created user, user key, and virtual MFA device.
"""
user = iam_resource.create_user(UserName=unique_name('user'))
print(f"Created user {user.name}.")

virtual_mfa_device = iam_resource.create_virtual_mfa_device(
    VirtualMFADeviceName=unique_name('mfa'))
print(f"Created virtual MFA device {virtual_mfa_device.serial_number}")

print(f"Showing the QR code for the device. Scan this in the MFA app of your "
      f"choice.")
with open('qr.png', 'wb') as qr_file:
    qr_file.write(virtual_mfa_device.qr_code_png)
webbrowser.open(qr_file.name)

print(f"Enter two consecutive code from your MFA device.")
mfa_code_1 = input("Enter the first code: ")
mfa_code_2 = input("Enter the second code: ")
user.enable_mfa(
    SerialNumber=virtual_mfa_device.serial_number,
    AuthenticationCode1=mfa_code_1,
    AuthenticationCode2=mfa_code_2)
os.remove(qr_file.name)
print(f"MFA device is registered with the user.")

user_key = user.create_access_key_pair()
print(f"Created access key pair for user.")

print(f"Wait for user to be ready.", end='')
progress_bar(10)

user.create_policy(
    PolicyName=unique_name('user-policy'),
    PolicyDocument=json.dumps({
        'Version': '2012-10-17',
        'Statement': [
            {
                'Effect': 'Allow',
                'Action': 's3>ListAllMyBuckets',
                'Resource': 'arn:aws:s3:::*',
                'Condition': {'Bool': {'aws:MultiFactorAuthPresent': True}}
            }
        ]
    })
)
print(f"Created an inline policy for {user.name} that lets the user list
buckets, "
      f"but only when MFA credentials are present.")
```

```

        print("Give AWS time to propagate these new resources and connections.",
end='')
progress_bar(10)

return user, user_key, virtual_mfa_device

```

Get temporary session credentials by passing an MFA token, and use the credentials to list S3 buckets for the account.

```

def list_buckets_with_session_token_with_mfa(mfa_serial_number, mfa_totp,
sts_client):
    """
    Gets a session token with MFA credentials and uses the temporary session
    credentials to list Amazon S3 buckets.

    Requires an MFA device serial number and token.

    :param mfa_serial_number: The serial number of the MFA device. For a virtual
MFA
                                device, this is an Amazon Resource Name (ARN).
    :param mfa_totp: A time-based, one-time password issued by the MFA device.
    :param sts_client: A Boto3 STS instance that has permission to assume the role.
    """
    if mfa_serial_number is not None:
        response = sts_client.get_session_token(
            SerialNumber=mfa_serial_number, TokenCode=mfa_totp)
    else:
        response = sts_client.get_session_token()
    temp_credentials = response['Credentials']

    s3_resource = boto3.resource(
        's3',
        aws_access_key_id=temp_credentials['AccessKeyId'],
        aws_secret_access_key=temp_credentials['SecretAccessKey'],
        aws_session_token=temp_credentials['SessionToken'])

    print(f"Buckets for the account:")
    for bucket in s3_resource.buckets.all():
        print(bucket.name)

```

Destroy the resources created for the demo.

```

def teardown(user, virtual_mfa_device):
    """
    Removes all resources created during setup.

    :param user: The demo user.
    :param role: The demo MFA device.
    """
    for user_pol in user.policies.all():
        user_pol.delete()
        print("Deleted inline user policy.")
    for key in user.access_keys.all():
        key.delete()
        print("Deleted user's access key.")
    for mfa in user.mfa_devices.all():
        mfa.disassociate()
    virtual_mfa_device.delete()
    user.delete()
    print(f"Deleted {user.name}.")

```

Run this scenario by using the previously defined functions.

```
def usage_demo():
    """Drives the demonstration."""
    print('*'*88)
    print(f"Welcome to the AWS Security Token Service assume role demo, "
          f"starring multi-factor authentication (MFA)!")
    print('*'*88)
    iam_resource = boto3.resource('iam')
    user, user_key, virtual_mfa_device = setup(iam_resource)
    try:
        sts_client = boto3.client(
            'sts', aws_access_key_id=user_key.id,
            aws_secret_access_key=user_key.secret)
        try:
            print("Listing buckets without specifying MFA credentials.")
            list_buckets_with_session_token_with_mfa(None, None, sts_client)
        except ClientError as error:
            if error.response['Error']['Code'] == 'AccessDenied':
                print("Got expected AccessDenied error.")
            mfa_totp = input('Enter the code from your registered MFA device: ')
            list_buckets_with_session_token_with_mfa(
                virtual_mfa_device.serial_number, mfa_totp, sts_client)
    finally:
        teardown(user, virtual_mfa_device)
        print("Thanks for watching!")
```

- For API details, see [GetSessionToken](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using IAM with an AWS SDK \(p. 21\)](#). This topic also includes information about getting started and details about previous SDK versions.

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

- [AWS security credentials \(p. 1019\)](#)
- [AWS security audit guidelines \(p. 1023\)](#)
- [Data protection in AWS Identity and Access Management \(p. 1027\)](#)
- [Logging and monitoring in AWS Identity and Access Management \(p. 1028\)](#)
- [Compliance validation for AWS Identity and Access Management \(p. 1029\)](#)
- [Resilience in AWS Identity and Access Management \(p. 1030\)](#)
- [Infrastructure security in AWS Identity and Access Management \(p. 1031\)](#)
- [Configuration and vulnerability analysis in AWS Identity and Access Management \(p. 1032\)](#)
- [Security best practices and use cases in AWS Identity and Access Management \(p. 1032\)](#)
- [AWS managed policies for AWS Identity and Access Management Access Analyzer \(p. 1040\)](#)

AWS security credentials

When you interact with AWS, you specify your *AWS security credentials* to verify who you are and whether you have permission to access the resources that you are requesting. AWS uses the security credentials to authenticate and authorize your requests.

For example, if you want to download a protected file from an Amazon Simple Storage Service (Amazon S3) bucket, your credentials must allow that access. If your credentials don't show you are authorized to download the file, AWS denies your request. However, your AWS security credentials aren't required for you to download a file in an Amazon S3 bucket that is publicly shared.

There are different types of users in AWS. All AWS users have security credentials. There is the account owner (root user), users in AWS IAM Identity Center (successor to AWS Single Sign-On), federated users, and IAM users.

Users have either long-term or temporary security credentials. Root user, IAM user, and access keys have long-term security credentials that do not expire. To protect long-term credentials you can [manage and rotate access keys \(p. 103\)](#), [change passwords \(p. 93\)](#), and [enable MFA \(p. 114\)](#).

IAM roles, users in AWS IAM Identity Center (successor to AWS Single Sign-On), and federated users have temporary security credentials. Temporary security credentials expire after a defined period of time or when the user ends their session. Temporary credentials work almost identically to long-term credentials, 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.

As a result, temporary credentials have the following advantages over long-term 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 \(p. 183\)](#).
- 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.

Security considerations

We recommend that you consider the following information when determining the security provisions for your AWS account:

- When you create an AWS account, we create the account root user. The credentials of the root user (account owner) allow full access to all resources in the account. The first task you perform with the root user is to grant another user administrative permissions to your AWS account so that you minimize the usage of the root user.
- You can't use IAM policies to deny the root user access to resources explicitly. You can only use an AWS Organizations [service control policy \(SCP\)](#) to limit the permissions of the root user.
- If you forget or lose your root user password, you must have access to the email address associated with your account in order to reset it.
- If you lose your root user access keys, you must be able to sign in to your account as the root user to create new ones.
- Do not use the root user for your everyday tasks. Use it to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see [Tasks that require root user credentials](#) in the *AWS Account Management Reference Guide*.
- Security credentials are account-specific. If you have access to multiple AWS accounts, you have separate credentials for each account.
- [Policies \(p. 485\)](#) determine what actions a user, role, or member of a user group can perform, on which AWS resources, and under what conditions. Using policies you can securely control access to AWS services and resources in your AWS account. If you must modify or revoke permissions in response to a security event, you delete or modify the policies instead of making changes directly to the identity.
- Be sure to save the sign-in credentials for your *Emergency Access* IAM user and any access keys you created for programmatic access in a secure location. If you lose your access keys, you must sign in to your account to create new ones.

- We strongly recommend that you use temporary credentials provided by IAM roles and federated users instead of the long-term credentials provided by IAM users and access keys.

Federated identity

Federated identities are users with external identities that are granted temporary AWS credentials that they can use to access secure AWS account resources. External identities can come from a corporate identity store (such as LDAP or Windows Active Directory) or from a third party (such as Login in with Amazon, Facebook, or Google). Federated identities do not sign in with the AWS Management Console or AWS access portal.

To enable federated identities to sign in to AWS, you must create a custom URL that includes `https://signin.amazonaws.com/federation`. For more information, see [Enabling custom identity broker access to the AWS console \(p. 233\)](#).

For more information about federated identities, see [Identity providers and federation \(p. 198\)](#).

Multi-factor authentication (MFA)

Multi-factor authentication (MFA) provides an extra level of security for users who can access your AWS account. For additional security, we recommend that you require MFA on the AWS account root user credentials and all IAM users. For more information, see [Using Multi-Factor Authentication \(MFA\) in AWS in the IAM User Guide](#).

When you activate MFA and you sign in to your AWS account, you are prompted for your sign-in credentials, plus a response generated by an MFA device, such as a code, a touch or tap, or a biometric scan. When you add MFA, your AWS account settings and resources are more secure.

By default, MFA isn't activated. You can activate and manage MFA devices for the AWS account root user by going to the [Security credentials](#) page or the [IAM](#) dashboard in the AWS Management Console. For more information about activating MFA for IAM users, see [Enabling MFA devices for users in AWS \(p. 115\)](#).

For more information about signing in with multi-factor authentication (MFA) devices, see [Using MFA devices with your IAM sign-in page \(p. 83\)](#).

Programmatic access

You provide your AWS access keys to make programmatic calls to AWS or to use the AWS Command Line Interface or AWS Tools for PowerShell. We recommend using short-term access keys when possible.

When you create a long-term access key, you create the access key ID (for example, AKIAIOSFODNN7EXAMPLE) and secret access key (for example, wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY) as a set. The secret access key is available for download only when you create it. If you don't download your secret access key or if you lose it, you must create a new one.

In many scenarios, you don't need long-term access keys that never expire (as you have when you create access keys for an IAM user). Instead, you can create IAM roles and generate temporary security credentials. Temporary security credentials include an access key ID and a secret access key, but they also include a security token that indicates when the credentials expire. After they expire, they're no longer valid.

Access key IDs beginning with AKIA are long-term access keys for an IAM user or an AWS account root user. Access key IDs beginning with ASIA are temporary credentials access keys that you create using AWS STS operations.

Users need programmatic access if they want to interact with AWS outside of the AWS Management Console. The way to grant programmatic access depends on the type of user that's accessing AWS.

To grant users programmatic access, choose one of the following options.

Which user needs programmatic access?	To	By
Workforce identity (Users managed in IAM Identity Center)	Use temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	<p>Following the instructions for the interface that you want to use.</p> <ul style="list-style-type: none"> For the AWS CLI, see Configuring the AWS CLI to use AWS IAM Identity Center (successor to AWS Single Sign-On) in the <i>AWS Command Line Interface User Guide</i>. For AWS SDKs, tools, and AWS APIs, see IAM Identity Center authentication in the <i>AWS SDKs and Tools Reference Guide</i>.
IAM	Use temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	<p>Following the instructions in Using temporary credentials with AWS resources in the <i>IAM User Guide</i>.</p>
IAM	(Not recommended) Use long-term credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	<p>Following the instructions for the interface that you want to use.</p> <ul style="list-style-type: none"> For the AWS CLI, see Authenticating using IAM user credentials in the <i>AWS Command Line Interface User Guide</i>. For AWS SDKs and tools, see Authenticate using long-term credentials in the <i>AWS SDKs and Tools Reference Guide</i>. For AWS APIs, see Managing access keys for IAM users in the <i>IAM User Guide</i>.

Alternatives for long-term access keys

For many common use cases, there are alternatives to long-term access keys. To improve your account security, consider the following.

- **Don't embed long-term access keys and secret access keys in your application code or in a code repository** – Instead, use AWS Secrets Manager, or other secrets management solution, so you don't have to hardcode keys in plaintext. The application or client can then retrieve secrets when needed. For more information, see [What is AWS Secrets Manager?](#) in the *AWS Secrets Manager User Guide*.

- **Use IAM roles to generate temporary security credentials whenever possible** – Always use mechanisms to issue temporary security credentials when possible, rather than long-term access keys. Temporary security credentials are more secure because they are not stored with the user but are generated dynamically and provided to the user when requested. Temporary security credentials have a limited lifetime so you don't have to manage or rotate them. Mechanisms that provide temporary access keys include IAM roles or the authentication of an IAM Identity Center user. For machines that run outside of AWS you can use [AWS Identity and Access Management Roles Anywhere](#).
- **Use alternatives to long-term access keys for the AWS Command Line Interface (AWS CLI) or the aws-shell** – Alternatives include the following.
 - **AWS CloudShell** is a browser-based, pre-authenticated shell that you can launch directly from the AWS Management Console. You can run AWS CLI commands against AWS services through your preferred shell (Bash, Powershell, or Z shell). When you do this, you don't need to download or install command line tools. For more information, see [What is AWS CloudShell?](#) in the [AWS CloudShell User Guide](#).
 - **AWS CLI Version 2 integration with AWS IAM Identity Center (successor to AWS Single Sign-On) (IAM Identity Center)**. You can authenticate users and provide short-term credentials to run AWS CLI commands. To learn more, see [Integrating AWS CLI with IAM Identity Center](#) in the [AWS IAM Identity Center \(successor to AWS Single Sign-On\) User Guide](#) and [Configuring the AWS CLI to use IAM Identity Center](#) in the [AWS Command Line Interface User Guide](#).
 - **Don't create long-term access keys for human users who need access to applications or AWS services** – IAM Identity Center can generate temporary access credentials for your external IdP users to access AWS services. This eliminates the need to create and manage long-term credentials in IAM. In IAM Identity Center, create an IAM Identity Center permission set that grants the external IdP users access. Then assign a group from IAM Identity Center to the permission set in the selected AWS accounts. For more information, see [What is AWS IAM Identity Center \(successor to AWS Single Sign-On\)](#), [Connect to your external identity provider](#), and [Permission sets](#) in the [AWS IAM Identity Center \(successor to AWS Single Sign-On\) User Guide](#).
 - **Don't store long-term access keys within an AWS compute service** – Instead, assign an IAM role to compute resources. This automatically supplies temporary credentials to grant access. For example, when you create an instance profile that is attached to an Amazon EC2 instance, you can assign an AWS role to the instance and make it available to all of its applications. An instance profile contains the role and enables programs that are running on the Amazon EC2 instance to get temporary credentials. To learn more, see [Using an IAM role to grant permissions to applications running on Amazon EC2 instances](#).

Accessing AWS using your AWS credentials

AWS requires different types of security credentials, depending on how you access AWS and what type of AWS user you are. For example, you use sign-in credentials for the AWS Management Console while you use access keys to make programmatic calls to AWS. Also, each identity you use, whether it be the account root user, an AWS Identity and Access Management (IAM) user, an AWS IAM Identity Center (successor to AWS Single Sign-On) user, or a federated identity, has unique credentials within AWS.

For step-by-step instructions on how to sign in to AWS according to your user type, see [How to sign in to AWS](#) in the [AWS Sign-In User Guide](#).

AWS security audit guidelines

Periodically audit your security configuration to make sure it meets your current business needs. An audit gives you an opportunity to remove unneeded IAM users, roles, groups, and policies, and to make sure that your users and software don't have excessive permissions.

Following are guidelines for systematically reviewing and monitoring your AWS resources for security best practices.

Tip

You can monitor your usage of IAM as it relates to security best practices by using [AWS Security Hub](#). Security Hub uses security controls to evaluate resource configurations and security standards to help you comply with various compliance frameworks. For more information about using Security Hub to evaluate IAM resources, see [AWS Identity and Access Management controls](#) in the AWS Security Hub User Guide.

Contents

- [When to perform a security audit \(p. 1024\)](#)
- [Guidelines for auditing \(p. 1024\)](#)
- [Review your AWS account credentials \(p. 1024\)](#)
- [Review your IAM users \(p. 1025\)](#)
- [Review your IAM groups \(p. 1025\)](#)
- [Review your IAM roles \(p. 1025\)](#)
- [Review your IAM providers for SAML and OpenID Connect \(OIDC\) \(p. 1025\)](#)
- [Review Your mobile apps \(p. 1026\)](#)
- [Tips for reviewing IAM policies \(p. 1026\)](#)

When to perform a security audit

Audit your security configuration in the following situations:

- On a periodic basis. Perform the steps described in this document at regular intervals as a best practice for security.
- If there are changes in your organization, such as people leaving.
- If you have stopped using one or more individual AWS services to verify that you have removed permissions that users in your account no longer need.
- If you've added or removed software in your accounts, such as applications on Amazon EC2 instances, AWS OpsWorks stacks, AWS CloudFormation templates, etc.
- If you suspect that an unauthorized person might have accessed your account.

Guidelines for auditing

As you review your account's security configuration, follow these guidelines:

- **Be thorough.** Look at all aspects of your security configuration, including those that are seldom used.
- **Don't assume.** If you are unfamiliar with some aspect of your security configuration (for example, the reasoning behind a particular policy or the existence of a role), investigate the business need until you understand the potential risk.
- **Keep things simple.** To make auditing (and management) easier, use IAM groups, IAM roles, consistent naming schemes, and straightforward policies.

Review your AWS account credentials

Take these steps when you audit your AWS account credentials:

1. If you have access keys for your root user that you're not using, you can remove them. We [strongly recommend](#) that you don't use root access keys for everyday work with AWS, and that instead you use users with temporary credentials, such users in AWS IAM Identity Center (successor to AWS Single Sign-On).

2. If you require access keys for your account, [rotate them regularly](#).

Review your IAM users

Take these steps when you audit your existing IAM users:

1. [List your users](#) and then [delete users](#) that aren't needed.
2. [Remove users from groups](#) that they don't require access to.
3. Review the policies attached to the groups the user is in. See [Tips for reviewing IAM policies \(p. 1026\)](#).
4. Delete security credentials that the user doesn't need or that might have been exposed. For example, an IAM user used for an application doesn't need a password (which is necessary only to sign in to AWS websites). Similarly, if a user doesn't use access keys, there's no reason for the user to have one. For more information, see [Managing Passwords for IAM users](#) and [Managing Access Keys for IAM users](#).

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 the date and time when the password or access key was last used. Consider removing credentials that haven't been used recently from your account. (Don't remove your Emergency Access user.) For more information, see [Getting Credential Reports for your AWS Account](#).

5. Rotate access keys regularly for use cases that require long-term credentials. For more information, see [Managing Passwords for IAM Users](#) and [Managing Access Keys for IAM users](#).
6. As a best practice, require human users to use federation with an identity provider to access AWS using temporary credentials. If possible, transition from IAM users to federated users, such as users in IAM Identity Center. Retain the minimum number of IAM users needed for your applications.

Review your IAM groups

Take these steps when you audit your IAM groups:

1. [List your groups](#) and then [delete groups](#) that you aren't using.
2. [Review users](#) in each group and [remove users](#) that don't belong.
3. Review the policies attached to the group. See [Tips for reviewing IAM policies \(p. 1026\)](#).

Review your IAM roles

Take these steps when you audit your IAM roles:

1. [List your roles](#) and then [delete roles](#) that you aren't using.
2. [Review](#) the role's trust policy. Make sure that you know who the principal is and that you understand why that account or user needs to be able to assume the role.
3. [Review](#) the access policy for the role to be sure that it grants suitable permissions to whoever assumes the role—see [Tips for reviewing IAM policies \(p. 1026\)](#).

Review your IAM providers for SAML and OpenID Connect (OIDC)

If you have created an IAM entity for establishing trust with a [SAML or OIDC identity provider \(IdP\)](#), take these steps:

1. Delete unused providers.
2. Download and review the AWS metadata documents for each SAML IdP and make sure the documents reflect your current business needs.
3. Get the latest metadata documents from the SAML IdPs and [update the provider in IAM](#).

Review Your mobile apps

If you have created a mobile app that makes requests to AWS, take these steps:

1. Make sure that the mobile app doesn't contain embedded access keys, even if they're in encrypted storage.
2. Get temporary credentials for the app by using APIs designed for that purpose.

Note

We recommend that you use [Amazon Cognito](#) to manage user identity in your app. This service lets you authenticate users using Login with Amazon, Facebook, Google, or any OpenID Connect (OIDC)-compatible identity provider. For more information, see [Amazon Cognito identity pools](#) in the [Amazon Cognito Developer Guide](#).

Tips for reviewing IAM policies

Policies are powerful and subtle, so it's important to study and understand the permissions granted by each policy. Use the following guidelines when reviewing policies:

- Attach policies to groups or roles instead of to individual users. If an individual user has a policy, make sure you understand why that user needs the policy.
- Make sure that IAM users, groups, and roles have the permissions that they need and don't have any additional permissions.
- Use the [IAM Policy Simulator](#) to test policies attached to users or groups.
- Remember that a user's permissions are the result of all applicable policies— both identity-based policies (on users, groups, or roles) and resource-based policies (on resources such as Amazon S3 buckets, Amazon SQS queues, Amazon SNS topics, and AWS KMS keys). It's important to examine all the policies that apply to a user and to understand the complete set of permissions granted to an individual user.
- Be aware that allowing a user to create an IAM user, group, role, or policy and attach a policy to the principal entity is effectively granting that user all permissions to all resources in your account. Users who can create policies and attach them to a user, group, or role can grant themselves any permissions. In general, don't grant IAM permissions to users or roles whom you don't trust with full access to the resources in your account. When conducting your security audit confirm that the following IAM permissions are granted to trusted identities:
 - `iam:PutGroupPolicy`
 - `iam:PutRolePolicy`
 - `iam:PutUserPolicy`
 - `iam>CreatePolicy`
 - `iam>CreatePolicyVersion`
 - `iam:AttachGroupPolicy`
 - `iam:AttachRolePolicy`
 - `iam:AttachUserPolicy`
- Make sure policies don't grant permissions for services that you don't use. For example, if you use [AWS managed policies](#), make sure the AWS managed policies that are in use in your account are for

services that you actually use. To find out which AWS managed policies are in use in your account, use the IAM [GetAccountAuthorizationDetails](#) API (AWS CLI command: `aws iam get-account-authorization-details`).

- If the policy grants a user permission to launch an Amazon EC2 instance, it might also allow the `iam:PassRole` action, but if so it should [explicitly list the roles](#) that the user can pass to the Amazon EC2 instance.
- Examine any values for the Action or Resource element that include *. When possible, grant Allow access to the individual actions and resources that users need. However, the following are reasons that it might be suitable to use * in a policy:
 - The policy is designed to grant administrative-level permissions.
 - The wildcard character is used for a set of similar actions (for example, `Describe*`) as a convenience, and you are comfortable with the complete list of actions that are referenced in this way.
 - The wildcard character is used to indicate a class of resources or a resource path (for example, `arn:aws:iam::account-id:users/division_abc/*`), and you are comfortable granting access to all the resources in that class or path.
 - A service action doesn't support resource-level permissions, and the only choice for a resource is *.
- Examine policy names to make sure they reflect the policy's function. For example, although a policy might have a name that includes "read only," the policy might actually grant write or change permissions.

For more information about planning for your security audit, see [Best Practices for Security, Identity, and Compliance](#) in the [AWS Architecture Center](#).

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 users with AWS IAM Identity Center (successor to AWS Single Sign-On) or 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 require TLS 1.2 and recommend TLS 1.3.
- 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 sensitive 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 confidential or sensitive information, such as your customers' email addresses, into tags or free-form text 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 tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not 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 1.2 and 1.3. All AWS STS endpoints support HTTPS for encrypting data in transit. For a list of AWS STS endpoints, see [Regions and endpoints \(p. 464\)](#).

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 \(p. 465\)](#).

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 \(p. 471\)](#). 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 \(p. 1032\)](#).

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.

To learn whether an AWS service is within the scope of specific compliance programs, see [AWS services in Scope by Compliance Program](#) and choose the compliance program that you are interested in. 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 baseline environments on AWS that are security and compliance focused.
- [Architecting for HIPAA Security and Compliance on Amazon Web Services](#) – This whitepaper describes how companies can use AWS to create HIPAA-eligible applications.

Note

Not all AWS services are HIPAA eligible. For more information, see the [HIPAA Eligible Services Reference](#).

- [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. Security Hub uses security controls to evaluate your AWS resources and to check your compliance against security industry standards and best practices. For a list of supported services and controls, see [Security Hub controls reference](#).

- [AWS Audit Manager](#) – This AWS service helps you continuously audit your AWS usage to simplify how you manage risk and compliance with regulations and industry standards.

Resilience in AWS Identity and Access Management

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions have multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. 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 self-sustaining, Region-based services that are available globally.

IAM is a critical AWS service. Every operation performed in AWS must be authenticated and authorized by IAM. IAM checks each request against the identities and policies stored in IAM to determine if the request is allowed or denied. IAM was designed with a separate *control plane* and *data plane* so that the service authenticates even during unexpected failures. IAM resources that are used in authorizations, such as roles and policies, are stored in the control plane. IAM customers can change the configuration of these resources by using IAM operations such as `DeletePolicy` and `AttachRolePolicy`. Those configuration change requests go to the control plane. There is one IAM control plane for all commercial AWS Regions, which is located in the US East (N. Virginia) Region. The IAM system then propagates configuration changes to the IAM data planes in every [enabled AWS Region](#). The IAM data plane is essentially a read-only replica of the IAM control plane configuration data. Each AWS Region has a completely independent instance of the IAM data plane, which performs authentication and authorization for requests from the same Region. In each Region, the IAM data plane is distributed across at least three Availability Zones, and has sufficient capacity to tolerate the loss of an Availability Zone without any customer impairment. Both the IAM control and data planes were built for *zero planned downtime*, with all software updates and scaling operations performed in a manner that is invisible to customers.

AWS STS requests always go to a single global endpoint by default. You can 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 \(p. 461\)](#).

Certain events can interrupt communication between AWS Regions over the network. However, even when you can't communicate with the global IAM endpoint, AWS STS can still authenticate IAM principals and IAM can authorize your requests. The specific details of an event that interrupts communication will determine your ability to access AWS services. In most situations, you can continue to use IAM credentials in your AWS environment. The following conditions might apply to an event that interrupts communication.

Access keys for IAM users

You can authenticate indefinitely in a Region with long-term [access keys for IAM users](#). When you use the AWS Command Line Interface and APIs, you can provide AWS access keys so that AWS can verify your identity in programmatic requests.

Important

As a [best practice \(p. 1032\)](#), we recommend that your users sign in with [temporary credentials](#) instead of long-term access keys.

Temporary credentials

You can [request new temporary credentials](#) with the AWS STS Regional [service endpoint](#) for at least 24 hours. The following API operations generate temporary credentials.

- `AssumeRole`

- [AssumeRoleWithWebIdentity](#)
- [AssumeRoleWithSAML](#)
- [GetFederationToken](#)
- [GetSessionToken](#)

Principals and permissions

- You might not be able to add, modify, or remove principals or permissions in IAM.
- Your credentials might not reflect changes to your permissions that you recently applied in IAM.
For more information, see [Changes that I make are not always immediately visible \(p. 1172\)](#).

AWS Management Console

- You might be able to use a Regional sign-in endpoint to sign in to the AWS Management Console as an IAM userIAM. Regional sign-in endpoints have the following URL format.

`https://{Account ID}.signin.aws.amazon.com/console?region={Region}`

Example: `https://111122223333.signin.aws.amazon.com/console?region=us-west-2`

- You might not be able to complete [Universal 2nd Factor \(U2F\)](#) multi-factor authentication (MFA).

Best practices for IAM resilience

AWS has built resilience into AWS Regions and Availability Zones. When you observe the following IAM best practices in the systems that interact with your environment, you take advantage of that resilience.

1. Use an AWS STS Regional [service endpoint](#) instead of the default global endpoint.
2. Review the configuration of your environment for vital resources that routinely create or modify IAM resources, and prepare a fallback solution that uses existing IAM resources.

Infrastructure security in AWS Identity and Access Management

As a managed service, AWS Identity and Access Management is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see [AWS Cloud Security](#). To design your AWS environment using the best practices for infrastructure security, see [Infrastructure Protection in Security Pillar AWS Well-Architected Framework](#).

You use AWS published API calls to access IAM through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). 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 the [AWS Security Token Service](#) (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 \(p. 1029\)](#)
- [Security best practices and use cases in AWS Identity and Access Management \(p. 1032\)](#)

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 \(p. 1032\)](#)
- [Business use cases for IAM \(p. 1037\)](#)

Security best practices in IAM

 [Follow us on Twitter](#)

The AWS Identity and Access Management best practices were updated on July 14, 2022.

To help secure your AWS resources, follow these best practices for AWS Identity and Access Management (IAM).

Topics

- [Require human users to use federation with an identity provider to access AWS using temporary credentials \(p. 1033\)](#)
- [Require workloads to use temporary credentials with IAM roles to access AWS \(p. 1033\)](#)
- [Require multi-factor authentication \(MFA\) \(p. 1034\)](#)
- [Rotate access keys regularly for use cases that require long-term credentials \(p. 1034\)](#)
- [Safeguard your root user credentials and don't use them for everyday tasks \(p. 1035\)](#)
- [Apply least-privilege permissions \(p. 1035\)](#)
- [Get started with AWS managed policies and move toward least-privilege permissions \(p. 1035\)](#)
- [Use IAM Access Analyzer to generate least-privilege policies based on access activity \(p. 1035\)](#)
- [Regularly review and remove unused users, roles, permissions, policies, and credentials \(p. 1035\)](#)
- [Use conditions in IAM policies to further restrict access \(p. 1036\)](#)
- [Verify public and cross-account access to resources with IAM Access Analyzer \(p. 1036\)](#)
- [Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions \(p. 1036\)](#)
- [Establish permissions guardrails across multiple accounts \(p. 1036\)](#)
- [Use permissions boundaries to delegate permissions management within an account \(p. 1036\)](#)

Require human users to use federation with an identity provider to access AWS using temporary credentials

Human users, also known as *human identities*, are the people, administrators, developers, operators, and consumers of your applications. They must have an identity to access your AWS environments and applications. Human users that are members of your organization are also known as *workforce identities*. Human users can also be external users with whom you collaborate, and who interact with your AWS resources. They can do this via a web browser, client application, mobile app, or interactive command-line tools.

Require your human users to use temporary credentials when accessing AWS. You can use an identity provider for your human users to provide federated access to AWS accounts by assuming roles, which provide temporary credentials. For centralized access management, we recommend that you use [AWS IAM Identity Center \(successor to AWS Single Sign-On\) \(IAM Identity Center\)](#) to manage access to your accounts and permissions within those accounts. You can manage your user identities with IAM Identity Center, or manage access permissions for user identities in IAM Identity Center from an external identity provider. For more information, see [What is AWS IAM Identity Center \(successor to AWS Single Sign-On\)](#) in the [AWS IAM Identity Center \(successor to AWS Single Sign-On\) User Guide](#).

For more information about roles, see [Roles terms and concepts \(p. 184\)](#).

Require workloads to use temporary credentials with IAM roles to access AWS

A *workload* is a collection of resources and code that delivers business value, such as an application or backend process. Your workload can have applications, operational tools, and components that require an identity to make requests to AWS services, such as requests to read data. These identities include machines running in your AWS environments, such as Amazon EC2 instances or AWS Lambda functions.

You can also manage *machine identities* for external parties who need access. To give access to machine identities, you can use IAM roles. IAM roles have specific permissions and provide a way to access AWS by relying on temporary security credentials with a role session. Additionally, you might have machines outside of AWS that need access to your AWS environments. For machines that run outside of AWS you

can use [AWS Identity and Access Management Roles Anywhere](#). For more information about roles, see [IAM roles \(p. 183\)](#). For details about how to use roles to delegate access across AWS accounts, see [IAM tutorial: Delegate access across AWS accounts using IAM roles \(p. 40\)](#).

Require multi-factor authentication (MFA)

We recommend using IAM roles for human users and workloads that access your AWS resources so that they use temporary credentials. However, for scenarios in which you need an IAM user or root user in your account, require MFA for additional security. With MFA, users have a device that generates a response to an authentication challenge. Each user's credentials and device-generated response are required to complete the sign-in process. For more information, see [Using multi-factor authentication \(MFA\) in AWS \(p. 114\)](#).

If you use IAM Identity Center for centralized access management for human users, you can use the IAM Identity Center MFA capabilities when your identity source is configured with the IAM Identity Center identity store, AWS Managed Microsoft AD, or AD Connector. For more information about MFA in IAM Identity Center see [Multi-factor authentication](#) in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*.

Rotate access keys regularly for use cases that require long-term credentials

Where possible, we recommend relying on temporary credentials instead of creating long-term credentials such as access keys. However, for scenarios in which you need IAM users with programmatic access and long-term credentials, we recommend that you rotate access keys. Regularly rotating long-term credentials helps you familiarize yourself with the process. This is useful in case you are ever in a situation where you must rotate credentials, such as when an employee leaves your company. We recommend that you use *IAM access last used information* to rotate and remove access keys safely. For more information, see [Rotating access keys \(p. 108\)](#).

There are specific use cases that require long-term credentials with IAM users in AWS. Some of the use cases include the following:

- **Workloads that cannot use IAM roles** – You might run a workload from a location that needs to access AWS. In some situations, you can't use IAM roles to provide temporary credentials, such as for WordPress plugins. In these situations, use IAM user long-term access keys for that workload to authenticate to AWS.
- **Third-party AWS clients** – If you are using tools that don't support access with IAM Identity Center, such as third-party AWS clients or vendors that are not hosted on AWS, use IAM user long-term access keys.
- **AWS CodeCommit access** – If you are using CodeCommit to store your code, you can use an IAM user with either SSH keys or service-specific credentials for CodeCommit to authenticate to your repositories. We recommend that you do this in addition to using a user in IAM Identity Center for normal authentication. Users in IAM Identity Center are the people in your workforce who need access to your AWS accounts or to your cloud applications. To give users access to your CodeCommit repositories without configuring IAM users, you can configure the **git-remote-codecommit** utility. For more information about IAM and CodeCommit, see [Using IAM with CodeCommit: Git credentials, SSH keys, and AWS access keys \(p. 168\)](#). For more information about configuring the **git-remote-codecommit** utility, see [Connecting to AWS CodeCommit repositories with rotating credentials](#) in the *AWS CodeCommit User Guide*.
- **Amazon Keyspaces (for Apache Cassandra) access** – In a situation where you are unable to use users in IAM Identity Center, such as for testing purposes for Cassandra compatibility, you can use an IAM user with service-specific credentials to authenticate with Amazon Keyspaces. Users in IAM Identity Center are the people in your workforce who need access to your AWS accounts or to your cloud applications. You can also connect to Amazon Keyspaces using temporary credentials. For more

information, see [Using temporary credentials to connect to Amazon Keyspaces using an IAM role and the SigV4 plugin](#) in the *Amazon Keyspaces (for Apache Cassandra) Developer Guide*.

Safeguard your root user credentials and don't use them for everyday tasks

When you create an AWS account you establish a root user name and password to sign in to the AWS Management Console. Safeguard your root user credentials the same way you would protect other sensitive personal information. You can do this by configuring MFA for your root user credentials. We don't recommend generating access keys for your root user, because they allow full access to all your resources for all AWS services, including your billing information. Don't use your root user for everyday tasks. Use the root user to complete the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see [Tasks that require root user credentials](#) in the *AWS Account Management Reference Guide*.

For additional best practices, see [Best practices to protect your account's root user](#) in the *AWS Account Management User Guide*.

Apply least-privilege permissions

When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. You might start with broad permissions while you explore the permissions that are required for your workload or use case. As your use case matures, you can work to reduce the permissions that you grant to work toward least privilege. For more information about using IAM to apply permissions, see [Policies and permissions in IAM \(p. 485\)](#).

Get started with AWS managed policies and move toward least-privilege permissions

To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. Keep in mind that AWS managed policies might not grant least-privilege permissions for your specific use cases because they are available for use by all AWS customers. As a result, we recommend that you reduce permissions further by defining [customer managed policies](#) that are specific to your use cases. For more information, see [AWS managed policies \(p. 495\)](#). For more information about AWS managed policies that are designed for specific job functions, see [AWS managed policies for job functions \(p. 1329\)](#).

Use IAM Access Analyzer to generate least-privilege policies based on access activity

To grant only the permissions required to perform a task, you can generate policies based on your access activity that is logged in AWS CloudTrail. [IAM Access Analyzer](#) analyzes the services and actions that your IAM roles use, and then generates a fine-grained policy that you can use. After you test each generated policy, you can deploy the policy to your production environment. This ensures that you grant only the required permissions to your workloads. For more information about policy generation, see [IAM Access Analyzer policy generation](#).

Regularly review and remove unused users, roles, permissions, policies, and credentials

You might have IAM users, roles, permissions, policies, or credentials that you no longer need in your AWS account. IAM provides *last accessed information* to help you identify the users, roles, permissions,

policies, and credentials that you no longer need so that you can remove them. This helps you reduce the number of users, roles, permissions, policies, and credentials that you have to monitor. You can also use this information to refine your IAM policies to better adhere to least-privilege permissions. For more information, see [Refining permissions in AWS using last accessed information \(p. 616\)](#).

Use conditions in IAM policies to further restrict access

You can specify conditions under which a policy statement is in effect. That way, you can grant access to actions and resources, but only if the access request meets specific conditions. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions, but only if they are used through a specific AWS service, such as AWS CloudFormation. For more information, see [IAM JSON policy elements: Condition \(p. 1278\)](#).

Verify public and cross-account access to resources with IAM Access Analyzer

Before you grant permissions for public or cross-account access in AWS, we recommend that you verify if such access is required. You can use IAM Access Analyzer to help you preview and analyze public and cross-account access for supported resource types. You do this by reviewing the [findings](#) that IAM Access Analyzer generates. These findings help you verify that your resource access controls grant the access that you expect. Additionally, as you update public and cross-account permissions, you can verify the effect of your changes before deploying new access controls to your resources. IAM Access Analyzer also monitors supported resource types continuously and generates a finding for resources that allow public or cross-account access. For more information, see [Previewing access with IAM Access Analyzer APIs](#).

Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions

Validate the policies you create to ensure that they adhere to the [IAM policy language \(p. 490\)](#) (JSON) and IAM best practices. You can validate your policies by using IAM Access Analyzer policy validation. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. As you author new policies or edit existing policies in the console, IAM Access Analyzer provides recommendations to help you refine and validate your policies before you save them. Additionally, we recommend that you review and validate all of your existing policies. For more information, see [IAM Access Analyzer policy validation](#). For more information about policy checks provided by IAM Access Analyzer, see [IAM Access Analyzer policy check reference](#).

Establish permissions guardrails across multiple accounts

As you scale your workloads, separate them by using multiple accounts that are managed with AWS Organizations. We recommend that you use Organizations [service control policies](#) (SCPs) to establish permissions guardrails to control access for all IAM users and roles across your accounts. SCPs are a type of organization policy that you can use to manage permissions in your organization at the AWS organization, OU, or account level. The permissions guardrails that you establish apply to all users and roles within the covered accounts. However, SCPs alone are insufficient to grant permissions to the accounts in your organization. To do this, your administrator must attach [identity-based or resource-based policies \(p. 511\)](#) to IAM users, IAM roles, or the resources in your accounts. For more information, see [AWS Organizations, accounts, and IAM guardrails](#).

Use permissions boundaries to delegate permissions management within an account

In some scenarios, you might want to delegate permissions management within an account to others. For example, you could allow developers to create and manage roles for their workloads. When you

delegate permissions to others, use *permissions boundaries* to set the maximum permissions that you delegate. 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 role. A permissions boundary does not grant permissions on its own. For more information, see [Permissions boundaries for IAM entities \(p. 501\)](#).

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 \(p. 1224\)](#).

Topics

- [Initial setup of example corp \(p. 1037\)](#)
- [Use case for IAM with Amazon EC2 \(p. 1038\)](#)
- [Use case for IAM with Amazon S3 \(p. 1038\)](#)

Initial setup of example corp

Nikki Wolf and Mateo Jackson are the founders of Example Corp. Upon starting the company, they create an AWS account and set up AWS IAM Identity Center (successor to AWS Single Sign-On) (IAM Identity Center) to create administrative accounts to use with their AWS resources. When you set up account access for the administrative user, IAM Identity Center creates a corresponding IAM role. This role, which is controlled by IAM Identity Center, is created in the relevant AWS account, and the policies specified in the **AdministratorAccess** permission set are attached to the role.

Since they now have administrator accounts, Nikki and Mateo no longer need to use their root user to access their AWS account. They plan to only use the root user to complete the tasks that only the root user can perform. After reviewing the security best practices they configure multi-factor authentication (MFA) for their root user credentials and decide how to safeguard their root user credentials.

As their company grows, they hire employees to work as developers, admins, testers, managers, and system administrators. Nikki is in charge of operations, while Mateo manages the engineering teams. They set up an Active Directory Domain Server to manage the employees accounts and manage access to internal company resources.

To give their employees access to AWS resources, they use IAM Identity Center to connect their company's Active Directory to their AWS account.

Because they connected Active Directory to IAM Identity Center, the users, group, and group membership are synchronized and defined. They must assign permission sets and roles to the different groups to give the users the correct level of access to AWS resources. They use [AWS managed policies for job functions \(p. 1329\)](#) in the AWS Management Console to create these permissions sets:

- *Administrator*
- *Billing*
- *Developers*
- *Network administrators*

- *Database administrators*
- *System administrators*
- *Support users*

Then they assign these permissions sets to the roles assigned to their Active Directory groups.

For a step-by-step guide describing the initial configuration of IAM Identity Center, see [Getting started](#) in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*. For more information about provisioning IAM Identity Center user access, see [Single sign-on access to AWS accounts](#) in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*.

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 user groups

To provide "perimeter" control, Nikki attaches a policy to the AllUsers user 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 user groups require different permissions:

- **System administrators** – Need permission to create and manage AMIs, instances, snapshots, volumes, security groups, and so on. Nikki attaches the AmazonEC2FullAccess AWS managed policy to the SysAdmins user group that gives members of the group permission to use all the Amazon EC2 actions.
- **Developers** – Need the ability to work with instances only. Mateo therefore creates and attaches a policy to the Developers user 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.

- **Support users** – Should not be able to perform any Amazon EC2 actions except listing the Amazon EC2 resources currently available. Therefore, Nikki creates and attaches a policy to the Support users 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 \(p. 529\)](#) 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 Santos, changes job functions and becomes a manager. As a manager, Paulo becomes part of the Support users group so that he can open support cases for his developers. Mateo moves Paulo from the Developers user group to the Support users group. As a result of this move, his 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 user groups

As employees, Zhang Wei and Mary Major 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, Mateo 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
    /major
  /share
    /developers
    /managers
```

Mateo 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 Mateo creates a set of policies to assign permissions to the users and user groups:

- **Home directory access for Zhang** – Mateo attaches a policy to Wei that lets him read, write, and list any objects with the Amazon S3 key prefix /aws-s3-bucket/home/zhang/
- **Home directory access for Major** – Mateo attaches a policy to Mary that lets her read, write, and list any objects with the Amazon S3 key prefix /aws-s3-bucket/home/major/
- **Shared directory access for the developers user group** – Mateo attaches a policy to the user group that lets developers read, write, and list any objects in /aws-s3-bucket/share/developers/
- **Shared directory access for the managers user group** – Mateo attaches a policy to the user 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 User Guide*. For information on how policies are evaluated at runtime, see [Policy evaluation logic \(p. 1306\)](#).

User's job function change

At some point, one of the developers, Zhang Wei, changes job functions and becomes a manager. We assume that he no longer needs access to the documents in the share/developers directory. Mateo, as an admin, moves Wei to the Managers user group and out of the Developers user group. With just that simple reassignment, Wei automatically gets all permissions granted to the Managers user 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 Rodriguez needs to put data into a bucket for Example Corp's use. Nikki creates a user group called *WidgetCo* and a user named Shirley and adds Shirley to the *WidgetCo* user group. Nikki also creates a special bucket called *aws-s3-bucket1* for Shirley to use.

Nikki updates existing policies or adds new ones to accommodate the partner Widget Company. For example, Nikki can create a new policy that denies members of the *WidgetCo* user 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.

AWS managed policies for AWS Identity and Access Management Access Analyzer

To add permissions to users, groups, and roles, it is easier to use AWS managed policies than to write policies yourself. It takes time and expertise to [create IAM customer managed policies](#) that provide your team with only the permissions they need. To get started quickly, you can use our AWS managed policies. These policies cover common use cases and are available in your AWS account. For more information about AWS managed policies, see [AWS managed policies](#) in the *IAM User Guide*.

AWS services maintain and update AWS managed policies. You can't change the permissions in AWS managed policies. Services occasionally add additional permissions to an AWS managed policy to support new features. This type of update affects all identities (users, groups, and roles) where the policy is attached. Services are most likely to update an AWS managed policy when a new feature is launched or when new operations become available. Services do not remove permissions from an AWS managed policy, so policy updates won't break your existing permissions.

Additionally, AWS supports managed policies for job functions that span multiple services. For example, the `ReadOnlyAccess` AWS managed policy provides read-only access to all AWS services and resources. When a service launches a new feature, AWS adds read-only permissions for new operations and resources. For a list and descriptions of job function policies, see [AWS managed policies for job functions](#) in the *IAM User Guide*.

IAMReadOnlyAccess

Use the `IAMReadOnlyAccess` managed policy to allow read only access to IAM resources. This policy grants permission to get and list all IAM resources. It allows viewing details and activity reports for users, groups, roles, policies, identity providers, and MFA devices. It does not include the ability to create or delete resources or access to IAM Access Analyzer resources. View the [policy](#) for the full list of services and actions supported by this policy.

IAMUserChangePassword

Use the `IAMUserChangePassword` managed policy to allow IAM users to change their password.

You configure your **IAM Account settings** and the **Password policy** to allow IAM users to change their IAM account password. When you allow this action, IAM attaches the following policy to each user:

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

```
        "arn:aws:iam::*:user/${aws:username}"
    ],
},
{
    "Effect": "Allow",
    "Action": [
        "iam:GetAccountPasswordPolicy"
    ],
    "Resource": "*"
}
]
```

IAMAccessAnalyzerFullAccess

Use the `IAMAccessAnalyzerFullAccess` AWS managed policy to allow your administrators to access IAM Access Analyzer.

Permissions groupings

This policy is grouped into statements based on the set of permissions provided.

- **IAM Access Analyzer** – Allows full administrative permissions to all resources in IAM Access Analyzer.
- **Create service linked role** – Allows the administrator to create a [service-linked role](#), which allows IAM Access Analyzer to analyze resources in other services on your behalf. This permission allows creating the service-linked role only for use by IAM Access Analyzer.
- **AWS Organizations** – Allows administrators to use IAM Access Analyzer for an organization in AWS Organizations. After [enabling trusted access](#) for IAM Access Analyzer in AWS Organizations, members of the management account can view findings across their organization.

```
{
    "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>ListChildren"
            ]
        }
    ]
}
```

```
        "organizations>ListDelegatedAdministrators",
        "organizations>ListOrganizationalUnitsForParent",
        "organizations>ListParents",
        "organizations>ListRoots"
    ],
    "Resource": "*"
}
]
```

IAMAccessAnalyzerReadOnlyAccess

Use the `IAMAccessAnalyzerReadOnlyAccess` AWS managed policy to allow read-only access to IAM Access Analyzer.

To also allow read-only access to IAM Access Analyzer for AWS Organizations, create a customer managed policy that allows the `Describe` and `List` actions from the [IAMAccessAnalyzerFullAccess \(p. 1041\)](#) AWS managed policy.

Service-level permissions

This policy provides read-only access to IAM Access Analyzer. No other service permissions are included in this policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "access-analyzer:Get*",
                "access-analyzer>List*",
                "access-analyzer:ValidatePolicy"
            ],
            "Resource": "*"
        }
    ]
}
```

AccessAnalyzerServiceRolePolicy

You can't attach `AccessAnalyzerServiceRolePolicy` to your IAM entities. This policy is attached to a service-linked role that allows IAM Access Analyzer to perform actions on your behalf. For more information, see [Using service-linked roles for AWS Identity and Access Management Access Analyzer](#).

Permissions groupings

This policy allows access to IAM Access Analyzer to analyze resource metadata from multiple AWS services.

- **Amazon Elastic Compute Cloud** – Allows permissions to describe IP addresses, snapshots, and VPCs.
- **Amazon Elastic Container Registry** – Allows permissions to describe image repositories and retrieve repository policies.
- **Amazon Elastic File System** – Allows permissions to view the description of an Amazon EFS file system and view the resource-level policy for an Amazon EFS file system.
- **AWS Identity and Access Management** – Allows permissions to retrieve information about a specified role and list the IAM roles that have a specified path prefix.

- **AWS Key Management Service** – Allows permissions to view detailed information about an KMS key and its key policies and grants.
- **AWS Lambda** – Allows permissions to view information about Lambda aliases, functions, layers, and aliases.
- **AWS Organizations** – Allows permissions to Organizations and allows the creation of an analyzer within the AWS organization as the zone of trust.
- **Amazon Relational Database Service** – Allows permissions to view detailed information about Amazon RDS DB snapshots and Amazon RDS DB cluster snapshots.
- **Amazon Simple Storage Service** – Allows permissions to view detailed information about Amazon S3 access points and buckets.
- **AWS Secrets Manager** – Allows permissions to view detailed information about secrets and resource policies attached to secrets.
- **Amazon Simple Notification Service** – Allows permissions to view detailed information about a topic.
- **Amazon Simple Queue Service** – Allows permissions to view detailed information about specified queues.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "ec2:DescribeAddresses",  
                "ec2:DescribeByoipCidrs",  
                "ec2:DescribeSnapshotAttribute",  
                "ec2:DescribeSnapshots",  
                "ec2:DescribeVpcEndpoints",  
                "ec2:DescribeVpcs",  
                "ecr:DescribeRepositories",  
                "ecr:GetRepositoryPolicy",  
                "elasticfilesystem:DescribeFileSystemPolicy",  
                "elasticfilesystem:DescribeFileSystems",  
                "iam:GetRole",  
                "iam>ListRoles",  
                "kms:DescribeKey",  
                "kms:GetKeyPolicy",  
                "kms>ListGrants",  
                "kms>ListKeyPolicies",  
                "kms>ListKeys",  
                "lambda:GetFunctionUrlConfig",  
                "lambda:GetLayerVersionPolicy",  
                "lambda:GetPolicy",  
                "lambda>ListAliases",  
                "lambda>ListFunctions",  
                "lambda>ListLayers",  
                "lambda>ListLayerVersions",  
                "lambda>ListVersionsByFunction",  
                "organizations:DescribeAccount",  
                "organizations:DescribeOrganization",  
                "organizations:DescribeOrganizationalUnit",  
                "organizations>ListAccounts",  
                "organizations>ListAccountsForParent",  
                "organizations>ListAWSAccessForOrganization",  
                "organizations>ListChildren",  
                "organizations>ListDelegatedAdministrators",  
                "organizations>ListOrganizationalUnitsForParent",  
                "organizations>ListParents",  
                "organizations>ListRoots",  
                "rds:DescribeDBClusterSnapshotAttributes",  
                "rds:DescribeDBClusterSnapshots",  
            ]  
        }  
    ]  
}
```

```

    "rds:DescribeDBSnapshotAttributes",
    "rds:DescribeDBSnapshots",
    "s3:DescribeMultiRegionAccessPointOperation",
    "s3:GetAccessPoint",
    "s3:GetAccessPointPolicy",
    "s3:GetAccessPointPolicyStatus",
    "s3:GetAccountPublicAccessBlock",
    "s3:GetBucketAcl",
    "s3:GetBucketLocation",
    "s3:GetBucketPolicyStatus",
    "s3:GetBucketPolicy",
    "s3:GetBucketPublicAccessBlock",
    "s3:GetMultiRegionAccessPoint",
    "s3:GetMultiRegionAccessPointPolicy",
    "s3:GetMultiRegionAccessPointPolicyStatus",
    "s3>ListAccessPoints",
    "s3>ListAllMyBuckets",
    "s3>ListMultiRegionAccessPoints",
    "sns:GetTopicAttributes",
    "sns>ListTopics",
    "secretsmanager:DescribeSecret",
    "secretsmanager:GetResourcePolicy",
    "secretsmanager>ListSecrets",
    "sns:GetQueueAttributes",
    "sns>ListQueues"
],
"Resource": "*"
}
]
}

```

IAM and IAM Access Analyzer updates to AWS managed policies

View details about updates to IAM and AWS managed policies since the service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the IAM and IAM Access Analyzer Document history pages.

Change	Description	Date
<u>AccessAnalyzerServiceRolePolicy</u> – Added permissions	IAM Access Analyzer added support for the following resource types to the service-level permissions of <code>AccessAnalyzerServiceRolePolicy</code> : <ul style="list-style-type: none"> • Amazon EBS volume snapshots • Amazon ECR repositories • Amazon EFS file systems • Amazon RDS DB snapshots • Amazon RDS DB cluster snapshots • Amazon SNS topics 	October 25, 2022

Change	Description	Date
AccessAnalyzerServiceRolePolicy – Added permissions	IAM Access Analyzer added the Lambda : GetFunctionUrlConfig action to the service-level permissions of AccessAnalyzerServiceRolePolicy.	April 6, 2022
AccessAnalyzerServiceRolePolicy – Added permissions	IAM Access Analyzer added new Amazon S3 actions to analyze metadata associated with multi-region access points.	September 2, 2021
IAMAccessAnalyzerReadOnlyAccess – Added permissions	IAM Access Analyzer added a new action to grant ValidatePolicy permissions to allow you to use the policy checks for validation. This permission is required by IAM Access Analyzer to perform policy checks on your policies.	March 16, 2021
IAM Access Analyzer started tracking changes	IAM Access Analyzer started tracking changes for its AWS managed policies.	March 1, 2021

Using AWS Identity and Access Management Access Analyzer

AWS IAM Access Analyzer provides the following capabilities:

- IAM Access Analyzer helps [identify resources \(p. 1046\)](#) in your organization and accounts that are shared with an external entity.
- IAM Access Analyzer [validates IAM policies \(p. 1047\)](#) against policy grammar and best practices.
- IAM Access Analyzer [generates IAM policies \(p. 1047\)](#) based on access activity in your AWS CloudTrail logs.

Identifying resources shared with an external entity

IAM Access Analyzer helps you identify the resources in your organization and accounts, such as Amazon S3 buckets or IAM roles, shared with an external entity. This lets you identify unintended access to your resources and data, which is a security risk. IAM Access Analyzer identifies resources 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 shared outside of your account, IAM Access Analyzer generates a finding. Findings include information about the access and the external principal granted to it. You can review findings to determine if the access is intended and safe or if the access is unintended and a security risk. In addition to helping you identify resources shared with an external entity, you can use IAM Access Analyzer findings to preview how your policy affects public and cross-account access to your resource before deploying resource permissions.

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 IAM 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 \(p. 1056\)](#) within your zone of trust. Any access to resources by principals within your zone of trust is considered trusted. Once enabled, IAM Access Analyzer analyzes the policies applied to all of the supported resources in your zone of trust. After the first analysis, IAM Access Analyzer analyzes these policies periodically. If you add a new policy, or change an existing policy, IAM Access Analyzer analyzes the new or updated policy within about 30 minutes.

When analyzing the policies, if IAM 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 with 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, IAM Access Analyzer analyzes the policy. IAM Access Analyzer also analyzes all resource-based policies periodically.

On rare occasions under certain conditions, IAM Access Analyzer does not receive notification of an added or updated policy. IAM Access Analyzer can take up to 6 hours to generate or resolve findings if you create or delete a multi-region access point associated with an S3 bucket, or update the policy for

the multi-region access point. Also, if there is a delivery issue with AWS CloudTrail log delivery, the policy change does not trigger a rescan of the resource reported in the finding. When this happens, IAM Access Analyzer analyzes the new or updated policy during the next periodic scan, which is within 24 hours. If you want to confirm 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 [Findings](#) details page, or by using the [StartResourceScan](#) operation of the IAM Access Analyzer API. To learn more, see [Resolving findings \(p. 1055\)](#).

Important

IAM Access Analyzer analyzes only policies applied to resources in the same AWS Region where it's enabled. To monitor all resources in your AWS environment, you must create an analyzer to enable IAM Access Analyzer in each Region where you're using supported AWS resources.

IAM Access Analyzer analyzes the following resource types:

- [Amazon Simple Storage Service buckets \(p. 1056\)](#)
- [AWS Identity and Access Management roles \(p. 1057\)](#)
- [AWS Key Management Service keys \(p. 1057\)](#)
- [AWS Lambda functions and layers \(p. 1058\)](#)
- [Amazon Simple Queue Service queues \(p. 1058\)](#)
- [AWS Secrets Manager secrets \(p. 1058\)](#)
- [Amazon Simple Notification Service topics \(p. 1059\)](#)
- [Amazon Elastic Block Store volume snapshots \(p. 1059\)](#)
- [Amazon Relational Database Service DB snapshots \(p. 1059\)](#)
- [Amazon Relational Database Service DB cluster snapshots \(p. 1059\)](#)
- [Amazon Elastic Container Registry repositories \(p. 1059\)](#)
- [Amazon Elastic File System file systems \(p. 1060\)](#)

Validating policies

You can validate your policies using IAM Access Analyzer policy checks. You can create or edit a policy using the AWS CLI, AWS API, or JSON policy editor in the IAM console. IAM Access Analyzer validates your policy against [IAM policy grammar \(p. 1323\)](#) and [best practices \(p. 1032\)](#). You can view policy validation check findings that include security warnings, errors, general warnings, and suggestions for your policy. These findings provide actionable recommendations that help you author policies that are functional and conform to security best practices. To learn more about validating policies using IAM Access Analyzer, see [IAM Access Analyzer policy validation \(p. 1080\)](#).

Generating policies

IAM Access Analyzer analyzes your AWS CloudTrail logs to identify actions and services that have been used by an IAM entity (user or role) within your specified date range. It then generates an IAM policy that is based on that access activity. You can use the generated policy to refine an entity's permissions by attaching it to an IAM user or role. To learn more about generating policies using IAM Access Analyzer, see [IAM Access Analyzer policy generation \(p. 1154\)](#).

Findings for public and cross-account access

IAM 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 IAM 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, IAM 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, IAM Access Analyzer generates a finding.

Topics

- [How IAM Access Analyzer findings work \(p. 1048\)](#)
- [Getting started with AWS Identity and Access Management Access Analyzer findings \(p. 1049\)](#)
- [Working with findings \(p. 1051\)](#)
- [Reviewing findings \(p. 1051\)](#)
- [Filtering findings \(p. 1053\)](#)
- [Archiving findings \(p. 1055\)](#)
- [Resolving findings \(p. 1055\)](#)
- [IAM Access Analyzer resource types \(p. 1056\)](#)
- [Settings for IAM Access Analyzer \(p. 1060\)](#)
- [Archive rules \(p. 1061\)](#)
- [Monitoring AWS Identity and Access Management Access Analyzer with Amazon EventBridge \(p. 1062\)](#)
- [Integration with AWS Security Hub \(p. 1067\)](#)
- [Logging IAM Access Analyzer API calls with AWS CloudTrail \(p. 1070\)](#)
- [IAM Access Analyzer filter keys \(p. 1072\)](#)
- [Using service-linked roles for AWS Identity and Access Management Access Analyzer \(p. 1075\)](#)

How IAM Access Analyzer findings work

This topic describes the concepts and terms that are used in IAM Access Analyzer to help you become familiar with how IAM Access Analyzer monitors access to your AWS resources.

AWS Identity and Access Management 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. IAM 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](#).

IAM 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. IAM 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 – IAM Access Analyzer doesn't consider who owns the other account. It is also for security – if the account is not owned by the IAM 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.

IAM Access Analyzer considers only certain IAM condition keys that external users cannot directly influence, or that are otherwise impactful to authorization. For examples of condition keys IAM Access Analyzer considers, see [IAM Access Analyzer filter keys \(p. 1072\)](#).

IAM Access Analyzer does not currently report findings from AWS service principals or internal service accounts. In rare cases where IAM 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. IAM 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 Identity and Access Management Access Analyzer findings

Use the information in this topic to learn about the requirements necessary to use and manage AWS Identity and Access Management Access Analyzer, and then how to enable IAM Access Analyzer. To learn more about the service-linked role for IAM Access Analyzer, see [Using service-linked roles for AWS Identity and Access Management Access Analyzer \(p. 1075\)](#).

Permissions required to use IAM Access Analyzer

To successfully configure and use IAM Access Analyzer, the account you use must be granted the required permissions.

AWS managed policies for IAM Access Analyzer

AWS Identity and Access Management Access Analyzer provides AWS managed policies to help you get started quickly.

- **IAMAccessAnalyzerFullAccess** - Allows full access to IAM Access Analyzer for administrators. This policy also allows creating the service-linked roles that are required to allow IAM Access Analyzer to analyze resources in your account or AWS organization.
- **IAMAccessAnalyzerReadOnlyAccess** - Allows read-only access to IAM Access Analyzer. You must add additional policies to your IAM identities (users, groups of users, or roles) to allow them to view their findings.

Resources defined by AWS Identity and Access Management Access Analyzer

To view the resources defined by IAM Access Analyzer, see [Resource types defined by AWS Identity and Access Management Access Analyzer](#) in the *Service Authorization Reference*.

Required IAM Access Analyzer service permissions

IAM 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 in your account. For more information, see [Using service-linked roles for AWS Identity and Access Management Access Analyzer \(p. 1075\)](#).

Note

IAM Access Analyzer is Regional. You must enable IAM Access Analyzer in each Region independently.

In some cases, after you enable IAM 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, IAM Access Analyzer generates a finding.

Note

It may take up to 30 minutes after a policy is modified for IAM 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 IAM Access Analyzer

To enable IAM 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 IAM 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 \(p. 1060\)](#) 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 IAM 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 IAM 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.

IAM 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 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.

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, IAM 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 IAM 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 IAM Access Analyzer \(p. 1050\)](#), 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 \(p. 1061\)](#) 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 \(p. 1055\)](#).
- 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 attached to the Amazon S3 bucket.
- **Access control list** – The access control list (ACL) attached to the Amazon S3 bucket.
- **Access point** – An access point or multi-region access point 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 the finding was generated if no updates have been made.

Note

It may take up to 30 minutes after a policy is modified for IAM 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 IAM 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 [IAM Access Analyzer filter keys \(p. 1072\)](#).

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 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 the **id** section in [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 the **id** section in [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, IAM 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 IAM Access Analyzer is not able to access a resource, it generates an error finding. If you resolve the issue that prevented IAM 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, IAM Access Analyzer generates a new Active finding.

Note

It may take up to 30 minutes after a policy is modified for IAM 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.

IAM Access Analyzer resource types

IAM Access Analyzer analyzes the resource-based policies that are applied to AWS resources in the Region where you enabled IAM Access Analyzer. It only analyzes resource-based policies. Review the information about each resource for details about how IAM Access Analyzer generates findings for each resource type.

Supported resource types:

- [Amazon Simple Storage Service buckets \(p. 1056\)](#)
- [AWS Identity and Access Management roles \(p. 1057\)](#)
- [AWS Key Management Service keys \(p. 1057\)](#)
- [AWS Lambda functions and layers \(p. 1058\)](#)
- [Amazon Simple Queue Service queues \(p. 1058\)](#)
- [AWS Secrets Manager secrets \(p. 1058\)](#)
- [Amazon Simple Notification Service topics \(p. 1059\)](#)
- [Amazon Elastic Block Store volume snapshots \(p. 1059\)](#)
- [Amazon Relational Database Service DB snapshots \(p. 1059\)](#)
- [Amazon Relational Database Service DB cluster snapshots \(p. 1059\)](#)
- [Amazon Elastic Container Registry repositories \(p. 1059\)](#)
- [Amazon Elastic File System file systems \(p. 1060\)](#)

Amazon Simple Storage Service buckets

When IAM Access Analyzer analyzes Amazon S3 buckets, it generates a finding when an Amazon S3 bucket policy, ACL, or access point, including a multi-Region 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 \(p. 1053\)](#) that isn't within your zone of trust. For example, if a bucket policy grants access to another account or allows public access, IAM 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.

Note

IAM Access Analyzer doesn't analyze the access point policy attached to cross-account access points because the access point and its policy are outside the analyzer account. IAM Access Analyzer generates a public finding when a bucket delegates access to a cross-account access point and Block Public Access is not enabled on the bucket or account. When you enable Block Public Access, the public finding is resolved and IAM Access Analyzer generates a cross-account finding for the cross-account access point.

Amazon S3 *Block Public Access* settings override the bucket policies applied to the bucket. The settings also override the access point policies applied to the bucket's access points. IAM 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 IAM 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, IAM 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, IAM 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. Resolution of public findings for cross-account access points can also take up to 6 hours once you enable Block Public Access at the account level.

For a multi-Region access point, IAM Access Analyzer uses an established policy for generating findings. IAM Access Analyzer evaluates changes to multi-Region access points once every 6 hours. This means IAM Access Analyzer doesn't generate or resolve a finding for up to 6 hours, even if you create or delete a multi-Region access point, or update the policy for it.

AWS Identity and Access Management roles

For IAM roles, IAM 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. 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, IAM Access Analyzer generates a finding in each enabled Region.

AWS Key Management Service keys

For AWS KMS keys, IAM Access Analyzer analyzes the key policies and grants applied to a key. IAM 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), IAM Access Analyzer generates a finding. To learn more about AWS KMS condition keys in IAM policy statements, see [AWS KMS Condition Keys](#).

When IAM 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 IAM 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 IAM 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 IAM 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 key, 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 IAM Access Analyzer permission to access the key.

```
{
  "Sid": "Allow IAM Access Analyzer access to key metadata",
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/aws-service-role/access-
analyzer.amazonaws.com/AWSServiceRoleForAccessAnalyzer"
  },
  "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, IAM Access Analyzer analyzes policies, including condition statements in a policy, that grant access to the function to an external entity. IAM 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, IAM Access Analyzer analyzes policies, including condition statements in a policy, that allow an external entity access to a queue.

AWS Secrets Manager secrets

For AWS Secrets Manager secrets, IAM Access Analyzer analyzes policies, including condition statements in a policy, that allow an external entity to access a secret.

Amazon Simple Notification Service topics

IAM Access Analyzer analyzes resource-based policies attached to Amazon SNS topics, including condition statements in the policies that allow external access to a topic. You can allow external accounts to perform Amazon SNS actions such as subscribing to and publishing topics through a resource-based policy. An Amazon SNS topic is externally accessible if principals from an account outside of your zone of trust can perform operations on the topic. When you choose Everyone in your policy when creating an Amazon SNS topic, you make the topic accessible to the public. AddPermission is another way to add a resource-based policy to an Amazon SNS topic that allows external access.

Amazon Elastic Block Store volume snapshots

Amazon Elastic Block Store volume snapshots do not have resource-based policies. A snapshot is shared through Amazon EBS sharing permissions. For Amazon EBS volume snapshots, IAM Access Analyzer analyzes access control lists that allow an external entity access to a snapshot. An Amazon EBS volume snapshot can be shared with external accounts when encrypted. An unencrypted volume snapshot can be shared with external accounts and grant public access. Sharing settings are in the `CreateVolumePermissions` attribute of the snapshot. When customers preview external access of an Amazon EBS snapshot, they can specify the encryption key as an indicator that the snapshot is encrypted, similar to how IAM Access Analyzer preview handles Secrets Manager secrets.

Amazon Relational Database Service DB snapshots

Amazon RDS DB snapshots do not have resource-based policies. A DB snapshot is shared through Amazon RDS database permissions, and only manual DB snapshots can be shared. For Amazon RDS DB snapshots, IAM Access Analyzer analyzes access control lists that allow an external entity access to a snapshot. Unencrypted DB snapshots can be public. Encrypted DB snapshots cannot be shared publicly, but they can be shared with up to 20 other accounts. For more information, see [Creating a DB snapshot](#). IAM Access Analyzer considers the ability to export a database manual snapshot (for example, to an Amazon S3 bucket) as trusted access.

Note

IAM Access Analyzer does not identify public or cross-account access configured directly on the database itself. IAM Access Analyzer only identifies findings for public or cross-account access configured on the Amazon RDS DB snapshot.

Amazon Relational Database Service DB cluster snapshots

Amazon RDS DB cluster snapshots do not have resource-based policies. A snapshot is shared through Amazon RDS DB cluster permissions. For Amazon RDS DB cluster snapshots, IAM Access Analyzer analyzes access control lists that allow an external entity access to a snapshot. Unencrypted cluster snapshots can be public. Encrypted cluster snapshots cannot be shared publicly. Both unencrypted and encrypted cluster snapshots can be shared with up to 20 other accounts. For more information, see [Creating a DB cluster snapshot](#). IAM Access Analyzer considers the ability to export a DB cluster snapshot (for example, to an Amazon S3 bucket) as trusted access.

Note

IAM Access Analyzer findings do not include monitoring of any share of Amazon RDS DB clusters and clones with another AWS account or organization using AWS Resource Access Manager. IAM Access Analyzer only identifies findings for public or cross-account access configured on the Amazon RDS DB cluster snapshot.

Amazon Elastic Container Registry repositories

For Amazon ECR repositories, IAM Access Analyzer analyzes resource-based policies, including condition statements in a policy, that allow an external entity access to a repository (similar to other resource types

like Amazon SNS topics and Amazon EFS file systems). For Amazon ECR repositories, a principal must have permission to `ecr:GetAuthorizationToken` through an identity-based policy to be considered externally available.

Amazon Elastic File System file systems

For Amazon EFS file systems, IAM Access Analyzer analyzes policies, including condition statements in a policy, that allow an external entity access to a file system. An Amazon EFS file system is externally accessible if principals from an account outside of your zone of trust can perform operations on that file system. Access is defined by a file system policy that uses IAM, and by how the file system is mounted. For example, mounting your Amazon EFS file system in another account is considered externally accessible, unless that account is in your organization and you have defined the organization as your zone of trust. If you are mounting the file system from a virtual private cloud with a public subnet, the file system is externally accessible. When you use Amazon EFS with AWS Transfer Family, file system access requests received from a Transfer Family server that is owned by a different account than the file system are blocked if the file system allows public access.

Settings for IAM Access Analyzer

If you're configuring AWS Identity and Access Management Access Analyzer in your AWS Organizations management account, you can add a member account in the organization as the delegated administrator to manage IAM 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 IAM Access Analyzer

The delegated administrator for IAM 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 IAM 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 IAM 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 IAM Access Analyzer and register the member account as a delegated administrator.

1. Enable trusted service access for IAM 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.

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 [IAM Access Analyzer filter keys \(p. 1072\)](#).

Note

When you create or edit an archive rule, IAM 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, IAM 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.

Monitoring AWS Identity and Access Management Access Analyzer with Amazon EventBridge

Use the information in this topic to learn how to monitor IAM Access Analyzer findings and access previews with Amazon EventBridge. EventBridge is the new version of Amazon CloudWatch Events.

Findings events

IAM 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 IAM Access Analyzer.

Access preview events

IAM Access Analyzer sends an event to EventBridge for each access preview and change to its status. This includes an event when the access preview is first created (status Creating), when the access preview is complete (status Completed), or when the access preview creation failed (status Failed). To receive notifications about access previews, you must create an event rule in EventBridge. When you create an event rule, you can 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 completed access preview is received from IAM Access Analyzer.

Event notification frequency

IAM 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. IAM 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`. IAM Access Analyzer also sends events for newly created access previews and access preview status changes to EventBridge.

Example findings events

The following is an example IAM 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. The `id` is the finding ID. The `resources` array is a singleton with the ARN of the analyzer that generated the finding.

```
{  
    "account": "111122223333",  
    "detail": {  
        "accountId": "111122223333",  
        "action": [  
            "s3:GetObject"  
        ],  
        "analyzedAt": "2019-11-21T01:22:22Z",  
        "condition": {},  
        "createdAt": "2019-11-20T04:58:50Z",  
        "id": "22222222-dcba-4444-dcba-333333333333",  
        "isDeleted": false,  
        "isPublic": false,  
        "principal": {  
            "AWS": "999988887777"  
        },  
        "region": "us-west-2",  
        "resource": "arn:aws:s3:::my-bucket",  
        "resourceType": "AWS::S3::Bucket",  
        "status": "ACTIVE",  
        "updatedAt": "2019-11-21T01:14:07Z",  
        "version": "1.0"  
    },  
    "detail-type": "Access Analyzer Finding",  
    "id": "11111111-2222-4444-aaaa-333333333333",  
    "source": "AWS.IAMAccessAnalyzer",  
    "time": "2019-11-21T01:22:22Z",  
    "version": "0.1.0"  
}
```

```

    "region": "us-west-2",
    "resources": [
        "arn:aws:access-analyzer:us-west-2:111122223333:analyzer/MyAnalyzer"
    ],
    "source": "aws.access-analyzer",
    "time": "2019-11-21T01:22:33Z",
    "version": "0"
}

```

IAM Access Analyzer also sends events to EventBridge for error findings. An error finding is a finding generated when IAM Access Analyzer can't analyze the resource. Events for error findings include an `error` attribute as shown in the following example.

```

{
    "account": "111122223333",
    "detail": {
        "accountId": "111122223333",
        "analyzedAt": "2019-11-21T01:22:22Z",
        "createdAt": "2019-11-20T04:58:50Z",
        "error": "ACCESS_DENIED",
        "id": "22222222-dcba-4444-dcba-333333333333",
        "isDeleted": false,
        "region": "us-west-2",
        "resource": "arn:aws:s3:::my-bucket",
        "resourceType": "AWS::S3::Bucket",
        "status": "ACTIVE",
        "updatedAt": "2019-11-21T01:14:07Z",
        "version": "1.0"
    },
    "detail-type": "Access Analyzer Finding",
    "id": "1111111-2222-4444-aaaa-333333333333",
    "region": "us-west-2",
    "resources": [
        "arn:aws:access-analyzer:us-west-2:111122223333:analyzer/MyAnalyzer"
    ],
    "source": "aws.access-analyzer",
    "time": "2019-11-21T01:22:33Z",
    "version": "0"
}

```

Example access preview events

The following example shows data for the first event that is sent to EventBridge when you create an access preview. The `resources` array is a singleton with the ARN of the analyzer that the access preview is associated with. In the `detail` object, the `id` refers to the access preview ID and `configuredResources` refers to the resource for which the access preview was created. The `status` is `Creating` and refers to the access preview status. The `previousStatus` is not specified because the access preview was just created.

```

{
    "account": "111122223333",
    "detail": {
        "accessPreviewId": "aaaabbbb-cccc-dddd-eeee-ffffaaaabb",
        "configuredResources": [
            "arn:aws:s3:::example-bucket"
        ],
        "createdAt": "2020-02-20T00:00:00.00Z",
        "region": "us-west-2",
        "status": "CREATING",
        "version": "1.0"
    },
}

```

```

"detail-type": "Access Preview State Change",
"id": "aaaabbbb-2222-3333-4444-555566667777",
"region": "us-west-2",
"resources": [
    "arn:aws:access-analyzer:us-west-2:111122223333:analyzer/MyAnalyzer"
],
"source": "aws.access-analyzer",
"time": "2020-02-20T00:00:00.00Z",
"version": "0"
}

```

The following example shows data for an event that is sent to EventBridge for an access preview with a status change from Creating to Completed. In the detail object, the id refers to the access preview ID. The status and previousStatus refer to the access preview status, where the previous status was Creating and the current status is Completed.

```

{
    "account": "111122223333",
    "detail": {
        "accessPreviewId": "aaaabbbb-cccc-dddd-eeee-ffffaaaabbbb",
        "configuredResources": [
            "arn:aws:s3:::example-bucket"
        ],
        "createdAt": "2020-02-20T00:00:00.000Z",
        "previousStatus": "CREATING",
        "region": "us-west-2",
        "status": "COMPLETED",
        "version": "1.0"
    },
    "detail-type": "Access Preview State Change",
    "id": "11112222-3333-4444-5555-66667778888",
    "region": "us-west-2",
    "resources": [
        "arn:aws:access-analyzer:us-west-2:111122223333:analyzer/MyAnalyzer"
    ],
    "source": "aws.access-analyzer",
    "time": "2020-02-20T00:00:00.00Z",
    "version": "0"
}

```

The following example shows data for an event that is sent to EventBridge for an access preview with a status change from Creating to Failed. In the detail object, the id refers to the access preview ID. The status and previousStatus refer to the access preview status, where the previous status was Creating and the current status is Failed. The statusReason field provides the reason code indicating that the access preview failed due to an invalid resource configuration.

```

{
    "account": "111122223333",
    "detail": {
        "accessPreviewId": "aaaabbbb-cccc-dddd-eeee-ffffaaaabbbb",
        "configuredResources": [
            "arn:aws:s3:::example-bucket"
        ],
        "createdAt": "2020-02-20T00:00:00.00Z",
        "previousStatus": "CREATING",
        "region": "us-west-2",
        "status": "FAILED",
        "statusReason": {
            "code": "INVALID_CONFIGURATION"
        },
        "version": "1.0"
    },
}

```

```
"detail-type": "Access Preview State Change",
"id": "99998888-7777-6666-5555-444433332222",
"region": "us-west-2",
"resources": [
    "arn:aws:access-analyzer:us-west-2:111122223333:analyzer/MyAnalyzer"
],
"source": "aws.access-analyzer",
"time": "2020-02-20T00:00:00.00Z",
"version": "0"
}
```

Creating an event rule using the console

The following procedure describes how to create an event rule using the console.

1. Open the Amazon EventBridge console at <https://console.aws.amazon.com/events/>.
2. Using the following values, create an EventBridge rule that monitors finding events or access preview events:
 - For **Rule type**, choose **Rule with an event pattern**.
 - For **Event source**, choose **Other**.
 - For **Event pattern**, choose **Custom patterns (JSON editor)**, and paste one of the following event pattern examples into the text area:
 - To create a rule based on a findings event, use the following pattern example:

```
{
  "source": [
    "aws.access-analyzer"
  ],
  "detail-type": [
    "Access Analyzer Finding"
  ]
}
```

- To create a rule based on an access preview event, use the following pattern example:

```
{
  "source": [
    "aws.access-analyzer"
  ],
  "detail-type": [
    "Access Preview State Change"
  ]
}
```

- For **Target types**, choose **AWS service**, and for **Select a target**, choose a target such as an Amazon SNS topic or AWS Lambda function. The target is triggered when an event is received that matches the event pattern defined in the rule.

To learn more about creating rules, see [Creating Amazon EventBridge rules that react to events](#) in the *Amazon EventBridge User Guide*.

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\"]}}"
```

The following example demonstrates how to create a rule that triggers a target action only for access previews with a status from Creating to Completed.

```
aws events put-rule --name TestRule --event-pattern "{\"source\":[\"aws.access-analyzer\"],\"detail-type\":[\"Access Preview State Change\"],\"detail\":{\"status\":[\"COMPLETED\"]}}"
```

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 \(p. 1068\)](#)
 - [Types of findings that Access Analyzer sends \(p. 1068\)](#)
 - [Latency for sending findings \(p. 1068\)](#)
 - [Retrying when Security Hub is not available \(p. 1068\)](#)
 - [Updating existing findings in Security Hub \(p. 1068\)](#)
- [Viewing Access Analyzer findings in Security Hub \(p. 1069\)](#)
 - [Interpreting Access Analyzer finding names in Security Hub \(p. 1069\)](#)
- [Typical finding from Access Analyzer \(p. 1069\)](#)
- [Enabling and configuring the integration \(p. 1070\)](#)

- [How to stop sending findings \(p. 1070\)](#)

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 \(p. 1056\)](#) 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 IAM Access Analyzer API calls with AWS CloudTrail

IAM 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 IAM Access Analyzer. CloudTrail captures all API calls for IAM Access Analyzer as events. The calls captured include calls from the IAM Access Analyzer console and code calls to the IAM 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 IAM 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 IAM 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](#).

IAM Access Analyzer information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in IAM 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 IAM 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 IAM 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 IAM 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 an assumed-role session named `Alice-tempcreds` on "June 14, 2021". The role session was issued by the role named `admin-tempcreds`.

```
{  
  "eventVersion": "1.05",  
  "userIdentity": {  
    "type": "AssumedRole",  
    "principalId": "AROAIKEVSQ6C2EXAMPLE:Alice-tempcreds",  
    "arn": "arn:aws:sts::111122223333:assumed-role/admin-tempcreds/Alice-tempcreds",  
    "accountId": "111122223333",  
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",  
    "sessionContext": {  
      "attributes": {  
        "mfaAuthenticated": "true",  
        "creationDate": "2021-06-14T22:54:20Z"  
      },  
      "sessionIssuer": {  
        "type": "Role",  
        "principalId": "AKIAI44QH8DHBEEXAMPLE",  
        "arn": "arn:aws:iam::111122223333:role/admin-tempcreds",  
        "accountId": "111122223333",  
        "sessionName": "Alice-tempcreds",  
        "creationDate": "2021-06-14T22:54:20Z",  
        "expirationDate": "2021-06-14T22:54:20Z",  
        "duration": 3600,  
        "externalId": null  
      }  
    }  
  }  
}
```

```

        "arn": "arn:aws:iam::111122223333:role/admin-tempcreds",
        "accountId": "111122223333",
        "userName": "admin-tempcreds"
    },
    "webIdFederationData": {},
}
},
"eventTime": "2021-06-14T22:57:36Z",
"eventSource": "access-analyzer.amazonaws.com",
"eventName": "CreateAnalyzer",
"awsRegion": "us-west-2",
"sourceIPAddress": "198.51.100.179",
"userAgent": "aws-sdk-java/1.12.79 Linux/5.4.141-78.230 OpenJDK_64-Bit_Server_VM/25.302-b08 java/1.8.0_302 vendor/Oracle_Corporation cfg/retry-mode/standard",
"requestParameters": {
    "analyzerName": "test",
    "type": "ACCOUNT",
    "clientToken": "11111111-abcd-2222-abcd-222222222222",
    "tags": {
        "tagkey1": "tagvalue1"
    }
},
"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",
"managementEvent": true,
"recipientAccountId": "111122223333",
"eventCategory": "Management"
}
}

```

IAM Access Analyzer filter keys

You can use the filter keys below to define an archive rule [CreateArchiveRule](#), update an archive rule [UpdateArchiveRule](#), retrieve a list of findings [ListFindings](#), or retrieve a list of access preview findings for a resource [ListAccessPreviewFindings](#). There is no difference between using IAM API and AWS CloudFormation for configuring archive rules.

Criterion	Description	Type	Archive rule	List findings	List access preview 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	✓ Yes
resourceType	The type of resource that the external principal has AWS::IAM::Role access to.	String	✓ Yes	✓ Yes	✓ Yes
	AWS::KMS::Key				
	AWS::Lambda::Function				

Criterion	Description	Type	Archive rule	List findings	List access preview findings
AWS::Lambda::LayerVersion AWS::S3::Bucket AWS::SQS::Queue AWS::SecretsManager::Secret					
resourceOwner <small>The 12-digit AWS account ID that owns the resource. To learn more, see AWS account identifiers.</small>		String	✓ Yes	✓ Yes	✓ Yes
isPublic	Indicates whether the finding reports a resource that has a policy that allows public access.	Boolean	✓ Yes	✓ Yes	✓ Yes
status ACTIVE ARCHIVED RESOLVED	The current status of the finding.	String	✗ No	✓ Yes	✓ Yes
error	Indicates the error reported for the finding.	String	✓ Yes	✓ Yes	✓ Yes
principal.AWS <small>The account granted access to the resource in the Principal field of the finding. Enter the 12-digit AWS account ID or the ARN of the external AWS user or role. To learn more, see AWS account identifiers.</small>		String	✓ Yes	✓ Yes	✓ Yes
principal.Federated <small>ARN of the federated identity that has access to the resource in the finding. To learn more, see Identity providers and federation</small>		String	✓ Yes	✓ Yes	✓ Yes
condition.aws:Principal <small>The principal (IAM user, role, or group) indicated as the condition for resource access. To learn more, see AWS global condition context keys.</small>		String	✓ Yes	✓ Yes	✓ Yes
condition.aws:PrincipalOrgID <small>Identifier of the principal indicated as the condition for resource access. To learn more, see AWS global condition context keys.</small>		String	✓ Yes	✓ Yes	✓ Yes

Criterion	Description	Type	Archive rule	List findings	List access preview findings
condition.aws:PrincipalOrgPath	organizational unit (OU) ID indicated as the condition for resource access. To learn more, see AWS global condition context keys .	String	✓ Yes	✓ Yes	✓ Yes
condition.aws:SourceIp	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	✓ Yes
condition.aws:SourceVpc	IP address 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	✓ Yes
condition.aws:UserIdentity	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	✓ Yes
condition.cognitoIdentity.amazonawsIdentityId	Amazon Cognito identity AWS Identity ID 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	✓ Yes
condition.graphqlFacebookApplication	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	✓ Yes
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	✓ Yes

Criterion	Description	Type	Archive rule	List findings	List access preview findings
condition.kms:CallingAccount	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	✓ 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 Amazon application IDs .	String	✓ Yes	✓ Yes	✓ Yes
id	The ID of the finding.	String	✗ No	✓ Yes	✓ Yes
changeType	Provides context on how the access preview finding compares to existing access identified in IAM Access Analyzer.	String	✗ No	✗ No	✓ Yes
existingFindingId	The existing ID of the finding in IAM Access Analyzer, provided only for existing findings in the access preview.	String	✗ No	✗ No	✓ Yes
existingFindingStatus	The existing status of the finding, provided only for existing findings in the access preview.	String	✗ No	✗ No	✓ Yes

Using service-linked roles for AWS Identity and Access Management Access Analyzer

AWS Identity and Access Management Access Analyzer uses an IAM [service-linked role](#). A service-linked role is a unique type of IAM role 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 Identity and Access Management Access Analyzer

AWS Identity and Access Management 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 named [AccessAnalyzerServiceRolePolicy \(p. 1042\)](#) allows Access Analyzer to complete actions on specific resources.

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 IAM Access Analyzer.

Note

IAM Access Analyzer is Regional. You must enable IAM Access Analyzer in each Region independently.

If you delete this service-linked role, IAM 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](#).

Preview access

In addition to helping you identify resources that are shared with an external entity, AWS IAM Access Analyzer also enables you to preview IAM Access Analyzer findings before deploying resource permissions so you can validate that your policy changes grant only intended public and cross-account access to your resource. This helps you start with intended external access to your resources.

You can preview and validate public and cross-account access to your Amazon S3 buckets in the [Amazon S3 console](#). You can also use IAM Access Analyzer APIs to preview public and cross-account access for your Amazon S3 buckets, AWS KMS keys, IAM roles, Amazon SQS queues and Secrets Manager secrets by providing proposed permissions for your resource.

Topics

- [Previewing access in Amazon S3 console \(p. 1077\)](#)
- [Previewing access with IAM Access Analyzer APIs \(p. 1078\)](#)

Previewing access in Amazon S3 console

After you complete your bucket policy in the Amazon S3 console you have the option to preview public and cross-account access to your Amazon S3 bucket. You can validate that your policy changes grant only intended external access before you choose **Save changes**. This optional step enables you to preview AWS Identity and Access Management Access Analyzer findings for your bucket. You can validate whether the policy change introduces new findings or resolves existing findings for external access. You can skip this validation step and save your Amazon S3 bucket policy at any time.

To preview external access to your bucket, you must have an active account analyzer in your bucket's region with the account as the zone of trust. You must also have the permissions required to use IAM Access Analyzer and preview access. For more information on enabling IAM Access Analyzer and permissions required, see [Enabling IAM Access Analyzer \(p. 1050\)](#).

To preview access to your Amazon S3 bucket when you create or edit your bucket policy

1. Once you finish creating or editing your bucket policy, ensure your policy is a valid Amazon S3 bucket policy. The policy ARN must match the bucket ARN and the [policy elements](#) must be valid.
2. Below the policy, under **Preview external access**, choose an active account analyzer, then choose **Preview**. A preview of IAM Access Analyzer findings is generated for your bucket. The preview analyzes the displayed Amazon S3 bucket policy, together with the existing bucket permissions. This includes the bucket and account BPA settings, bucket ACL, the Amazon S3 access points and multi-region access points attached to the bucket, and their policies and BPA settings.

3. When the access preview completes, a preview of IAM Access Analyzer findings is displayed. Each finding reports an instance of a principal outside of the account with access to your bucket after you save the policy. You can validate access to your bucket by reviewing each finding. The finding header provides a summary of the access and you can expand the finding to review the [finding details](#). Finding badges provide context on how saving the bucket policy would change access to the bucket. For example, they help you validate whether the policy change introduces new findings or resolves existing findings for external access:
 - a. **New** – indicates a finding for new external access that the policy would introduce.
 - b. **Resolved** – indicates a finding for existing external access that the policy would remove.
 - c. **Archived** – indicates a finding for new external access that would be automatically archived, based on the archive rules for the analyzer that define when findings should be marked as intended.
 - d. **Existing** – indicates an existing finding for external access that would remain unchanged.
 - e. **Public** – if a finding is for public access to the resource, it will have a **Public** badge, in addition to one of the badges above.
4. If you identify external access you do not intend to introduce or remove, you can revise the policy and then choose **Preview** again until you have achieved the external access you intend. If you have a finding labeled **Public**, we recommend you revise the policy to remove public access before you choose **Save changes**. Previewing access is an optional step and you can choose **Save changes** at any time.

Previewing access with IAM Access Analyzer APIs

You can use [IAM Access Analyzer APIs](#) to preview public and cross-account access for your Amazon S3 buckets, AWS KMS keys, IAM roles, Amazon SQS queues and Secrets Manager secrets. You can preview access by providing proposed permissions for an existing resource you own or a new resource you want to deploy.

To preview external access to your resource, you must have an active account analyzer for the account and region of the resource. You must also have the permissions required to use IAM Access Analyzer and preview access. For more information on enabling IAM Access Analyzer and permissions required, see [Enabling IAM Access Analyzer \(p. 1050\)](#).

To preview access for a resource, you can use the `CreateAccessPreview` operation and provide the analyzer ARN and the access control configuration for the resource. The service returns the unique ID for the access preview, which you can use to check the status of the access preview with the `GetAccessPreview` operation. When the status is `Completed`, you can use the `ListAccessPreviewFindings` operation to retrieve the findings generated for the access preview. The `GetAccessPreview` and `ListAccessPreviewFindings` operations will retrieve access previews and findings created within about 24 hours.

Each finding retrieved contains [finding details](#) describing the access. A preview status of the finding describing whether the finding would be `Active`, `Archived`, or `Resolved` after permissions deployment, and a `changeType`. The `changeType` provides context on how the access preview finding compares to existing access identified in IAM Access Analyzer:

- **New** – the finding is for newly introduced access.
- **Unchanged** – the preview finding is an existing finding that would remain unchanged.
- **Changed** – the preview finding is an existing finding with a change in status.

The status and the `changeType` help you understand how the resource configuration would change existing resource access. If the `changeType` is `Unchanged` or `Changed`, the finding will also contain the existing ID and status of the finding in IAM Access Analyzer. For example, a `Changed` finding with

preview status Resolved and existing status Active indicates the existing Active finding for the resource would become Resolved as a result of the proposed permissions change.

You can use the `ListAccessPreviews` operation to retrieve a list of access previews for the specified analyzer. This operation will retrieve information on access preview created within about one hour.

In general, if the access preview is for an existing resource and you leave a configuration option unspecified, the access preview will use the existing resource configuration by default. If the access preview is for a new resource and you leave a configuration option unspecified, the access preview will use the default value depending on the resource type. For configuration cases for each resource type, see below.

Preview access to your Amazon S3 bucket

To create an access preview for a new Amazon S3 bucket or an existing Amazon S3 bucket that you own, you can propose a bucket configuration by specifying the Amazon S3 bucket policy, bucket ACLs, bucket BPA settings, and Amazon S3 access points, including multi-region access points, attached to the bucket.

Note

Before attempting to create an access preview for a new bucket, we recommend you call the Amazon S3 [HeadBucket](#) operation to check whether the named bucket already exists. This operation is useful to determine if a bucket exists and you have permission to access it.

Bucket policy – If the configuration is for an existing Amazon S3 bucket and you do not specify the Amazon S3 bucket policy, the access preview uses the existing policy attached to the bucket. If the access preview is for a new resource and you do not specify the Amazon S3 bucket policy, the access preview assumes a bucket without a policy. To propose deletion of an existing bucket policy, you can specify an empty string. For more information about supported bucket policy limits, see [Bucket policy examples](#).

Bucket ACL grants – You can propose up to 100 ACL grants per bucket. If the proposed grant configuration is for an existing bucket, the access preview uses the proposed list of grant configurations in place of the existing grants. Otherwise, the access preview uses the existing grants for the bucket.

Bucket access points – The analysis supports up to 100 access points, including multi-region access points, per bucket, including up to ten new access points you can propose per bucket. If the proposed Amazon S3 access point configuration is for an existing bucket, the access preview uses the proposed access point configuration in place of the existing access points. To propose an access point without a policy, you can provide an empty string as the access point policy. For more information about access point policy limits, see [Access points restrictions and limitations](#).

Block public access configuration – If the proposed configuration is for an existing Amazon S3 bucket and you do not specify the configuration, the access preview uses the existing setting. If the proposed configuration is for a new bucket and you do not specify the bucket BPA configuration, the access preview uses `false`. If the proposed configuration is for a new access point or multi-region access point, and you do not specify the access point BPA configuration, the access preview uses `true`.

Preview access to your AWS KMS key

To create an access preview for a new AWS KMS key or an existing AWS KMS key that you own, you can propose a AWS KMS key configuration by specifying the key policy and the AWS KMS grant configuration.

AWS KMS key policy – If the configuration is for an existing key and you do not specify the key policy, the access preview uses the existing policy for the key. If the access preview is for a new resource and you do not specify the key policy, then the access preview uses the default key policy. The proposed key policy cannot be an empty string.

AWS KMS grants – The analysis supports up to 100 KMS grants per configuration*. If the proposed grant configuration is for an existing key, the access preview uses the proposed list of grant

configurations in place of the existing grants. Otherwise, the access preview uses the existing grants for the key.

Preview access to your IAM role

To create an access preview for a new IAM role or an existing IAM role that you own, you can propose an IAM role configuration by specifying the trust policy.

Role trust policy – If the configuration is for a new IAM role, you must specify the trust policy. If the configuration is for an existing IAM role that you own and you do not propose the trust policy, the access preview uses the existing trust policy for the role. The proposed trust policy cannot be an empty string.

Preview access to your Amazon SQS queue

To create an access preview for a new Amazon SQS queue or an existing Amazon SQS queue that you own, you can propose an Amazon SQS queue configuration by specifying the Amazon SQS policy for the queue.

Amazon SQS queue policy – If the configuration is for an existing Amazon SQS queue and you do not specify the Amazon SQS policy, the access preview uses the existing Amazon SQS policy for the queue. If the access preview is for a new resource and you do not specify the policy, the access preview assumes an Amazon SQS queue without a policy. To propose deletion of an existing Amazon SQS queue policy, you can specify an empty string for the Amazon SQS policy.

Preview access to your Secrets Manager secret

To create an access preview for a new Secrets Manager secret or an existing Secrets Manager secret that you own, you can propose a Secrets Manager secret configuration by specifying the secret policy and optional AWS KMS encryption key.

Secret policy – If the configuration is for an existing secret and you do not specify the secret policy, the access preview uses the existing policy for the secret. If the access preview is for a new resource and you do not specify the policy, the access preview assumes a secret without a policy. To propose deletion of an existing policy, you can specify an empty string.

AWS KMS encryption key – If the proposed configuration is for a new secret and you do not specify the AWS KMS key ID, the access preview uses the default KMS key of the AWS account. If you specify an empty string for the AWS KMS key ID, the access preview uses the default KMS key of the AWS account.

IAM Access Analyzer policy validation

You can validate your policies using AWS Identity and Access Management Access Analyzer policy checks. You can create or edit a policy using the AWS CLI, AWS API, or JSON policy editor in the IAM console. IAM Access Analyzer validates your policy against [IAM policy grammar \(p. 1323\)](#) and [best practices \(p. 1032\)](#). You can view policy validation check findings that include security warnings, errors, general warnings, and suggestions for your policy. These findings provide actionable recommendations that help you author policies that are functional and conform to security best practices. To view a list of the warnings, errors, and suggestions that are returned by Access Analyzer, see [Access Analyzer policy check reference \(p. 1082\)](#).

Validating policies in IAM (console)

You can view findings generated by the policy checks when you create or edit a managed policy in the IAM console. You can also view these findings for inline user or role policies. Access Analyzer does not generate these findings for inline group policies.

To view findings generated by policy checks for IAM JSON policies

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. Begin creating or editing a policy using one of the following methods:
 - a. To create a new managed policy, go to the **Policies** page and create a new policy. For more information, see [Creating policies using the JSON editor \(p. 583\)](#).
 - b. To view policy checks for an existing managed policy, go to the **Policies** page, choose the name of a policy, and then choose **Edit policy**. For more information, see [Editing customer managed policies \(console\) \(p. 610\)](#).
 - c. To view policy checks for an inline policy on a user or role, go the **Users** or **Roles** page, choose the name of a user or role, and choose **Edit policy** on the **Permissions** tab. For more information, see [Editing customer managed policies \(console\) \(p. 610\)](#).
3. In the policy editor, choose the **JSON** tab.
4. In the policy validation pane below the policy, choose one or more of the following tabs. The tab names also indicate the number of each finding type for your policy.
 - **Security** – View warnings if your policy allows access that AWS considers a security risk because the access is overly permissive.
 - **Errors** – View errors if your policy includes lines that prevent the policy from functioning.
 - **General** – View warnings if your policy doesn't conform to best practices, but the issues are not security risks.
 - **Suggestions** – View suggestions if AWS recommends improvements that don't impact the permissions of the policy.
5. Review the finding details provided by the IAM Access Analyzer policy check. Each finding indicates the location of the reported issue. To learn more about what causes the issue and how to resolve it, choose the **Learn more** link next to the finding. You can also search for the policy check associated with each finding in the [Access Analyzer policy checks \(p. 1082\)](#) reference page.
6. Update your policy to resolve the findings.

Important

Test new or edited policies thoroughly before implementing them in your production workflow.

7. When you are finished, choose **Review policy**. The [Policy Validator \(p. 588\)](#) reports any syntax errors that are not reported by Access Analyzer.

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 \(p. 1181\)](#).

8. 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.

Validating policies using Access Analyzer (AWS CLI or AWS API)

You can view findings generated by IAM Access Analyzer policy checks from the AWS Command Line Interface (AWS CLI).

To view findings generated by IAM Access Analyzer policy checks (AWS CLI or AWS API)

Use one of the following:

- AWS CLI: [aws accessanalyzer validate-policy](#)
- AWS API: [ValidatePolicy](#)

Access Analyzer policy check reference

You can validate your policies using AWS IAM Access Analyzer policy checks. You can create or edit a policy using the AWS CLI, AWS API, or JSON policy editor in the IAM console. Access Analyzer validates your policy against IAM [policy grammar \(p. 1323\)](#) and [best practices \(p. 1032\)](#). You can view policy validation check findings that include security warnings, errors, general warnings, and suggestions for your policy. These findings provide actionable recommendations that help you author policies that are functional and conform to security best practices. To learn more about validating policies using Access Analyzer, see [IAM Access Analyzer policy validation \(p. 1080\)](#).

Error – ARN account not allowed

In the AWS Management Console, the finding for this check includes the following message:

```
ARN account not allowed: The service {{service}} does not support specifying an account ID in the resource ARN.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The service {{service}} does not support specifying an account ID in the resource ARN."
```

Resolving the error

Remove the account ID from the resource ARN. The resource ARNs for some AWS services do not support specifying an account ID.

For example, Amazon S3 does not support an account ID as a namespace in bucket ARNs. An Amazon S3 bucket name is globally unique, and the namespace is shared by all AWS accounts. To view all of the resource types available in Amazon S3, see [Resource types defined by Amazon S3](#) in the *Service Authorization Reference*.

Related terms

- [Policy resources \(p. 1276\)](#)
- [Account Identifiers](#)
- [Resource ARNs \(p. 1213\)](#)
- [AWS service resources with ARN formats](#)

Error – ARN Region not allowed

In the AWS Management Console, the finding for this check includes the following message:

```
ARN Region not allowed: The service {{service}} does not support specifying a Region in the resource ARN.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The service {{service}} does not support specifying a Region in the resource ARN."
```

Resolving the error

Remove the Region from the resource ARN. The resource ARNs for some AWS services do not support specifying a Region.

For example, IAM is a global service. The Region portion of an IAM resource ARN is always kept blank. IAM resources are global, like an AWS account is today. For example, after you sign in as an IAM user, you can access AWS services in any geographic region.

- [Policy resources \(p. 1276\)](#)
- [Resource ARNs \(p. 1213\)](#)
- [AWS service resources with ARN formats](#)

Error – Data type mismatch

In the AWS Management Console, the finding for this check includes the following message:

```
Data type mismatch: The text does not match the expected JSON data type {{data_type}}.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The text does not match the expected JSON data type {{data_type}}."
```

Resolving the error

Update the text to use the supported data type.

For example, the Version global condition key requires a String data type. If you provide a date or an integer, the data type won't match.

Related terms

- [Global condition keys \(p. 1338\)](#)
- [IAM JSON policy elements: Condition operators \(p. 1281\)](#)

Error – Duplicate keys with different case

In the AWS Management Console, the finding for this check includes the following message:

```
Duplicate keys with different case: The condition key {{key}} appears more than once with different capitalization in the same condition block. Remove the duplicate condition keys.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The condition key {{key}} appears more than once with different capitalization in the same condition block. Remove the duplicate condition keys."
```

Resolving the error

Review the similar condition keys within the same condition block and use the same capitalization for all instances.

A *condition block* is the text within the Condition element of a policy statement. Condition key *names* are not case-sensitive. The case-sensitivity of condition key *values* depends on the condition operator that you use. For more information about case-sensitivity in condition keys, see [IAM JSON policy elements: Condition \(p. 1278\)](#).

Related terms

- [Conditions \(p. 1278\)](#)
- [Condition block \(p. 1280\)](#)
- [Global condition keys \(p. 1338\)](#)
- [AWS service condition keys](#)

Error – Invalid action

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid action: The action {{action}} does not exist. Did you mean {{valid_action}}?
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The action {{action}} does not exist. Did you mean {{valid_action}}?"
```

Resolving the error

The action that you specified is not valid. This can happen if you mis-type the service prefix or the action name. For some common issues, the policy check returns a suggested action.

Related terms

- [Policy actions \(p. 1273\)](#)
- [AWS service actions](#)

AWS managed policies with this error

[AWS managed policies \(p. 495\)](#) enable you to get started with AWS by assigning permissions based on general AWS use cases.

The following AWS managed policies include invalid actions in their policy statements. Invalid actions do not affect the permissions granted by the policy. When using an AWS managed policy as a reference to create your managed policy, AWS recommends that you remove invalid actions from your policy.

- [AmazonEMRFullAccessPolicy_v2](#)
- [CloudWatchSyntheticsFullAccess](#)

Error – Invalid ARN account

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid ARN account: The resource ARN account ID {{account}} is not valid. Provide a 12-digit account ID.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The resource ARN account ID {{account}} is not valid. Provide a 12-digit account ID."
```

Resolving the error

Update the account ID in the resource ARN. Account IDs are 12-digit integers. To learn how to view your account ID, see [Finding your AWS account ID](#).

Related terms

- [Policy resources \(p. 1276\)](#)
- [Account Identifiers](#)
- [Resource ARNs \(p. 1213\)](#)
- [AWS service resources with ARN formats](#)

Error – Invalid ARN prefix

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid ARN prefix: Add the required prefix (arn) to the resource ARN.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Add the required prefix (arn) to the resource ARN."
```

Resolving the error

AWS resource ARNs must include the required arn: prefix.

Related terms

- [Policy resources \(p. 1276\)](#)
- [Resource ARNs \(p. 1213\)](#)
- [AWS service resources with ARN formats](#)

Error – Invalid ARN Region

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid ARN Region: The Region {{region}} is not valid for this resource. Update the resource ARN to include a supported Region.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The Region {{region}} is not valid for this resource. Update the resource ARN to include a supported Region."
```

Resolving the error

The resource type is not supported in the specified Region. For a table of AWS services supported in each Region, see the [Region table](#).

Related terms

- [Policy resources \(p. 1276\)](#)
- [Resource ARNs \(p. 1213\)](#)
- [Region names and codes](#)

Error – Invalid ARN resource

In the AWS Management Console, the finding for this check includes the following message:

Invalid ARN resource: Resource ARN does not match the expected ARN format. Update the resource portion of the ARN.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Resource ARN does not match the expected ARN format. Update the resource portion of the ARN."
```

Resolving the error

The resource ARN must match the specifications for known resource types. To view the expected ARN format for a service, see [Actions, resources, and condition keys for AWS services](#). Choose the name of the service to view its resource types and ARN formats.

Related terms

- [Policy resources \(p. 1276\)](#)
- [Resource ARNs \(p. 1213\)](#)
- [AWS service resources with ARN formats](#)

Error – Invalid ARN service case

In the AWS Management Console, the finding for this check includes the following message:

Invalid ARN service case: Update the service name \${service} in the resource ARN to use all lowercase letters.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Update the service name ${service} in the resource ARN to use all lowercase letters."
```

Resolving the error

The service in the resource ARN must match the specifications (including capitalization) for service prefixes. To view the prefix for a service, see [Actions, resources, and condition keys for AWS services](#). Choose the name of the service and locate its prefix in the first sentence.

Related terms

- [Policy resources \(p. 1276\)](#)
- [Resource ARNs \(p. 1213\)](#)
- [AWS service resources with ARN formats](#)

Error – Invalid condition data type

In the AWS Management Console, the finding for this check includes the following message:

Invalid condition data type: The condition value data types do not match. Use condition values of the same JSON data type.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The condition value data types do not match. Use condition values of the same JSON data type."
```

Resolving the error

The value in the condition key-value pair must match the data type of the condition key and condition operator. To view the condition key data type for a service, see [Actions, resources, and condition keys for AWS services](#). Choose the name of the service to view the condition keys for that service.

For example, the [CurrentTime \(p. 1341\)](#) global condition key supports the Date condition operator. If you provide a string or an integer for the value in the condition block, the data type won't match.

Related terms

- [Conditions \(p. 1278\)](#)
- [Condition block \(p. 1280\)](#)
- [IAM JSON policy elements: Condition operators \(p. 1281\)](#)
- [Global condition keys \(p. 1338\)](#)
- [AWS service condition keys](#)

Error – Invalid condition key format

In the AWS Management Console, the finding for this check includes the following message:

Invalid condition key format: The condition key format is not valid. Use the format service:keyname.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The condition key format is not valid. Use the format service:keyname."
```

Resolving the error

The key in the condition key-value pair must match the specifications for the service. To view the condition keys for a service, see [Actions, resources, and condition keys for AWS services](#). Choose the name of the service to view the condition keys for that service.

Related terms

- [Conditions \(p. 1278\)](#)
- [Global condition keys \(p. 1338\)](#)
- [AWS service condition keys](#)

Error – Invalid condition multiple Boolean

In the AWS Management Console, the finding for this check includes the following message:

Invalid condition multiple Boolean: The condition key does not support multiple Boolean values. Use a single Boolean value.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

"findingDetails": "The condition key does not support multiple Boolean values. Use a single Boolean value."

Resolving the error

The key in the condition key-value pair expects a single Boolean value. When you provide multiple Boolean values, the condition match might not return the results that you expect.

To view the condition keys for a service, see [Actions, resources, and condition keys for AWS services](#). Choose the name of the service to view the condition keys for that service.

- [Conditions \(p. 1278\)](#)
- [Global condition keys \(p. 1338\)](#)
- [AWS service condition keys](#)

Error – Invalid condition operator

In the AWS Management Console, the finding for this check includes the following message:

Invalid condition operator: The condition operator {{operator}} is not valid. Use a valid condition operator.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

"findingDetails": "The condition operator {{operator}} is not valid. Use a valid condition operator."

Resolving the error

Update the condition to use a supported condition operator.

Related terms

- [IAM JSON policy elements: Condition operators \(p. 1281\)](#)
- [Condition element \(p. 1278\)](#)
- [Overview of JSON policies \(p. 490\)](#)

Error – Invalid effect

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid effect: The effect {{effect}} is not valid. Use Allow or Deny.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The effect {{effect}} is not valid. Use Allow or Deny."
```

Resolving the error

Update the Effect element to use a valid effect. Valid values for Effect are **Allow** and **Deny**.

Related terms

- [Effect element \(p. 1263\)](#)
- [Overview of JSON policies \(p. 490\)](#)

Error – Invalid global condition key

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid global condition key: The condition key {{key}} does not exist. Use a valid condition key.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The condition key {{key}} does not exist. Use a valid condition key."
```

Resolving the error

Update the condition key in the condition key-value pair to use a supported global condition key.

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.

Related terms

- [Global condition keys \(p. 1338\)](#)

Error – Invalid partition

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid partition: The resource ARN for the service {{service}} does not support the partition {{partition}}. Use the supported values: {{partitions}}
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The resource ARN for the service {{service}} does not support the partition {{partition}}. Use the supported values: {{partitions}}"
```

Resolving the error

Update the resource ARN to include a supported partition. If you included a supported partition, then the service or resource might not support the partition that you included.

A *partition* is a group of AWS Regions. Each AWS account is scoped to one partition. In Classic Regions, use the aws partition. In China Regions, use aws-cn.

Related terms

- [Amazon Resource Names \(ARNs\) - Partitions](#)

Error – Invalid policy element

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid policy element: The policy element {{element}} is not valid.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The policy element {{element}} is not valid."
```

Resolving the error

Update the policy to include only supported JSON policy elements.

Related terms

- [JSON policy elements \(p. 1260\)](#)

Error – Invalid principal format

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid principal format: The Principal element contents are not valid. Specify a key-value pair in the Principal element.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The Principal element contents are not valid. Specify a key-value pair in the Principal element."
```

Resolving the error

Update the principal to use a supported key-value pair format.

You can specify a principal in a resource-based policy, but not an identity-based policy.

For example, to define access for everyone in an AWS account, use the following principal in your policy:

```
"Principal": { "AWS": "123456789012" }
```

Related terms

- [JSON policy elements: Principal \(p. 1264\)](#)
- [Identity-based policies and resource-based policies \(p. 511\)](#)

Error – Invalid principal key

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid principal key: The principal key {{principal-key}} is not valid.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The principal key {{principal-key}} is not valid."
```

Resolving the error

Update the key in the principal key-value pair to use a supported principal key. The following are supported principal keys:

- AWS
- CanonicalUser
- Federated
- Service

Related terms

- [Principal element \(p. 1264\)](#)

Error – Invalid Region

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid Region: The Region {{region}} is not valid. Update the condition value to a supported Region.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The Region {{region}} is not valid. Update the condition value to a supported Region."
```

Resolving the error

Update the value of the condition key-value pair to include a supported Region. For a table of AWS services supported in each Region, see the [Region table](#).

Related terms

- [Policy resources \(p. 1276\)](#)
- [Resource ARNs \(p. 1213\)](#)
- [Region names and codes](#)

Error – Invalid service

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid service: The service {{service}} does not exist. Use a valid service name.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The service {{service}} does not exist. Use a valid service name."
```

Resolving the error

The service prefix in the action or condition key must match the specifications (including capitalization) for service prefixes. To view the prefix for a service, see [Actions, resources, and condition keys for AWS services](#). Choose the name of the service and locate its prefix in the first sentence.

Related terms

- [Known services and their actions, resources, and condition keys](#)

Error – Invalid service condition key

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid service condition key: The condition key {{key}} does not exist in the service {{service}}. Use a valid condition key.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The condition key {{key}} does not exist in the service {{service}}. Use a valid condition key."
```

Resolving the error

Update the key in the condition key-value pair to use a known condition key for the service. Global condition key names begin with the aws prefix. AWS services can provide service-specific keys that include their service prefix. To view the prefix for a service, see [Actions, resources, and condition keys for AWS services](#).

Related terms

- [Global condition keys \(p. 1338\)](#)
- [Known services and their actions, resources, and condition keys](#)

Error – Invalid service in action

In the AWS Management Console, the finding for this check includes the following message:

Invalid service in action: The service {{service}} specified in the action does not exist.
Did you mean {{service2}}?

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The service {{service}} specified in the action does not exist. Did you  
mean {{service2}}?"
```

Resolving the error

The service prefix in the action must match the specifications (including capitalization) for service prefixes. To view the prefix for a service, see [Actions, resources, and condition keys for AWS services](#). Choose the name of the service and locate its prefix in the first sentence.

Related terms

- [Action element \(p. 1273\)](#)
- [Known services and their actions](#)

Error – Invalid variable for operator

In the AWS Management Console, the finding for this check includes the following message:

Invalid variable for operator: Policy variables can only be used with String and ARN operators.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Policy variables can only be used with String and ARN operators."
```

Resolving the error

You can use policy variables in the Resource element and in string comparisons in the Condition element. Conditions support variables when you use string operators or 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.

Related terms

- [Using policy variables in the Condition element \(p. 1302\)](#)
- [Condition element \(p. 1278\)](#)

Error – Invalid version

In the AWS Management Console, the finding for this check includes the following message:

Invalid version: The version \${version} is not valid. Use one of the following versions:
\${versions}

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The version ${version} is not valid. Use one of the following versions:  
${versions}"
```

Resolving the error

The `Version` policy element specifies the language syntax rules that AWS uses to process a policy. To use all of the available policy features, include the latest `Version` element before the `Statement` element in all of your policies.

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

Related terms

- [Version element \(p. 1261\)](#)

Error – Json syntax error

In the AWS Management Console, the finding for this check includes the following message:

```
Json syntax error: Fix the JSON syntax error at index {{index}} line {{line}} column  
{{column}}.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Fix the JSON syntax error at index {{index}} line {{line}} column  
{{column}}."
```

Resolving the error

Your policy includes a syntax error. Check your JSON syntax.

Related terms

- [JSON validator](#)
- [IAM JSON policy elements reference \(p. 1260\)](#)
- [Overview of JSON policies \(p. 490\)](#)

Error – Json syntax error

In the AWS Management Console, the finding for this check includes the following message:

```
Json syntax error: Fix the JSON syntax error.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Fix the JSON syntax error."
```

Resolving the error

Your policy includes a syntax error. Check your JSON syntax.

Related terms

- [JSON validator](#)
- [IAM JSON policy elements reference \(p. 1260\)](#)
- [Overview of JSON policies \(p. 490\)](#)

Error – Missing action

In the AWS Management Console, the finding for this check includes the following message:

```
Missing action: Add an Action or NotAction element to the policy statement.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Add an Action or NotAction element to the policy statement."
```

Resolving the error

AWS JSON policies must include an Action or NotAction element.

Related terms

- [Action element \(p. 1273\)](#)
- [NotAction element \(p. 1274\)](#)
- [Overview of JSON policies \(p. 490\)](#)

Error – Missing ARN field

In the AWS Management Console, the finding for this check includes the following message:

```
Missing ARN field: Resource ARNs must include at least {{fields}} fields in the following structure: arn:partition:service:region:account:resource
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Resource ARNs must include at least {{fields}} fields in the following structure: arn:partition:service:region:account:resource"
```

Resolving the error

All of the fields in the resource ARN must match the specifications for a known resource type. To view the expected ARN format for a service, see [Actions, resources, and condition keys for AWS services](#). Choose the name of the service to view its resource types and ARN formats.

Related terms

- [Policy resources \(p. 1276\)](#)
- [Resource ARNs \(p. 1213\)](#)
- [AWS service resources with ARN formats](#)

Error – Missing ARN Region

In the AWS Management Console, the finding for this check includes the following message:

```
Missing ARN Region: Add a Region to the {{service}} resource ARN.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Add a Region to the {{service}} resource ARN."
```

Resolving the error

The resource ARNs for most AWS services require that you specify a Region. For a table of AWS services supported in each Region, see the [Region table](#).

Related terms

- [Policy resources \(p. 1276\)](#)
- [Resource ARNs \(p. 1213\)](#)
- [Region names and codes](#)

Error – Missing effect

In the AWS Management Console, the finding for this check includes the following message:

```
Missing effect: Add an Effect element to the policy statement with a value of Allow or Deny.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Add an Effect element to the policy statement with a value of Allow or Deny."
```

Resolving the error

AWS JSON policies must include an Effect element with a value of **Allow** and **Deny**.

Related terms

- [Effect element \(p. 1263\)](#)
- [Overview of JSON policies \(p. 490\)](#)

Error – Missing principal

In the AWS Management Console, the finding for this check includes the following message:

```
Missing principal: Add a Principal element to the policy statement.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Add a Principal element to the policy statement."
```

Resolving the error

Resource-based policies must include a Principal element.

For example, to define access for everyone in an AWS account, use the following principal in your policy:

```
"Principal": { "AWS": "123456789012" }
```

Related terms

- [Principal element \(p. 1264\)](#)
- [Identity-based policies and resource-based policies \(p. 511\)](#)

Error – Missing qualifier

In the AWS Management Console, the finding for this check includes the following message:

```
Missing qualifier: The request context key ${key} has multiple values. Use the ForAllValues or ForAnyValue condition key qualifiers in your policy.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The request context key ${key} has multiple values. Use the ForAllValues or ForAnyValue condition key qualifiers in your policy."
```

Resolving the error

In the Condition element, you build expressions in which you use condition operators like equal or less than to compare a condition in the policy against keys and values in the request context. For requests that include multiple values for a single condition 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. These qualifiers add set-operation functionality to the condition operator so that you can test multiple request values against multiple condition values.

Related terms

- [Multivalued context keys \(p. 1294\)](#)
- [Condition element \(p. 1278\)](#)

AWS managed policies with this error

[AWS managed policies \(p. 495\)](#) enable you to get started with AWS by assigning permissions based on general AWS use cases.

The following AWS managed policies include a missing qualifier for condition keys in their policy statements. When using the AWS managed policy as a reference to create your customer managed policy, AWS recommends that you add the ForAllValues or ForAnyValue condition key qualifiers to your Condition element.

- [AWSGlueConsoleSageMakerNotebookFullAccess](#)

Error – Missing resource

In the AWS Management Console, the finding for this check includes the following message:

```
Missing resource: Add a Resource or NotResource element to the policy statement.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Add a Resource or NotResource element to the policy statement."
```

Resolving the error

Identity-based policies must include a Resource or NotResource element.

Related terms

- [Resource element \(p. 1276\)](#)
- [NotResource element \(p. 1277\)](#)
- [Identity-based policies and resource-based policies \(p. 511\)](#)
- [Overview of JSON policies \(p. 490\)](#)

Error – Missing statement

In the AWS Management Console, the finding for this check includes the following message:

```
Missing statement: Add a statement to the policy
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Add a statement to the policy"
```

Resolving the error

A JSON policy must include a statement.

Related terms

- [JSON policy elements \(p. 1260\)](#)

Error – Null with if exists

In the AWS Management Console, the finding for this check includes the following message:

```
Null with if exists: The Null condition operator cannot be used with the IfExists suffix.  
Update the operator or the suffix.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The Null condition operator cannot be used with the IfExists suffix.  
Update the operator or the suffix."
```

Resolving the error

You can add `IfExists` to the end of any condition operator name except the `Null` condition operator. Use a `Null` condition operator to check if a condition key is present at the time of authorization. Use `...IfExists` 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."

Related terms

- [...IfExists condition operators \(p. 1288\)](#)
- [Null condition operator \(p. 1290\)](#)
- [Condition element \(p. 1278\)](#)

Error – SCP syntax error action wildcard

In the AWS Management Console, the finding for this check includes the following message:

```
SCP syntax error action wildcard: SCP actions can include wildcards (*) only at the end of a string. Update {{action}}.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "SCP actions can include wildcards (*) only at the end of a string. Update {{action}}."
```

Resolving the error

AWS Organizations service control policies (SCPs) support specifying values in the `Action` or `NotAction` elements. However, these values can include wildcards (*) only at the end of the string. This means that you can specify `iam:Get*` but not `iam:*`role.

To specify multiple actions, AWS recommends that you list them individually.

Related terms

- [SCP Action and NotAction elements](#)
- [Strategies for using SCPs](#)
- [AWS Organizations service control policies](#)
- [IAM JSON policy elements: Action \(p. 1273\)](#)

Error – SCP syntax error allow condition

In the AWS Management Console, the finding for this check includes the following message:

```
SCP syntax error allow condition: SCPs do not support the Condition element with effect Allow. Update the element Condition or the effect.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "SCPs do not support the Condition element with effect Allow. Update the element Condition or the effect."
```

Resolving the error

AWS Organizations service control policies (SCPs) support specifying values in the Condition element only when you use "Effect": "Deny".

To allow only a single action, you can deny access to everything except the condition that you specify using the . . .NotEquals version of a condition operator. This negates the comparison made by the operator.

Related terms

- [SCP Condition element](#)
- [Strategies for using SCPs](#)
- [AWS Organizations service control policies](#)
- [Example policy: Denies access to AWS based on the requested Region \(p. 543\)](#)
- [IAM JSON policy elements: Condition operators \(p. 1281\)](#)
- [IAM JSON policy elements: Condition \(p. 1278\)](#)

Error – SCP syntax error allow NotAction

In the AWS Management Console, the finding for this check includes the following message:

SCP syntax error allow NotAction: SCPs do not support NotAction with effect Allow. Update the element NotAction or the effect.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

"findingDetails": "SCPs do not support NotAction with effect Allow. Update the element NotAction or the effect."

Resolving the error

AWS Organizations service control policies (SCPs) do not support using the NotAction element with "Effect": "Allow".

You must rewrite the logic to allow a list of actions, or to deny every action that is not listed.

Related terms

- [SCP Action and NotAction elements](#)
- [Strategies for using SCPs](#)
- [AWS Organizations service control policies](#)
- [IAM JSON policy elements: Action \(p. 1273\)](#)

Error – SCP syntax error allow resource

In the AWS Management Console, the finding for this check includes the following message:

SCP syntax error allow resource: SCPs do not support Resource with effect Allow. Update the element Resource or the effect.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "SCPs do not support Resource with effect Allow. Update the element Resource or the effect."
```

Resolving the error

AWS Organizations service control policies (SCPs) support specifying values in the Resource element only when you use "Effect": "Deny".

You must rewrite the logic to allow all resources, or to deny every resource that is listed.

Related terms

- [SCP Resource element](#)
- [Strategies for using SCPs](#)
- [AWS Organizations service control policies](#)
- [IAM JSON policy elements: Resource \(p. 1276\)](#)

Error – SCP syntax error NotResource

In the AWS Management Console, the finding for this check includes the following message:

```
SCP syntax error NotResource: SCPs do not support the NotResource element. Update the policy to use Resource instead.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "SCPs do not support the NotResource element. Update the policy to use Resource instead."
```

Resolving the error

AWS Organizations service control policies (SCPs) do not support the NotResource element.

You must rewrite the logic to allow all resources, or to deny every resource that is listed.

Related terms

- [SCP Resource element](#)
- [Strategies for using SCPs](#)
- [AWS Organizations service control policies](#)
- [IAM JSON policy elements: Resource \(p. 1276\)](#)

Error – SCP syntax error principal

In the AWS Management Console, the finding for this check includes the following message:

```
SCP syntax error principal: SCPs do not support specifying principals. Remove the Principal or NotPrincipal element.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "SCPs do not support specifying principals. Remove the Principal or NotPrincipal element."
```

Resolving the error

AWS Organizations service control policies (SCPs) do not support the Principal or NotPrincipal elements.

You can specify the Amazon Resource Name (ARN) using the aws:PrincipalArn global condition key in the Condition element.

Related terms

- [SCP syntax](#)
- [Global condition keys for principals \(p. 1347\)](#)

Error – Unique Sids required

In the AWS Management Console, the finding for this check includes the following message:

```
Unique Sids required: Duplicate statement IDs are not supported for this policy type.  
Update the Sid value.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Duplicate statement IDs are not supported for this policy type. Update  
the Sid value."
```

Resolving the error

For some policy types, statement IDs must be unique. The Sid (statement ID) element allows you to enter an optional identifier that you provide for the policy statement. You can assign a statement ID value to each statement in a statement array using the SID element. 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. For example, in IAM, the Sid value must be unique within a JSON policy.

Related terms

- [IAM JSON policy elements: Sid \(p. 1263\)](#)

Error – Unsupported action in policy

In the AWS Management Console, the finding for this check includes the following message:

```
Unsupported action in policy: The action {{action}} is not supported for the resource-based  
policy attached to the resource type {{resourceType}}.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The action {{action}} is not supported for the resource-based policy  
attached to the resource type {{resourceType}}."
```

Resolving the error

Some actions aren't supported in the Action element in the resource-based policy attached to a different resource type. For example, AWS Key Management Service actions aren't supported in Amazon S3 bucket policies. Specify an action that is supported by resource type attached to your resource-based policy.

Related terms

- [JSON policy elements: Action \(p. 1273\)](#)

Error – Unsupported element combination

In the AWS Management Console, the finding for this check includes the following message:

Unsupported element combination: The policy elements \${element1} and \${element2} can not be used in the same statement. Remove one of these elements.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The policy elements ${element1} and ${element2} can not be used in the same statement. Remove one of these elements."
```

Resolving the error

Some combinations of JSON policy elements can't be used together. 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.

Related terms

- [IAM JSON policy elements reference \(p. 1260\)](#)
- [Overview of JSON policies \(p. 490\)](#)

Error – Unsupported global condition key

In the AWS Management Console, the finding for this check includes the following message:

Unsupported global condition key: The condition key aws:ARN is not supported. Use aws:PrincipalArn or aws:SourceArn instead.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The condition key aws:ARN is not supported. Use aws:PrincipalArn or aws:SourceArn instead."
```

Resolving the error

AWS does not support using the specified global condition key. Depending on your use case, you can use the aws:PrincipalArn or aws:SourceArn global condition keys. For example, instead of aws:ARN, use the aws:PrincipalArn to compare the Amazon Resource Name (ARN) of the principal that made the request with the ARN that you specify in the policy. Alternatively, use the aws:SourceArn global

condition key to compare the Amazon Resource Name (ARN) of the resource making a service-to-service request with the ARN that you specify in the policy.

Related terms

- [AWS global condition context keys \(p. 1338\)](#)

Error – Unsupported principal

In the AWS Management Console, the finding for this check includes the following message:

Unsupported principal: The policy type \${policy_type} does not support the Principal element. Remove the Principal element.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

"findingDetails": "The policy type \${policy_type} does not support the Principal element. Remove the Principal element."

Resolving the error

The Principal element specifies 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 key.

Related terms

- [AWS JSON policy elements: Principal \(p. 1264\)](#)
- [Cross account resource access in IAM \(p. 526\)](#)

Error – Unsupported resource ARN in policy

In the AWS Management Console, the finding for this check includes the following message:

Unsupported resource ARN in policy: The resource ARN is not supported for the resource-based policy attached to the resource type {{resourceType}}.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

"findingDetails": "The resource ARN is not supported for the resource-based policy attached to the resource type {{resourceType}}."

Resolving the error

Some resource ARNs aren't supported in the Resource element of the resource-based policy when the policy is attached to a different resource type. For example, AWS KMS ARNs aren't supported in the Resource element for Amazon S3 bucket policies. Specify a resource ARN that is supported by a resource type attached to your resource-based policy.

Related terms

- [JSON policy elements: Action \(p. 1273\)](#)

Error – Unsupported Sid

In the AWS Management Console, the finding for this check includes the following message:

Unsupported Sid: Update the characters in the Sid element to use one of the following character types: [a-z, A-Z, 0-9]

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

"findingDetails": "Update the characters in the Sid element to use one of the following character types: [a-z, A-Z, 0-9]"

Resolving the error

The Sid element supports uppercase letters, lowercase letters, and numbers.

Related terms

- [IAM JSON policy elements: Sid \(p. 1263\)](#)

Error – Unsupported wildcard in principal

In the AWS Management Console, the finding for this check includes the following message:

Unsupported wildcard in principal: Wildcards (*, ?) are not supported with the principal key {{principal_key}}. Replace the wildcard with a valid principal value.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

"findingDetails": "Wildcards (*, ?) are not supported with the principal key {{principal_key}}. Replace the wildcard with a valid principal value."

Resolving the error

The Principal element structure supports using a key-value pair. The principal value specified in the policy includes a wildcard (*). You can't include a wildcard with the principal key that you specified. For example, when you specify users in a Principal element, you cannot use a wildcard to mean "all users". You must name a specific user or users. Similarly, when you specify an assumed-role session, you cannot use a wildcard to mean "all sessions". You must name a specific session. You also cannot use a wildcard to match part of a name or an ARN.

To resolve this finding, remove the wildcard and provide a more specific principal.

Related terms

- [AWS JSON policy elements: Principal \(p. 1264\)](#)

Error – Missing brace in variable

In the AWS Management Console, the finding for this check includes the following message:

Missing brace in variable: The policy variable is missing a closing curly brace. Add } after the variable text.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The policy variable is missing a closing curly brace. Add } after the variable text."
```

Resolving the error

Policy variable structure supports using a \$ prefix followed by a pair of curly braces ({ }). Inside the \${ } characters, include the name of the value from the request that you want to use in the policy.

To resolve this finding, add the missing brace to make sure the full opening and closing set of braces is present.

Related terms

- [IAM policy elements: Variables \(p. 1298\)](#)

Error – Missing quote in variable

In the AWS Management Console, the finding for this check includes the following message:

```
Missing quote in variable: The policy variable default value must begin and end with a single quote. Add the missing quote.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The policy variable default value must begin and end with a single quote. Add the missing quote."
```

Resolving the error

When you add a variable to your policy, you can specify a default value for the variable. If a variable is not present, AWS uses the default text that you provide.

To add a default value to a variable, surround the default value with single quotes (' '), and separate the variable text and the default value with a comma and space (,).

For example, if a principal is tagged with team=yellow, they can access the **DOC-EXAMPLE-BUCKET** Amazon S3 bucket with the name **DOC-EXAMPLE-BUCKET**-yellow. A policy with this resource might allow team members to access their own resources, but not those of other teams. For users without team tags, you might set a default value of company-wide. These users can access only the **DOC-EXAMPLE-BUCKET**-company-wide bucket where they can view broad information, such as instructions for joining a team.

```
"Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET-${aws:PrincipalTag/team, 'company-wide'}"
```

Related terms

- [IAM policy elements: Variables \(p. 1298\)](#)

Error – Unsupported space in variable

In the AWS Management Console, the finding for this check includes the following message:

Unsupported space in variable: A space is not supported within the policy variable text. Remove the space.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "A space is not supported within the policy variable text. Remove the space."
```

Resolving the error

Policy variable structure supports using a \$ prefix followed by a pair of curly braces ({}). Inside the \${ } characters, include the name of the value from the request that you want to use in the policy. Although you can include a space when you specify a default variable, you cannot include a space in the variable name.

Related terms

- [IAM policy elements: Variables \(p. 1298\)](#)

Error – Empty variable

In the AWS Management Console, the finding for this check includes the following message:

Empty variable: Empty policy variable. Remove the \${ } variable structure or provide a variable within the structure.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Empty policy variable. Remove the ${ } variable structure or provide a variable within the structure."
```

Resolving the error

Policy variable structure supports using a \$ prefix followed by a pair of curly braces ({}). Inside the \${ } characters, include the name of the value from the request that you want to use in the policy.

Related terms

- [IAM policy elements: Variables \(p. 1298\)](#)

Error – Variable unsupported in element

In the AWS Management Console, the finding for this check includes the following message:

Variable unsupported in element: Policy variables are supported in the Resource and Condition elements. Remove the policy variable {{variable}} from this element.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Policy variables are supported in the Resource and Condition elements. Remove the policy variable {{variable}} from this element."
```

Resolving the error

You can use policy variables in the Resource element and in string comparisons in the Condition element.

Related terms

- [IAM policy elements: Variables \(p. 1298\)](#)

Error – Variable unsupported in version

In the AWS Management Console, the finding for this check includes the following message:

```
Variable unsupported in version: To include variables in your policy, use the policy version 2012-10-17 or later.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "To include variables in your policy, use the policy version 2012-10-17 or later."
```

Resolving the error

To use policy variables, you must include the Version element and set it 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 set the Version to 2012-10-17 or later, 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, 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.

Related terms

- [IAM policy elements: Variables \(p. 1298\)](#)
- [IAM JSON policy elements: Version \(p. 1261\)](#)

Error – Private IP address

In the AWS Management Console, the finding for this check includes the following message:

```
Private IP address: aws:SourceIp works only for public IP address ranges. The values for condition key aws:SourceIp include only private IP addresses and will not have the desired effect. Update the value to include only public IP addresses.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "aws:SourceIp works only for public IP address ranges. The values for condition key aws:SourceIp include only private IP addresses and will not have the desired effect. Update the value to include only public IP addresses."
```

Resolving the error

The global condition key `aws:SourceIp` works only for public IP address ranges. You receive this error when your policy allows only private IP addresses. In this case, the condition would never match.

- [aws:SourceIp global condition key \(p. 1363\)](#)
- [IAM JSON policy elements: Condition \(p. 1278\)](#)

Error – Private NotIpAddress

In the AWS Management Console, the finding for this check includes the following message:

Private NotIpAddress: The values for condition key `aws:SourceIp` include only private IP addresses and has no effect. `aws:SourceIp` works only for public IP address ranges. Update the value to include only public IP addresses.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

"findingDetails": "The values for condition key `aws:SourceIp` include only private IP addresses and has no effect. `aws:SourceIp` works only for public IP address ranges. Update the value to include only public IP addresses."

Resolving the error

The global condition key `aws:SourceIp` works only for public IP address ranges. You receive this error when you use the `NotIpAddress` condition operator and list only private IP addresses. In this case, the condition would always match and would be ineffective.

- [aws:SourceIp global condition key \(p. 1363\)](#)
- [IAM JSON policy elements: Condition \(p. 1278\)](#)

Error – Policy size exceeds SCP quota

In the AWS Management Console, the finding for this check includes the following message:

Policy size exceeds SCP quota: The `{{policySize}}` characters in the service control policy (SCP) exceed the `{{policySizeQuota}}` character maximum for SCPs. We recommend that you use multiple granular policies.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

"findingDetails": "The `{{policySize}}` characters in the service control policy (SCP) exceed the `{{policySizeQuota}}` character maximum for SCPs. We recommend that you use multiple granular policies."

Resolving the error

AWS Organizations service control policies (SCPs) support specifying values in the `Action` or `NotAction` elements. However, these values can include wildcards (*) only at the end of the string. This means that you can specify `iam:Get*` but not `iam:*`role.

To specify multiple actions, AWS recommends that you list them individually.

Related terms

- [Quotas for AWS Organizations](#)
- [AWS Organizations service control policies](#)

Error – Invalid service principal format

In the AWS Management Console, the finding for this check includes the following message:

Invalid service principal format: The service principal does not match the expected format. Use the format {{expectedFormat}}.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The service principal does not match the expected format. Use the format {{expectedFormat}}."
```

Resolving the error

The value in the condition key-value pair must match a defined service principal format.

A *service principal* is an identifier that is used to grant permissions to a service. You can specify a service principal in the Principal element or as a value for some global condition keys and service-specific keys. The service principal is defined by each service.

The identifier for a service principal includes the service name, and is usually in the following format in all lowercase letters:

service-name.amazonaws.com

Some service-specific keys may use a different format for service principals. For example, the kms:ViaService condition key requires the following format for service principals in all lowercase letters:

service-name.AWS_region.amazonaws.com

Related terms

- [Service principals](#)
- [AWS global condition keys \(p. 1338\)](#)
- [kms:ViaService condition key](#)

Error – Missing tag key in condition

In the AWS Management Console, the finding for this check includes the following message:

Missing tag key in condition: The condition key {{conditionKeyName}} must include a tag key to control access based on tags. Use the format {{conditionKeyName}}tag-key and specify a key name for tag-key.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The condition key {{conditionKeyName}} must include a tag key to control access based on tags. Use the format {{conditionKeyName}}tag-key and specify a key name for tag-key."
```

Resolving the error

To control access based on tags, you provide tag information in the [condition element \(p. 1278\)](#) of a policy.

For example, to [control access to AWS resources](#), you include the aws:ResourceTag condition key. This key requires the format aws:ResourceTag/[tag-key](#). To specify the tag key owner and the tag value JaneDoe in a condition, use the following format.

```
"Condition": {  
    "StringEquals": {"aws:ResourceTag/owner": "JaneDoe"}  
}
```

Related terms

- [Controlling access using tags \(p. 521\)](#)
- [Conditions \(p. 1278\)](#)
- [Global condition keys \(p. 1338\)](#)
- [AWS service condition keys](#)

Error – Invalid vpc format

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid vpc format: The VPC identifier in the condition key value is not valid. Use the prefix 'vpc-' followed by 8 or 17 alphanumeric characters.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The VPC identifier in the condition key value is not valid. Use the prefix 'vpc-' followed by 8 or 17 alphanumeric characters."
```

Resolving the error

The aws:SourceVpc condition key must use the prefix vpc- followed by either 8 or 17 alphanumeric characters, for example, vpc-11223344556677889 or vpc-12345678.

Related terms

- [AWS global condition keys: aws:SourceVpc](#)

Error – Invalid vpce format

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid vpce format: The VPCE identifier in the condition key value is not valid. Use the prefix 'vpce-' followed by 8 or 17 alphanumeric characters.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The VPCE identifier in the condition key value is not valid. Use the prefix 'vpce-' followed by 8 or 17 alphanumeric characters."
```

Resolving the error

The `aws:SourceVpce` condition key must use the prefix `vpce-` followed by either 8 or 17 alphanumeric characters, for example, `vpce-11223344556677889` or `vpce-12345678`.

Related terms

- [AWS global condition keys: aws:SourceVpce](#)

Error – Federated principal not supported

In the AWS Management Console, the finding for this check includes the following message:

Federated principal not supported: The policy type does not support a federated identity provider in the principal element. Use a supported principal.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

"findingDetails": "The policy type does not support a federated identity provider in the principal element. Use a supported principal."

Resolving the error

The `Principal` element uses federated principals for trust policies attached to IAM roles to provide access through identity federation. Identity policies and other resource-based policies don't support a federated identity provider in the `Principal` element. For example, you can't use a SAML principal in an Amazon S3 bucket policy. Change the `Principal` element to a supported principal type.

Related terms

- [Creating a role for identity federation \(p. 260\)](#)
- [JSON policy elements: Principal \(p. 1264\)](#)

Error – Unsupported action for condition key

In the AWS Management Console, the finding for this check includes the following message:

Unsupported action for condition key: The following actions: `{{actions}}` are not supported by the condition key `{{key}}`. The condition will not be evaluated for these actions. We recommend that you move these actions to a different statement without this condition key.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

"findingDetails": "The following actions: `{{actions}}` are not supported by the condition key `{{key}}`. The condition will not be evaluated for these actions. We recommend that you move these actions to a different statement without this condition key."

Resolving the error

Make sure that the condition key in the `Condition` element of the policy statement applies to every action in the `Action` element. To ensure that the actions you specify are effectively allowed or denied by your policy, you should move the unsupported actions to a different statement without the condition key.

Note

If the Action element has actions with wildcards, IAM Access Analyzer doesn't evaluate those actions for this error.

Related terms

- [JSON policy elements: Action \(p. 1273\)](#)

Error – Unsupported action in policy

In the AWS Management Console, the finding for this check includes the following message:

Unsupported action in policy: The action {{action}} is not supported for the resource-based policy attached to the resource type {{resourceType}}.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The action {{action}} is not supported for the resource-based policy attached to the resource type {{resourceType}}."
```

Resolving the error

Some actions aren't supported in the Action element in the resource-based policy attached to a different resource type. For example, AWS Key Management Service actions aren't supported in Amazon S3 bucket policies. Specify an action that is supported by resource type attached to your resource-based policy.

Related terms

- [JSON policy elements: Action \(p. 1273\)](#)

Error – Unsupported resource ARN in policy

In the AWS Management Console, the finding for this check includes the following message:

Unsupported resource ARN in policy: The resource ARN is not supported for the resource-based policy attached to the resource type {{resourceType}}.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The resource ARN is not supported for the resource-based policy attached to the resource type {{resourceType}}."
```

Resolving the error

Some resource ARNs aren't supported in the Resource element of the resource-based policy when the policy is attached to a different resource type. For example, AWS KMS ARNs aren't supported in the Resource element for Amazon S3 bucket policies. Specify a resource ARN that is supported by a resource type attached to your resource-based policy.

Related terms

- [JSON policy elements: Action \(p. 1273\)](#)

Error – Unsupported condition key for service principal

In the AWS Management Console, the finding for this check includes the following message:

Unsupported condition key for service principal: The following condition keys are not supported when used with the service principal: {{conditionKeys}}.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

"findingDetails": "The following condition keys are not supported when used with the service principal: {{conditionKeys}}."

Resolving the error

You can specify AWS services in the Principal element of a resource-based policy using a *service principal*, which is an identifier for the service. You can't use some condition keys with certain service principals. For example, you can't use the aws:PrincipalOrgID condition key with the service principal cloudfront.amazonaws.com. You should remove condition keys that do not apply to the service principal in the Principal element.

Related terms

- [Service principals](#)
- [JSON policy elements: Principal \(p. 1264\)](#)

Error – Role trust policy syntax error notprincipal

In the AWS Management Console, the finding for this check includes the following message:

Role trust policy syntax error notprincipal: Role trust policies do not support NotPrincipal. Update the policy to use a Principal element instead.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

"findingDetails": "Role trust policies do not support NotPrincipal. Update the policy to use a Principal element instead."

Resolving the error

A role trust policy is a resource-based policy that is attached to an IAM role. Trust policies define which principal entities (accounts, users, roles, and federated users) can assume the role. Role trust policies do not support NotPrincipal. Update the policy to use a Principal element instead.

Related terms

- [JSON policy elements: Principal \(p. 1264\)](#)
- [JSON policy elements: NotPrincipal \(p. 1272\)](#)

Error – Role trust policy unsupported wildcard in principal

In the AWS Management Console, the finding for this check includes the following message:

Role trust policy unsupported wildcard in principal: "Principal:" "*" is not supported in the principal element of a role trust policy. Replace the wildcard with a valid principal value.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "'Principal:' '*' is not supported in the principal element of a role trust policy. Replace the wildcard with a valid principal value."
```

Resolving the error

A role trust policy is a resource-based policy that is attached to an IAM role. Trust policies define which principal entities (accounts, users, roles, and federated users) can assume the role. "Principal:" "*" is not supported in the Principal element of a role trust policy. Replace the wildcard with a valid principal value.

Related terms

- [JSON policy elements: Principal \(p. 1264\)](#)

Error – Role trust policy syntax error resource

In the AWS Management Console, the finding for this check includes the following message:

Role trust policy syntax error resource: Role trust policies apply to the role that they are attached to. You cannot specify a resource. Remove the Resource or NotResource element.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Role trust policies apply to the role that they are attached to. You cannot specify a resource. Remove the Resource or NotResource element."
```

Resolving the error

A role trust policy is a resource-based policy that is attached to an IAM role. Trust policies define which principal entities (accounts, users, roles, and federated users) can assume the role. Role trust policies apply to the role that they are attached to. You cannot specify a Resource or NotResource element in a role trust policy. Remove the Resource or NotResource element.

- [JSON policy elements: Resource \(p. 1276\)](#)
- [JSON policy elements: NotResource \(p. 1277\)](#)

Error – Type mismatch IP range

In the AWS Management Console, the finding for this check includes the following message:

Type mismatch IP range: The condition operator {{operator}} is used with an invalid IP range value. Specify the IP range in standard CIDR format.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The condition operator {{operator}} is used with an invalid IP range value. Specify the IP range in standard CIDR format."
```

Resolving the error

Update the text to use the IP address condition operator data type, in a CIDR format.

Related terms

- [IP address condition operators \(p. 1286\)](#)
- [IAM JSON policy elements: Condition operators \(p. 1281\)](#)

Error – Missing action for condition key

In the AWS Management Console, the finding for this check includes the following message:

```
Missing action for condition key: The {{actionName}} action must be in the action block to allow setting values for the condition key {{keyName}}. Add {{actionName}} to the action block.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The {{actionName}} action must be in the action block to allow setting values for the condition key {{keyName}}. Add {{actionName}} to the action block."
```

Resolving the error

The condition key in the Condition element of the policy statement is not evaluated unless the specified action is in the Action element. To ensure that the condition keys you specify are effectively allowed or denied by your policy, add the action to the Action element.

Related terms

- [JSON policy elements: Action \(p. 1273\)](#)

Error – Invalid federated principal syntax in role trust policy

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid federated principal syntax in role trust policy: The principal value specifies a federated principal that does not match the expected format. Update the federated principal to a domain name or a SAML metadata ARN.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The principal value specifies a federated principal that does not match the expected format. Update the federated principal to a domain name or a SAML metadata ARN."
```

Resolving the error

The principal value specifies a federated principal that does not match the expected format. Update the format of the federated principal to a valid domain name or a SAML metadata ARN.

Related terms

- [Federated users and roles](#)

Error – Mismatched action for principal

In the AWS Management Console, the finding for this check includes the following message:

Mismatched action for principal: The {{actionName}} action is invalid with the following principal(s): {{principalNames}}. Use a SAML provider principal with the sts:AssumeRoleWithSAML action or use an OIDC provider principal with the sts:AssumeRoleWithWebIdentity action. Ensure the provider is Federated if you use either of the two options.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The {{actionName}} action is invalid with the following principal(s): {{principalNames}}. Use a SAML provider principal with the sts:AssumeRoleWithSAML action or use an OIDC provider principal with the sts:AssumeRoleWithWebIdentity action. Ensure the provider is Federated if you use either of the two options."
```

Resolving the error

The action specified in the Action element of the policy statement is invalid with the principal specified in the Principal element. For example, you can't use a SAML provider principal with the sts:AssumeRoleWithWebIdentity action. You should use a SAML provider principal with the sts:AssumeRoleWithSAML action or use an OIDC provider principal with the sts:AssumeRoleWithWebIdentity action.

Related terms

- [AssumeRoleWithSAML](#)
- [AssumeRoleWithWebIdentity](#)

Error – Missing action for roles anywhere trust policy

In the AWS Management Console, the finding for this check includes the following message:

Missing action for roles anywhere trust policy: The rolesanywhere.amazonaws.com service principal requires the sts:AssumeRole, sts:SetSourceIdentity, and sts:TagSession permissions to assume a role. Add the missing permissions to the policy.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The rolesanywhere.amazonaws.com service principal requires the sts:AssumeRole, sts:SetSourceIdentity, and sts:TagSession permissions to assume a role. Add the missing permissions to the policy."
```

Resolving the error

For IAM Roles Anywhere to be able to assume a role and deliver temporary AWS credentials, the role must trust the IAM Roles Anywhere service principal. The IAM Roles Anywhere service principal requires

the `sts:AssumeRole`, `sts:SetSourceIdentity`, and `sts:TagSession` permissions to assume a role. If any of the permissions are missing, you must add them to your policy.

Related terms

- [Trust model in AWS Identity and Access Management Roles Anywhere](#)

General Warning – Create SLR with NotResource

In the AWS Management Console, the finding for this check includes the following message:

Create SLR with NotResource: Using the `iam:CreateServiceLinkedRole` action with `NotResource` can allow creation of unintended service-linked roles for multiple resources. We recommend that you specify resource ARNs instead.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

"findingDetails": "Using the `iam:CreateServiceLinkedRole` action with `NotResource` can allow creation of unintended service-linked roles for multiple resources. We recommend that you specify resource ARNs instead."

Resolving the general warning

The action `iam:CreateServiceLinkedRole` grants permission to create an IAM role that allows an AWS service to perform actions on your behalf. Using `iam:CreateServiceLinkedRole` in a policy with the `NotResource` element can allow creating unintended service-linked roles for multiple resources. AWS recommends that you specify allowed ARNs in the `Resource` element instead.

- [CreateServiceLinkedRole operation](#)
- [IAM JSON policy elements: NotResource \(p. 1277\)](#)
- [IAM JSON policy elements: Resource \(p. 1276\)](#)

General Warning – Create SLR with star in action and NotResource

In the AWS Management Console, the finding for this check includes the following message:

Create SLR with star in action and NotResource: Using an action with a wildcard(*) and `NotResource` can allow creation of unintended service-linked roles because it can allow `iam:CreateServiceLinkedRole` permissions on multiple resources. We recommend that you specify resource ARNs instead.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

"findingDetails": "Using an action with a wildcard(*) and `NotResource` can allow creation of unintended service-linked roles because it can allow `iam:CreateServiceLinkedRole` permissions on multiple resources. We recommend that you specify resource ARNs instead."

Resolving the general warning

The action `iam:CreateServiceLinkedRole` grants permission to create an IAM role that allows an AWS service to perform actions on your behalf. Policies with a wildcard (*) in the Action and that

include the `NotResource` element can allow creation of unintended service-linked roles for multiple resources. AWS recommends that you specify allowed ARNs in the `Resource` element instead.

- [CreateServiceLinkedRole operation](#)
- [IAM JSON policy elements: NotResource \(p. 1277\)](#)
- [IAM JSON policy elements: Resource \(p. 1276\)](#)

General Warning – Create SLR with NotAction and NotResource

In the AWS Management Console, the finding for this check includes the following message:

Create SLR with NotAction and NotResource: Using `NotAction` with `NotResource` can allow creation of unintended service-linked roles because it allows `iam:CreateServiceLinkedRole` permissions on multiple resources. We recommend that you specify resource ARNs instead.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Using NotAction with NotResource can allow creation of unintended service-linked roles because it allows iam:CreateServiceLinkedRole permissions on multiple resources. We recommend that you specify resource ARNs instead."
```

Resolving the general warning

The action `iam:CreateServiceLinkedRole` grants permission to create an IAM role that allows an AWS service to perform actions on your behalf. Using the `NotAction` element with the `NotResource` element can allow creating unintended service-linked roles for multiple resources. AWS recommends that you rewrite the policy to allow `iam:CreateServiceLinkedRole` on a limited list of ARNs in the `Resource` element instead. You can also add `iam:CreateServiceLinkedRole` to the `NotAction` element.

- [CreateServiceLinkedRole operation](#)
- [IAM JSON policy elements: NotAction \(p. 1274\)](#)
- [IAM JSON policy elements: Action \(p. 1273\)](#)
- [IAM JSON policy elements: NotResource \(p. 1277\)](#)
- [IAM JSON policy elements: Resource \(p. 1276\)](#)

General Warning – Create SLR with star in resource

In the AWS Management Console, the finding for this check includes the following message:

Create SLR with star in resource: Using the `iam:CreateServiceLinkedRole` action with wildcards (*) in the `resource` can allow creation of unintended service-linked roles. We recommend that you specify resource ARNs instead.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Using the iam:CreateServiceLinkedRole action with wildcards (*) in the resource can allow creation of unintended service-linked roles. We recommend that you specify resource ARNs instead."
```

Resolving the general warning

The action `iam:CreateServiceLinkedRole` grants permission to create an IAM role that allows an AWS service to perform actions on your behalf. Using `iam:CreateServiceLinkedRole` in a policy with a wildcard (*) in the Resource element can allow creating unintended service-linked roles for multiple resources. AWS recommends that you specify allowed ARNs in the Resource element instead.

- [CreateServiceLinkedRole operation](#)
- [IAM JSON policy elements: Resource \(p. 1276\)](#)

AWS managed policies with this general warning

[AWS managed policies \(p. 495\)](#) enable you to get started with AWS by assigning permissions based on general AWS use cases.

Some of those use cases are for power users within your account. The following AWS managed policies provide power user access and grant permissions to create [service-linked roles \(p. 242\)](#) for any AWS service. AWS recommends that you attach the following AWS managed policies to only IAM identities that you consider power users.

- [PowerUserAccess](#)
- [AlexaForBusinessFullAccess](#)
- [AWSOrganizationsServiceTrustPolicy](#) – This AWS managed policy provides permissions for use by the AWS Organizations service-linked role. This role allows Organizations to create additional service-linked roles for other services in your AWS organization.

General Warning – Create SLR with star in action and resource

In the AWS Management Console, the finding for this check includes the following message:

Create SLR with star in action and resource: Using wildcards (*) in the action and the resource can allow creation of unintended service-linked roles because it allows `iam:CreateServiceLinkedRole` permissions on all resources. We recommend that you specify resource ARNs instead.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Using wildcards (*) in the action and the resource can allow creation of unintended service-linked roles because it allows iam:CreateServiceLinkedRole permissions on all resources. We recommend that you specify resource ARNs instead."
```

Resolving the general warning

The action `iam:CreateServiceLinkedRole` grants permission to create an IAM role that allows an AWS service to perform actions on your behalf. Policies with a wildcard (*) in the Action and Resource elements can allow creating unintended service-linked roles for multiple resources. This allows creating a service-linked role when you specify "Action": "*", "Action": "iam:*", or "Action": "iam:Create*". AWS recommends that you specify allowed ARNs in the Resource element instead.

- [CreateServiceLinkedRole operation](#)
- [IAM JSON policy elements: Action \(p. 1273\)](#)
- [IAM JSON policy elements: Resource \(p. 1276\)](#)

AWS managed policies with this general warning

[AWS managed policies \(p. 495\)](#) enable you to get started with AWS by assigning permissions based on general AWS use cases.

Some of those use cases are for administrators within your account. The following AWS managed policies provide administrator access and grant permissions to create [service-linked roles \(p. 242\)](#) for any AWS service. AWS recommends that you attach the following AWS managed policies to only the IAM identities that you consider administrators.

- [AdministratorAccess](#)
- [IAMFullAccess](#)

General Warning – Create SLR with star in resource and NotAction

In the AWS Management Console, the finding for this check includes the following message:

Create SLR with star in resource and NotAction: Using a resource with wildcards (*) and NotAction can allow creation of unintended service-linked roles because it allows iam:CreateServiceLinkedRole permissions on all resources. We recommend that you specify resource ARNs instead.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

"findingDetails": "Using a resource with wildcards (*) and NotAction can allow creation of unintended service-linked roles because it allows iam:CreateServiceLinkedRole permissions on all resources. We recommend that you specify resource ARNs instead."

Resolving the general warning

The action iam:CreateServiceLinkedRole grants permission to create an IAM role that allows an AWS service to perform actions on your behalf. Using the NotAction element in a policy with a wildcard (*) in the Resource element can allow creating unintended service-linked roles for multiple resources. AWS recommends that you specify allowed ARNs in the Resource element instead. You can also add iam:CreateServiceLinkedRole to the NotAction element.

- [CreateServiceLinkedRole operation](#)
- [IAM JSON policy elements: NotAction \(p. 1274\)](#)
- [IAM JSON policy elements: Action \(p. 1273\)](#)
- [IAM JSON policy elements: Resource \(p. 1276\)](#)

General Warning – Deprecated global condition key

In the AWS Management Console, the finding for this check includes the following message:

Deprecated global condition key: We recommend that you update aws:ARN to use the newer condition key aws:PrincipalArn.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "We recommend that you update aws:ARN to use the newer condition key aws:PrincipalArn."
```

Resolving the general warning

The policy includes a deprecated global condition key. Update the condition key in the condition key-value pair to use a supported global condition key.

- [Global condition keys \(p. 1338\)](#)

General Warning – Invalid date value

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid date value: The date {{date}} might not resolve as expected. We recommend that you use the YYYY-MM-DD format.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The date {{date}} might not resolve as expected. We recommend that you use the YYYY-MM-DD format."
```

Resolving the general warning

Unix Epoch time describes a point in time that has elapsed since January 1, 1970, minus leap seconds. Epoch time might not resolve to the precise time that you expect. AWS recommends that you use the W3C standard for date and time formats. For example, you could specify a complete date, such as YYYY-MM-DD (1997-07-16), or you could also append the time to the second, such as YYYY-MM-DDThh:mm:ssTZD (1997-07-16T19:20:30+01:00).

- [W3C Date and Time Formats](#)
- [IAM JSON policy elements: Version \(p. 1261\)](#)
- [aws:CurrentTime global condition key \(p. 1341\)](#)

General Warning – Invalid role reference

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid role reference: The Principal element includes the IAM role ID {{roleid}}. We recommend that you use a role ARN instead.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The Principal element includes the IAM role ID {{roleid}}. We recommend that you use a role ARN instead."
```

Resolving the general warning

AWS recommends that you specify the Amazon Resource Name (ARN) for an IAM role instead of its principal ID. When IAM saves the policy, it will transform the ARN into the principal ID for the existing role. AWS includes a safety precaution. If someone deletes and recreates the role, it will have a new ID, and the policy won't match the new role's ID.

- [Specifying a principal: IAM roles \(p. 1266\)](#)
- [IAM ARNs \(p. 1213\)](#)
- [IAM unique IDs \(p. 1218\)](#)

General Warning – Invalid user reference

In the AWS Management Console, the finding for this check includes the following message:

Invalid user reference: The Principal element includes the IAM user ID {{userid}}. We recommend that you use a user ARN instead.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

"findingDetails": "The Principal element includes the IAM user ID {{userid}}. We recommend that you use a user ARN instead."

Resolving the general warning

AWS recommends that you specify the Amazon Resource Name (ARN) for an IAM user instead of its principal ID. When IAM saves the policy, it will transform the ARN into the principal ID for the existing user. AWS includes a safety precaution. If someone deletes and recreates the user, it will have a new ID, and the policy won't match the new user's ID.

- [Specifying a principal: IAM users \(p. 1268\)](#)
- [IAM ARNs \(p. 1213\)](#)
- [IAM unique IDs \(p. 1218\)](#)

General Warning – Missing version

In the AWS Management Console, the finding for this check includes the following message:

Missing version: We recommend that you specify the Version element to help you with debugging permission issues.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

"findingDetails": "We recommend that you specify the Version element to help you with debugging permission issues."

Resolving the general warning

AWS recommends that you include the optional Version parameter in your policy. If you do not include a Version element, the value defaults to 2012-10-17, 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: Version \(p. 1261\)](#)

General Warning – Unique Sids recommended

In the AWS Management Console, the finding for this check includes the following message:

Unique Sids recommended: We recommend that you use statement IDs that are unique to your policy. Update the Sid value.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "We recommend that you use statement IDs that are unique to your policy.  
Update the Sid value."
```

Resolving the general warning

AWS recommends that you use unique statement IDs. The Sid (statement ID) element allows you to enter an optional identifier that you provide for the policy statement. You can assign a statement ID value to each statement in a statement array using the SID element.

Related terms

- [IAM JSON policy elements: Sid \(p. 1263\)](#)

General Warning – Wildcard without like operator

In the AWS Management Console, the finding for this check includes the following message:

Wildcard without like operator: Your condition value includes a * or ? character. If you meant to use a wildcard (*, ?), update the condition operator to include Like.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Your condition value includes a * or ? character. If you meant to use a  
wildcard (*, ?), update the condition operator to include Like."
```

Resolving the general warning

The Condition element structure requires that you use a condition operator and a key-value pair. When you specify a condition value that uses a wildcard (*, ?), you must use the Like version of the condition operator. For example, instead of the StringEquals string condition operator, use StringLike.

```
"Condition": {"StringLike": {"aws:PrincipalTag/job-category": "admin-*"}}
```

- [IAM JSON policy elements: Condition operators \(p. 1281\)](#)
- [IAM JSON policy elements: Condition \(p. 1278\)](#)

AWS managed policies with this general warning

[AWS managed policies \(p. 495\)](#) enable you to get started with AWS by assigning permissions based on general AWS use cases.

The following AWS managed policies include wildcards in their condition value without a condition operator that includes Like for pattern-matching. When using the AWS managed policy as a reference to create your customer managed policy, AWS recommends that you use a condition operator that supports pattern-matching with wildcards (*, ?), such as StringLike.

- [AWSGlueConsoleSageMakerNotebookFullAccess](#)

General Warning – Policy size exceeds identity policy quota

In the AWS Management Console, the finding for this check includes the following message:

Policy size exceeds identity policy quota: The {{policySize}} characters in the identity policy, excluding whitespace, exceed the {{policySizeQuota}} character maximum for inline and managed policies. We recommend that you use multiple granular policies.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The {{policySize}} characters in the identity policy, excluding whitespace, exceed the {{policySizeQuota}} character maximum for inline and managed policies. We recommend that you use multiple granular policies."
```

Resolving the general warning

You can attach up to 10 managed policies to an IAM identity (user, group of users, or role). However, the size of each managed policy cannot exceed the default quota of 6,144 characters. IAM does not count white space when calculating the size of a policy against this quota. Quotas, also referred to as limits in AWS, are the maximum values for the resources, actions, and items in your AWS account.

Additionally, you can add as many inline policies as you want to an IAM identity. However, the sum size of all inline policies per identity cannot exceed the specified quota.

If your policy is larger than the quota, you can organize your policy into multiple statements and group the statements into multiple policies.

Related terms

- [IAM and AWS STS character quotas \(p. 1220\)](#)
- [Multiple statements and multiple policies \(p. 492\)](#)
- [IAM customer managed policies \(p. 496\)](#)
- [Overview of JSON policies \(p. 490\)](#)
- [IAM JSON policy grammar \(p. 1323\)](#)

AWS managed policies with this general warning

[AWS managed policies \(p. 495\)](#) enable you to get started with AWS by assigning permissions based on general AWS use cases.

The following AWS managed policies grant permissions to actions across many AWS services and exceed the maximum policy size. When using the AWS managed policy as a reference to create your managed policy, you must split the policy into multiple policies.

- [ReadOnlyAccess](#)
- [AWSSupportServiceRolePolicy](#)

General Warning – Policy size exceeds resource policy quota

In the AWS Management Console, the finding for this check includes the following message:

Policy size exceeds resource policy quota: The {{policySize}} characters in the resource policy exceed the {{policySizeQuota}} character maximum for resource policies. We recommend that you use multiple granular policies.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The {{policySize}} characters in the resource policy exceed the {{policySizeQuota}} character maximum for resource policies. We recommend that you use multiple granular policies."
```

Resolving the general warning

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 define under what conditions this applies. The size of resource-based policies cannot exceed the quota set for that resource. Quotas, also referred to as limits in AWS, are the maximum values for the resources, actions, and items in your AWS account.

If your policy is larger than the quota, you can organize your policy into multiple statements and group the statements into multiple policies.

Related terms

- [Resource-based policies \(p. 486\)](#)
- [Amazon S3 bucket policies](#)
- [Multiple statements and multiple policies \(p. 492\)](#)
- [Overview of JSON policies \(p. 490\)](#)
- [IAM JSON policy grammar \(p. 1323\)](#)

General Warning – Type mismatch

In the AWS Management Console, the finding for this check includes the following message:

Type mismatch: Use the operator type {{allowed}} instead of operator {{operator}} for the condition key {{key}}.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Use the operator type {{allowed}} instead of operator {{operator}} for the condition key {{key}}."
```

Resolving the general warning

Update the text to use the supported condition operator data type.

For example, the `aws:MultiFactorAuthPresent` global condition key requires a condition operator with the Boolean data type. If you provide a date or an integer, the data type won't match.

Related terms

- [Global condition keys \(p. 1338\)](#)
- [IAM JSON policy elements: Condition operators \(p. 1281\)](#)

General Warning – Type mismatch Boolean

In the AWS Management Console, the finding for this check includes the following message:

```
Type mismatch Boolean: Add a valid Boolean value (true or false) for the condition operator {{operator}}.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Add a valid Boolean value (true or false) for the condition operator {{operator}}."
```

Resolving the general warning

Update the text to use a Boolean condition operator data type, such as true or false.

For example, the aws :MultiFactorAuthPresent global condition key requires a condition operator with the Boolean data type. If you provide a date or an integer, the data type won't match.

Related terms

- [Boolean condition operators \(p. 1285\)](#)
- [IAM JSON policy elements: Condition operators \(p. 1281\)](#)

General Warning – Type mismatch date

In the AWS Management Console, the finding for this check includes the following message:

```
Type mismatch date: The date condition operator is used with an invalid value. Specify a valid date using YYYY-MM-DD or other ISO 8601 date/time format.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The date condition operator is used with an invalid value. Specify a valid date using YYYY-MM-DD or other ISO 8601 date/time format."
```

Resolving the general warning

Update the text to use the date condition operator data type, in a YYYY-MM-DD or other ISO 8601 date time format.

Related terms

- [Date condition operators \(p. 1285\)](#)
- [IAM JSON policy elements: Condition operators \(p. 1281\)](#)

General Warning – Type mismatch number

In the AWS Management Console, the finding for this check includes the following message:

```
Type mismatch number: Add a valid numeric value for the condition operator {{operator}}.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Add a valid numeric value for the condition operator {{operator}}."
```

Resolving the general warning

Update the text to use the numeric condition operator data type.

Related terms

- [Numeric condition operators \(p. 1284\)](#)
- [IAM JSON policy elements: Condition operators \(p. 1281\)](#)

General Warning – Type mismatch string

In the AWS Management Console, the finding for this check includes the following message:

```
Type mismatch string: Add a valid base64-encoded string value for the condition operator {{operator}}.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Add a valid base64-encoded string value for the condition operator {{operator}}."
```

Resolving the general warning

Update the text to use the string condition operator data type.

Related terms

- [String condition operators \(p. 1282\)](#)
- [IAM JSON policy elements: Condition operators \(p. 1281\)](#)

General Warning – Specific github repo and branch recommended

In the AWS Management Console, the finding for this check includes the following message:

```
Specific github repo and branch recommended: Using a wildcard (*) in token.actions.githubusercontent.com:sub can allow requests from more sources than you intended. Specify the value of token.actions.githubusercontent.com:sub with the repository and branch name.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Using a wildcard (*) in token.actions.githubusercontent.com:sub can allow requests from more sources than you intended. Specify the value of token.actions.githubusercontent.com:sub with the repository and branch name."
```

Resolving the general warning

If you use GitHub as an OIDC IdP, best practice is to limit the entities that can assume the role associated with the IAM IdP. When you include a Condition statement in a role trust policy, you can limit the role to a specific GitHub organization, repository, or branch. You can use the condition key `token.actions.githubusercontent.com:sub` to limit access. We recommend that you limit the condition to a specific set of repositories or branches. If you use a wildcard (*) in `token.actions.githubusercontent.com:sub`, then GitHub Actions from organizations or repositories outside of your control are able to assume roles associated with the GitHub IAM IdP in your AWS account.

Related terms

- [Configuring a role for GitHub OIDC identity provider](#)

General Warning – Policy size exceeds role trust policy quota

In the AWS Management Console, the finding for this check includes the following message:

Policy size exceeds role trust policy quota: The characters in the role trust policy, excluding whitespace, exceed the character maximum. We recommend that you request a role trust policy length quota increase using Service Quotas and AWS Support Center. If the quotas have already been increased, then you can ignore this warning.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

"findingDetails": "The characters in the role trust policy, excluding whitespace, exceed the character maximum. We recommend that you request a role trust policy length quota increase using Service Quotas and AWS Support Center. If the quotas have already been increased, then you can ignore this warning."

Resolving the general warning

IAM and AWS STS have quotas that limit the size of role trust policies. The characters in the role trust policy, excluding whitespace, exceed the character maximum. We recommend that you request a role trust policy length quota increase using Service Quotas and the AWS Support Center Console.

Related terms

- [IAM and AWS STS quotas, name requirements, and character limits](#)

Security Warning – Allow with NotPrincipal

In the AWS Management Console, the finding for this check includes the following message:

Allow with NotPrincipal: Using Allow with NotPrincipal can be overly permissive. We recommend that you use Principal instead.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

"findingDetails": "Using Allow with NotPrincipal can be overly permissive. We recommend that you use Principal instead."

Resolving the security warning

Using "Effect": "Allow" with the NotPrincipal can be overly permissive. For example, this can grant permissions to anonymous principals. AWS recommends that you specify principals that need access using the Principal element. Alternatively, you can allow broad access and then add another statement that uses the NotPrincipal element with "Effect": "Deny".

- [AWS JSON policy elements: Principal \(p. 1264\)](#)
- [AWS JSON policy elements: NotPrincipal \(p. 1272\)](#)

Security Warning – ForAllValues with single valued key

In the AWS Management Console, the finding for this check includes the following message:

ForAllValues with single valued key: Using ForAllValues qualifier with the single-valued condition key {{key}} can be overly permissive. We recommend that you remove ForAllValues:.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

"findingDetails": "Using ForAllValues qualifier with the single-valued condition key {{key}} can be overly permissive. We recommend that you remove ForAllValues:."

Resolving the security warning

AWS recommends that you use the ForAllValues only with multivalued conditions. The ForAllValues set operator 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.

To learn whether a condition supports a single value or multiple values, review the [Actions, resources, and condition keys](#) page for the service. Condition keys with the ArrayOf data type prefix are multivalued condition keys. For example, Amazon SES supports keys with single values (String) and the ArrayOfString multivalued data type.

- [Multivalued context keys \(p. 1294\)](#)

Security Warning – Pass role with NotResource

In the AWS Management Console, the finding for this check includes the following message:

Pass role with NotResource: Using the iam:PassRole action with NotResource can be overly permissive because it can allow iam:PassRole permissions on multiple resources. We recommend that you specify resource ARNs instead.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

"findingDetails": "Using the iam:PassRole action with NotResource can be overly permissive because it can allow iam:PassRole permissions on multiple resources. We recommend that you specify resource ARNs instead."

Resolving the security warning

To configure many AWS services, you must pass an IAM role to the service. To allow this you must grant the `iam:PassRole` permission to an identity (user, group of users, or role). Using `iam:PassRole` in a policy with the `NotResource` element can allow your principals to access more services or features than you intended. AWS recommends that you specify allowed ARNs in the `Resource` element instead. Additionally, you can reduce permissions to a single service by using the `iam:PassedToService` condition key.

- [Passing a role to a service \(p. 279\)](#)
- [iam:PassedToService \(p. 1371\)](#)
- [IAM JSON policy elements: NotResource \(p. 1277\)](#)
- [IAM JSON policy elements: Resource \(p. 1276\)](#)

Security Warning – Pass role with star in action and NotResource

In the AWS Management Console, the finding for this check includes the following message:

Pass role with star in action and NotResource: Using an action with a wildcard (*) and `NotResource` can be overly permissive because it can allow `iam:PassRole` permissions on multiple resources. We recommend that you specify resource ARNs instead.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Using an action with a wildcard (*) and NotResource can be overly permissive because it can allow iam:PassRole permissions on multiple resources. We recommend that you specify resource ARNs instead."
```

Resolving the security warning

To configure many AWS services, you must pass an IAM role to the service. To allow this you must grant the `iam:PassRole` permission to an identity (user, group of users, or role). Policies with a wildcard (*) in the `Action` and that include the `NotResource` element can allow your principals to access more services or features than you intended. AWS recommends that you specify allowed ARNs in the `Resource` element instead. Additionally, you can reduce permissions to a single service by using the `iam:PassedToService` condition key.

- [Passing a role to a service \(p. 279\)](#)
- [iam:PassedToService \(p. 1371\)](#)
- [IAM JSON policy elements: NotResource \(p. 1277\)](#)
- [IAM JSON policy elements: Resource \(p. 1276\)](#)

Security Warning – Pass role with NotAction and NotResource

In the AWS Management Console, the finding for this check includes the following message:

Pass role with NotAction and NotResource: Using `NotAction` with `NotResource` can be overly permissive because it can allow `iam:PassRole` permissions on multiple resources.. We recommend that you specify resource ARNs instead.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

"findingDetails": "Using NotAction with NotResource can be overly permissive because it can allow iam:PassRole permissions on multiple resources.. We recommend that you specify resource ARNs instead."

Resolving the security warning

To configure many AWS services, you must pass an IAM role to the service. To allow this you must grant the iam:PassRole permission to an identity (user, group of users, or role). Using the NotAction element and listing some resources in the NotResource element can allow your principals to access more services or features than you intended. AWS recommends that you specify allowed ARNs in the Resource element instead. Additionally, you can reduce permissions to a single service by using the iam:PassedToService condition key.

- [Passing a role to a service \(p. 279\)](#)
- [iam:PassedToService \(p. 1371\)](#)
- [IAM JSON policy elements: NotAction \(p. 1274\)](#)
- [IAM JSON policy elements: Action \(p. 1273\)](#)
- [IAM JSON policy elements: NotResource \(p. 1277\)](#)
- [IAM JSON policy elements: Resource \(p. 1276\)](#)

Security Warning – Pass role with star in resource

In the AWS Management Console, the finding for this check includes the following message:

Pass role with star in resource: Using the iam:PassRole action with wildcards (*) in the resource can be overly permissive because it allows iam:PassRole permissions on multiple resources. We recommend that you specify resource ARNs or add the iam:PassedToService condition key to your statement.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

"findingDetails": "Using the iam:PassRole action with wildcards (*) in the resource can be overly permissive because it allows iam:PassRole permissions on multiple resources. We recommend that you specify resource ARNs or add the iam:PassedToService condition key to your statement."

Resolving the security warning

To configure many AWS services, you must pass an IAM role to the service. To allow this you must grant the iam:PassRole permission to an identity (user, group of users, or role). Policies that allow iam:PassRole and that include a wildcard (*) in the Resource element can allow your principals to access more services or features than you intended. AWS recommends that you specify allowed ARNs in the Resource element instead. Additionally, you can reduce permissions to a single service by using the iam:PassedToService condition key.

Some AWS services include their service namespace in the name of their role. This policy check takes these conventions into account while analyzing the policy to generate findings. For example, the following resource ARN might not generate a finding:

`arn:aws:iam::*:role/Service*`

- [Passing a role to a service \(p. 279\)](#)
- [iam:PassedToService \(p. 1371\)](#)

- [IAM JSON policy elements: Resource \(p. 1276\)](#)

AWS managed policies with this security warning

[AWS managed policies \(p. 495\)](#) enable you to get started with AWS by assigning permissions based on general AWS use cases.

One of those use cases is for administrators within your account. The following AWS managed policies provide administrator access and grant permissions to pass any IAM role to any service. AWS recommends that you attach the following AWS managed policies only to IAM identities that you consider administrators.

- [AdministratorAccess-Amplify](#)

The following AWS managed policies include permissions to `iam:PassRole` with a wildcard (*) in the resource and are on a [deprecation path \(p. 501\)](#). For each of these policies, we updated the permission guidance, such as recommending a new AWS managed policy that supports the use case. To view alternatives to these policies, see the guides for [each service \(p. 1224\)](#).

- AWSElasticBeanstalkFullAccess
- AWSElasticBeanstalkService
- AWSLambdaFullAccess
- AWSLambdaReadOnlyAccess
- AWSOpsWorksFullAccess
- AWSOpsWorksRole
- AWSDataPipelineRole
- AmazonDynamoDBFullAccesswithDataPipeline
- AmazonElasticMapReduceFullAccess
- AmazonDynamoDBFullAccesswithDataPipeline
- AmazonEC2ContainerServiceFullAccess

The following AWS managed policies provide permissions for only [service-linked roles \(p. 242\)](#), which allow AWS services to perform actions on your behalf. You cannot attach these policies to your IAM identities.

- [AWSServiceRoleForAmazonEKSNodegroup](#)

Security Warning – Pass role with star in action and resource

In the AWS Management Console, the finding for this check includes the following message:

Pass role with star in action and resource: Using wildcards (*) in the action and the resource can be overly permissive because it allows `iam:PassRole` permissions on all resources. We recommend that you specify resource ARNs or add the `iam:PassedToService` condition key to your statement.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Using wildcards (*) in the action and the resource can be overly permissive because it allows iam:PassRole permissions on all resources. We recommend that you specify resource ARNs or add the iam:PassedToService condition key to your statement."
```

Resolving the security warning

To configure many AWS services, you must pass an IAM role to the service. To allow this you must grant the `iam:PassRole` permission to an identity (user, group of users, or role). Policies with a wildcard (*) in the Action and Resource elements can allow your principals to access more services or features than you intended. AWS recommends that you specify allowed ARNs in the Resource element instead. Additionally, you can reduce permissions to a single service by using the `iam:PassedToService` condition key.

- [Passing a role to a service \(p. 279\)](#)
- [iam:PassedToService \(p. 1371\)](#)
- [IAM JSON policy elements: Action \(p. 1273\)](#)
- [IAM JSON policy elements: Resource \(p. 1276\)](#)

AWS managed policies with this security warning

[AWS managed policies \(p. 495\)](#) enable you to get started with AWS by assigning permissions based on general AWS use cases.

Some of those use cases are for administrators within your account. The following AWS managed policies provide administrator access and grant permissions to pass any IAM role to any AWS service. AWS recommends that you attach the following AWS managed policies to only the IAM identities that you consider administrators.

- [AdministratorAccess](#)
- [IAMFullAccess](#)

Security Warning – Pass role with star in resource and NotAction

In the AWS Management Console, the finding for this check includes the following message:

Pass role with star in resource and NotAction: Using a resource with wildcards (*) and NotAction can be overly permissive because it allows `iam:PassRole` permissions on all resources. We recommend that you specify resource ARNs or add the `iam:PassedToService` condition key to your statement.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Using a resource with wildcards (*) and NotAction can be overly permissive because it allows iam:PassRole permissions on all resources. We recommend that you specify resource ARNs or add the iam:PassedToService condition key to your statement."
```

Resolving the security warning

To configure many AWS services, you must pass an IAM role to the service. To allow this you must grant the `iam:PassRole` permission to an identity (user, group of users, or role). Using the `NotAction` element in a policy with a wildcard (*) in the Resource element can allow your principals to access more services or features than you intended. AWS recommends that you specify allowed ARNs in the Resource element instead. Additionally, you can reduce permissions to a single service by using the `iam:PassedToService` condition key.

- [Passing a role to a service \(p. 279\)](#)
- [iam:PassedToService \(p. 1371\)](#)

- [IAM JSON policy elements: NotAction \(p. 1274\)](#)
- [IAM JSON policy elements: Action \(p. 1273\)](#)
- [IAM JSON policy elements: Resource \(p. 1276\)](#)

Security Warning – Missing paired condition keys

In the AWS Management Console, the finding for this check includes the following message:

Missing paired condition keys: Using the condition key {{conditionKeyName}} can be overly permissive without also using the following condition keys: {{recommendedKeys}}. Condition keys like this one are more secure when paired with a related key. We recommend that you add the related condition keys to the same condition block.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

"findingDetails": "Using the condition key {{conditionKeyName}} can be overly permissive without also using the following condition keys: {{recommendedKeys}}. Condition keys like this one are more secure when paired with a related key. We recommend that you add the related condition keys to the same condition block."

Resolving the security warning

Some condition keys are more secure when paired with other related condition keys. AWS recommends that you include the related condition keys in the same condition block as the existing condition key. This makes the permissions granted through the policy more secure.

For example, you can use the aws:VpcSourceIp condition key to compare the IP address from which a request was made with the IP address that you specify in the policy. AWS recommends that you add the related aws:SourceVPC condition key. This checks whether the request comes from the VPC that you specify in the policy *and* the IP address that you specify.

Related terms

- [aws:VpcSourceIp global condition key \(p. 1366\)](#)
- [aws:SourceVPC global condition key \(p. 1364\)](#)
- [Global condition keys \(p. 1338\)](#)
- [Condition element \(p. 1278\)](#)
- [Overview of JSON policies \(p. 490\)](#)

Security Warning – Deny with unsupported tag condition key for service

In the AWS Management Console, the finding for this check includes the following message:

Deny with unsupported tag condition key for service: Using the effect Deny with the tag condition key {{conditionKeyName}} and actions for services with the following prefixes can be overly permissive: {{serviceNames}}. Actions for the listed services are not denied by this statement. We recommend that you move these actions to a different statement without this condition key.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

"findingDetails": "Using the effect Deny with the tag condition key {{conditionKeyName}} and actions for services with the following prefixes can be overly permissive: {{serviceNames}}. Actions for the listed services are not denied by this statement. We recommend that you move these actions to a different statement without this condition key."

Resolving the security warning

Using unsupported tag condition keys in the Condition element of a policy with "Effect": "Deny" can be overly permissive, because the condition is ignored for that service. AWS recommends that you remove the service actions that don't support the condition key and create another statement to deny access to specific resources for those actions.

If you use the aws:ResourceTag condition key and it's not supported by a service action, then the key is not included in the request context. In this case, the condition in the Deny statement always returns false and the action is never denied. This happens even if the resource is tagged correctly.

When a service supports the aws:ResourceTag condition key, you can use tags to control access to that service's resources. This is known as [attribute-based access control \(ABAC\) \(p. 15\)](#). Services that don't support these keys require you to control access to resources using [resource-based access control \(RBAC\) \(p. 16\)](#).

Note

Some services allow support for the aws:ResourceTag condition key for a subset of their resources and actions. IAM Access Analyzer returns findings for the service actions that are not supported. For example, Amazon S3 supports aws:ResourceTag for a subset of its resources. To view all of the resource types available in Amazon S3 that support the aws:ResourceTag condition key, see [Resource types defined by Amazon S3](#) in the Service Authorization Reference.

For example, assume that you want to deny access to untag delete specific resources that are tagged with the key-value pair status=Confidential. Also assume that AWS Lambda allows you to tag and untag resources, but doesn't support the aws:ResourceTag condition key. To deny the delete actions for AWS App Mesh and AWS Backup if this tag is present, use the aws:ResourceTag condition key. For Lambda, use a resource naming convention that includes the "Confidential" prefix. Then include a separate statement that prevents deleting resources with that naming convention.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "DenyDeleteSupported",  
            "Effect": "Deny",  
            "Action": [  
                "appmesh:DeleteMesh",  
                "backup:DeleteBackupPlan"  
            ],  
            "Resource": "*",  
            "Condition": {  
                "StringEquals": {  
                    "aws:ResourceTag/status                }  
            }  
        },  
        {  
            "Sid": "DenyDeleteUnsupported",  
            "Effect": "Deny",  
            "Action": "lambda:DeleteFunction",  
            "Resource": "arn:aws:lambda:*:123456789012:function:status-Confidential*"  
        }  
    ]  
}
```

Warning

Do not use the [...IfExists \(p. 1288\)](#) version of the condition operator as a workaround for this finding. This means "Deny the action if the key is present in the request context and the values match. Otherwise, deny the action." In the previous example, including the `lambda:DeleteFunction` action in the `DenyDeleteSupported` statement with the `StringEqualsIfExists` operator always denies the action. For that action, the key is not present in the context, and every attempt to delete that resource type is denied, regardless of whether the resource is tagged.

Related terms

- [Global condition keys \(p. 1338\)](#)
- [Comparing ABAC to RBAC \(p. 16\)](#)
- [IAM JSON policy elements: Condition operators \(p. 1281\)](#)
- [Condition element \(p. 1278\)](#)
- [Overview of JSON policies \(p. 490\)](#)

Security Warning – Deny NotAction with unsupported tag condition key for service

In the AWS Management Console, the finding for this check includes the following message:

Deny NotAction with unsupported tag condition key for service: Using the effect Deny with NotAction and the tag condition key `{conditionKeyName}` can be overly permissive because some service actions are not denied by this statement. This is because the condition key doesn't apply to some service actions. We recommend that you use Action instead of NotAction.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

"findingDetails": "Using the effect Deny with NotAction and the tag condition key `{conditionKeyName}` can be overly permissive because some service actions are not denied by this statement. This is because the condition key doesn't apply to some service actions. We recommend that you use Action instead of NotAction."

Resolving the security warning

Using tag condition keys in the Condition element of a policy with the element NotAction and "Effect": "Deny" can be overly permissive. The condition is ignored for service actions that don't support the condition key. AWS recommends that you rewrite the logic to deny a list of actions.

If you use the `aws:ResourceTag` condition key with NotAction, any new or existing service actions that don't support the key are not denied. AWS recommends that you explicitly list the actions that you want to deny. IAM Access Analyzer returns a separate finding for listed actions that don't support the `aws:ResourceTag` condition key. For more information, see [Security Warning – Deny with unsupported tag condition key for service \(p. 1135\)](#).

When a service supports the `aws:ResourceTag` condition key, you can use tags to control access to that service's resources. This is known as [attribute-based access control \(ABAC\) \(p. 15\)](#). Services that don't support these keys require you to control access to resources using [resource-based access control \(RBAC\) \(p. 16\)](#).

Related terms

- [Global condition keys \(p. 1338\)](#)
- [Comparing ABAC to RBAC \(p. 16\)](#)
- [IAM JSON policy elements: Condition operators \(p. 1281\)](#)
- [Condition element \(p. 1278\)](#)
- [Overview of JSON policies \(p. 490\)](#)

Security Warning – Restrict access to service principal

In the AWS Management Console, the finding for this check includes the following message:

Restrict access to service principal: Granting access to a service principal without specifying a source is overly permissive. Use aws:SourceArn or aws:SourceAccount condition key and scope access to a specific source to grant fine-grained access.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

"findingDetails": "Granting access to a service principal without specifying a source is overly permissive. Use aws:SourceArn or aws:SourceAccount condition key and scope access to a specific source to grant fine-grained access."

Resolving the security warning

You can specify AWS services in the Principal element of a resource-based policy using a *service principal*, which is an identifier for the service. When granting access to a service principal to act on your behalf, restrict access. You can prevent overly permissive policies by using the aws:SourceAccount or aws:SourceArn condition keys to restrict access to a specific source, such as a specific resource ARN or AWS account. Restricting access helps you prevent a security issue called the *confused deputy problem*.

Related terms

- [Service principals](#)
- [AWS global condition keys: aws:SourceAccount](#)
- [AWS global condition keys: aws:SourceArn](#)
- [The confused deputy problem](#)

Security Warning – Missing condition key for oidc principal

In the AWS Management Console, the finding for this check includes the following message:

Missing condition key for oidc principal: Using an Open ID Connect principal without a condition can be overly permissive. Add condition keys with a prefix that matches your federated OIDC principals to ensure that only the intended identity provider assumes the role.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

"findingDetails": "Using an Open ID Connect principal without a condition can be overly permissive. Add condition keys with a prefix that matches your federated OIDC principals to ensure that only the intended identity provider assumes the role."

Resolving the security warning

Using an Open ID Connect principal without a condition can be overly permissive. Add condition keys with a prefix that matches your federated OIDC principals to ensure that only the intended identity provider assumes the role.

Related terms

- [Creating a role for web identity or OpenID Connect Federation \(console\)](#)

Security Warning – Missing github repo condition key

In the AWS Management Console, the finding for this check includes the following message:

Missing github repo condition key: Granting a federated GitHub principal permissions without a condition key can allow more sources to assume the role than you intended. Add the token.actions.githubusercontent.com:sub condition key and specify the branch and repository name in the value.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

"findingDetails": "Granting a federated GitHub principal permissions without a condition key can allow more sources to assume the role than you intended. Add the token.actions.githubusercontent.com:sub condition key and specify the branch and repository name in the value."

Resolving the security warning

If you use GitHub as an OIDC IdP, best practice is to limit the entities that can assume the role associated with the IAM IdP. When you include a Condition statement in a role trust policy, you can limit the role to a specific GitHub organization, repository, or branch. You can use the condition key token.actions.githubusercontent.com:sub to limit access. We recommend that you limit the condition to a specific set of repositories or branches. If you do not include this condition, then GitHub Actions from organizations or repositories outside of your control are able to assume roles associated with the GitHub IAM IdP in your AWS account.

Related terms

- [Configuring a role for GitHub OIDC identity provider](#)

Suggestion – Empty array action

In the AWS Management Console, the finding for this check includes the following message:

Empty array action: This statement includes no actions and does not affect the policy. Specify actions.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

"findingDetails": "This statement includes no actions and does not affect the policy. Specify actions."

Resolving the suggestion

Statements must include either an Action or NotAction element that includes a set of actions. When the element is empty, the policy statement provides no permissions. Specify actions in the Action element.

- [IAM JSON policy elements: Action \(p. 1273\)](#)

Suggestion – Empty array condition

In the AWS Management Console, the finding for this check includes the following message:

Empty array condition: There are no values for the condition key {{key}} and it does not affect the policy. Specify conditions.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

"findingDetails": "There are no values for the condition key {{key}} and it does not affect the policy. Specify conditions."

Resolving the suggestion

The optional Condition element structure requires that you use a condition operator and a key-value pair. When the condition value is empty, the condition returns true and the policy statement provides no permissions. Specify a condition value.

- [IAM JSON policy elements: Condition \(p. 1278\)](#)

Suggestion – Empty array condition ForAllValues

In the AWS Management Console, the finding for this check includes the following message:

Empty array condition ForAllValues: The ForAllValues prefix with an empty condition key matches only if the key {{key}} is missing from the request context. To determine if the request context is empty, we recommend that you use the Null condition operator with the value of true instead.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

"findingDetails": "The ForAllValues prefix with an empty condition key matches only if the key {{key}} is missing from the request context. To determine if the request context is empty, we recommend that you use the Null condition operator with the value of true instead."

Resolving the suggestion

The Condition element structure requires that you use a condition operator and a key-value pair. The ForAllValues set operator tests whether the value of every member of the request set is a subset of the condition key set.

When you use ForAllValues with an empty condition key, the condition matches only if there are no keys in the request. AWS recommends that if you want to test whether a request context is empty, use the Null condition operator instead.

- [Multivalued context keys \(p. 1294\)](#)

- [Null condition operator \(p. 1290\)](#)
- [IAM JSON policy elements: Condition \(p. 1278\)](#)

Suggestion – Empty array condition ForAnyValue

In the AWS Management Console, the finding for this check includes the following message:

Empty array condition ForAnyValue: The ForAnyValue prefix with an empty condition key {{key}} never matches the request context and it does not affect the policy. Specify conditions.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

"findingDetails": "The ForAnyValue prefix with an empty condition key {{key}} never matches the request context and it does not affect the policy. Specify conditions."

Resolving the suggestion

The Condition element structure requires that you use a condition operator and a key-value pair. The ForAnyValues set operator tests whether at least one member of the set of request values matches at least one member of the set of condition key values.

When you use ForAnyValues with an empty condition key, the condition never matches. This means that the statement has no effect on the policy. AWS recommends that you rewrite the condition.

- [Multivalued context keys \(p. 1294\)](#)
- [IAM JSON policy elements: Condition \(p. 1278\)](#)

Suggestion – Empty array condition IfExists

In the AWS Management Console, the finding for this check includes the following message:

Empty array condition IfExists: The IfExists suffix with an empty condition key matches only if the key {{key}} is missing from the request context. To determine if the request context is empty, we recommend that you use the Null condition operator with the value of true instead.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

"findingDetails": "The IfExists suffix with an empty condition key matches only if the key {{key}} is missing from the request context. To determine if the request context is empty, we recommend that you use the Null condition operator with the value of true instead."

Resolving the suggestion

The ...IfExists suffix edits a condition operator. It means that 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.

When you use ...IfExists with an empty condition key, the condition matches only if there are no keys in the request. AWS recommends that if you want to test whether a request context is empty, use the Null condition operator instead.

- [...IfExists condition operators \(p. 1288\)](#)
- [IAM JSON policy elements: Condition \(p. 1278\)](#)

Suggestion – Empty array principal

In the AWS Management Console, the finding for this check includes the following message:

Empty array principal: This statement includes no principals and does not affect the policy. Specify principals.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

"findingDetails": "This statement includes no principals and does not affect the policy. Specify principals."

Resolving the suggestion

You must use the Principal or NotPrincipal element in the trust policies for IAM roles and in resource-based policies. Resource-based policies are policies that you embed directly in a resource.

When you provide an empty array in a statement's Principal element, the statement has no effect on the policy. AWS recommends that you specify the principals that should have access to the resource.

- [IAM JSON policy elements: Principal \(p. 1264\)](#)
- [IAM JSON policy elements: NotPrincipal \(p. 1272\)](#)

Suggestion – Empty array resource

In the AWS Management Console, the finding for this check includes the following message:

Empty array resource: This statement includes no resources and does not affect the policy. Specify resources.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

"findingDetails": "This statement includes no resources and does not affect the policy. Specify resources."

Resolving the suggestion

Statements must include either a Resource or a NotResource element.

When you provide an empty array in a statement's resource element, the statement has no effect on the policy. AWS recommends that you specify Amazon Resource Names (ARNs) for resources.

- [IAM JSON policy elements: Resource \(p. 1276\)](#)
- [IAM JSON policy elements: NotResource \(p. 1277\)](#)

Suggestion – Empty object condition

In the AWS Management Console, the finding for this check includes the following message:

Empty object condition: This condition block is empty and it does not affect the policy. Specify conditions.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "This condition block is empty and it does not affect the policy. Specify conditions."
```

Resolving the suggestion

The Condition element structure requires that you use a condition operator and a key-value pair.

When you provide an empty object in a statement's condition element, the statement has no effect on the policy. Remove the optional element or specify conditions.

- [IAM JSON policy elements: Condition \(p. 1278\)](#)

Suggestion – Empty object principal

In the AWS Management Console, the finding for this check includes the following message:

Empty object principal: This statement includes no principals and does not affect the policy. Specify principals.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "This statement includes no principals and does not affect the policy. Specify principals."
```

Resolving the suggestion

You must use the Principal or NotPrincipal element in the trust policies for IAM roles and in resource-based policies. Resource-based policies are policies that you embed directly in a resource.

When you provide an empty object in a statement's Principal element, the statement has no effect on the policy. AWS recommends that you specify the principals that should have access to the resource.

- [IAM JSON policy elements: Principal \(p. 1264\)](#)
- [IAM JSON policy elements: NotPrincipal \(p. 1272\)](#)

Suggestion – Empty Sid value

In the AWS Management Console, the finding for this check includes the following message:

Empty Sid value: Add a value to the empty string in the Sid element.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Add a value to the empty string in the Sid element."
```

Resolving the suggestion

The optional Sid (statement ID) element allows you to enter an identifier that you provide for the policy statement. You can assign an Sid value to each statement in a statement array. If you choose to use the Sid element, you must provide a string value.

Related terms

- [IAM JSON policy elements: Sid \(p. 1263\)](#)

Suggestion – Improve IP range

In the AWS Management Console, the finding for this check includes the following message:

```
Improve IP range: The non-zero bits in the IP address after the masked bits are ignored.  
Replace address with {{addr}}.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The non-zero bits in the IP address after the masked bits are ignored.  
Replace address with {{addr}}."
```

Resolving the suggestion

IP address conditions must be in the standard CIDR format, such as 203.0.113.0/24 or 2001:DB8:1234:5678::/64. When you include non-zero bits after the masked bits, they are not considered for the condition. AWS recommends that you use the new address included in the message.

- [IP address condition operators \(p. 1286\)](#)
- [IAM JSON policy elements: Condition \(p. 1278\)](#)

Suggestion – Null with qualifier

In the AWS Management Console, the finding for this check includes the following message:

```
Null with qualifier: Avoid using the Null condition operator with the ForAllValues or  
ForAnyValue qualifiers because they always return a true or false respectively.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Avoid using the Null condition operator with the ForAllValues or  
ForAnyValue qualifiers because they always return a true or false respectively."
```

Resolving the suggestion

In the Condition element, you build expressions in which you use condition operators like equal or less than to compare a condition in the policy against keys and values in the request context. For requests that include multiple values for a single condition key, you must use the ForAllValues or ForAnyValue set operators.

When you use the Null condition operator with ForAllValues, the statement always returns true. When you use the Null condition operator with ForAnyValue, the statement always returns false. AWS recommends that you use the StringLike condition operator with these set operators.

Related terms

- [Multivalued context keys \(p. 1294\)](#)
- [Null condition operator \(p. 1290\)](#)
- [Condition element \(p. 1278\)](#)

Suggestion – Private IP address subset

In the AWS Management Console, the finding for this check includes the following message:

Private IP address subset: The values for condition key aws:SourceIp include a mix of private and public IP addresses. The private addresses will not have the desired effect. aws:SourceIp works only for public IP address ranges. To define permissions for private IP ranges, use aws:VpcSourceIp.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The values for condition key aws:SourceIp include a mix of private and public IP addresses. The private addresses will not have the desired effect. aws:SourceIp works only for public IP address ranges. To define permissions for private IP ranges, use aws:VpcSourceIp."
```

Resolving the suggestion

The global condition key aws:SourceIp works only for public IP address ranges.

When your Condition element includes a mix of private and public IP addresses, the statement might not have the desired effect. You can specify private IP addresses using aws:VpcSourceIP.

Note

The global condition key aws:VpcSourceIP matches only if the request originates from the specified IP address and it goes through a VPC endpoint.

- [aws:SourceIp global condition key \(p. 1363\)](#)
- [aws:VpcSourceIp global condition key \(p. 1366\)](#)
- [IP address condition operators \(p. 1286\)](#)
- [IAM JSON policy elements: Condition \(p. 1278\)](#)

Suggestion – Private NotIpAddress subset

In the AWS Management Console, the finding for this check includes the following message:

Private NotIpAddress subset: The values for condition key aws:SourceIp include a mix of private and public IP addresses. The private addresses have no effect. aws:SourceIp works only for public IP address ranges. To define permissions for private IP ranges, use aws:VpcSourceIp.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The values for condition key aws:SourceIp include a mix of private and public IP addresses. The private addresses have no effect. aws:SourceIp works only for public IP address ranges. To define permissions for private IP ranges, use aws:VpcSourceIp."
```

Resolving the suggestion

The global condition key `aws:SourceIp` works only for public IP address ranges.

When your Condition element includes the `NotIpAddress` condition operator and a mix of private and public IP addresses, the statement might not have the desired effect. Every public IP addresses that is not specified in the policy will match. No private IP addresses will match. To achieve this effect, you can use `NotIpAddress` with `aws:VpcSourceIP` and specify the private IP addresses that should not match.

- [aws:Sourcelp global condition key \(p. 1363\)](#)
- [aws:VpcSourcelp global condition key \(p. 1366\)](#)
- [IP address condition operators \(p. 1286\)](#)
- [IAM JSON policy elements: Condition \(p. 1278\)](#)

Suggestion – Redundant action

In the AWS Management Console, the finding for this check includes the following message:

Redundant action: The {{redundantActionCount}} action(s) are redundant because they provide similar permissions. Update the policy to remove the redundant action such as: {{redundantAction}}.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The {{redundantActionCount}} action(s) are redundant because they provide similar permissions. Update the policy to remove the redundant action such as: {{redundantAction}}."
```

Resolving the suggestion

When you use wildcards (*) in the Action element, you can include redundant permissions. AWS recommends that you review your policy and include only the permissions that you need. This can help you remove redundant actions.

For example, the following actions include the `iam:GetCredentialReport` action twice.

```
"Action": [  
    "iam:Get*",  
    "iam>List*",  
    "iam:GetCredentialReport"  
,
```

In this example, permissions are defined for every IAM action that begins with Get or List. When IAM adds additional get or list operations, this policy will allow them. You might want to allow all of these read-only actions. The `iam:GetCredentialReport` action is already included as part of `iam:Get*`. To remove the duplicate permissions, you could remove `iam:GetCredentialReport`.

You receive a finding for this policy check when all of the contents of an action are redundant. In this example, if the element included `iam:*CredentialReport`, it is not considered redundant. That includes `iam:GenerateCredentialReport`, which is redundant, and `iam:GenerateCredentialReport`, which is not. Removing either `iam:Get*` or `iam:*CredentialReport` would change the policy's permissions.

- [IAM JSON policy elements: Action \(p. 1273\)](#)

AWS managed policies with this suggestion

[AWS managed policies \(p. 495\)](#) enable you to get started with AWS by assigning permissions based on general AWS use cases.

Redundant actions do not affect the permissions granted by the policy. When using an AWS managed policy as a reference to create your customer managed policy, AWS recommends that you remove redundant actions from your policy.

Suggestion – Redundant condition value num

In the AWS Management Console, the finding for this check includes the following message:

Redundant condition value num: Multiple values in {{operator}} are redundant. Replace with the {{greatest/least}} single value for {{key}}.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

"findingDetails": "Multiple values in {{operator}} are redundant. Replace with the {{greatest/least}} single value for {{key}}."

Resolving the suggestion

When you use numeric condition operators for similar values in a condition key, you can create an overlap that results in redundant permissions.

For example, the following Condition element includes multiple aws:MultiFactorAuthAge conditions that have an age overlap of 1200 seconds.

```
"Condition": {  
    "NumericLessThan": {  
        "aws:MultiFactorAuthAge": [  
            "2700",  
            "3600"  
        ]  
    }  
}
```

In this example, the permissions are defined if multi-factor authentication (MFA) was completed less than 3600 seconds (1 hour) ago. You could remove the redundant 2700 value.

- [Numeric condition operators \(p. 1284\)](#)
- [IAM JSON policy elements: Condition \(p. 1278\)](#)

Suggestion – Redundant resource

In the AWS Management Console, the finding for this check includes the following message:

Redundant resource: The {{redundantResourceCount}} resource ARN(s) are redundant because they reference the same resource. Review the use of wildcards (*)

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The {{redundantResourceCount}} resource ARN(s) are redundant because they reference the same resource. Review the use of wildcards (*)"
```

Resolving the suggestion

When you use wildcards (*) in Amazon Resource Names (ARNs), you can create redundant resource permissions.

For example, the following Resource element includes multiple ARNs with redundant permissions.

```
"Resource": [  
    "arn:aws:iam::111122223333:role/jane-admin",  
    "arn:aws:iam::111122223333:role/jane-s3only",  
    "arn:aws:iam::111122223333:role/jane*"  
,
```

In this example, the permissions are defined for any role with a name starting with jane. You could remove the redundant jane-admin and jane-s3only ARNs without changing the resulting permissions. This does make the policy dynamic. It will define permissions for any future roles that begin with jane. If the intention of the policy is to allow access to a static number of roles, then remove the last ARN and list only the ARNs that should be defined.

- [IAM JSON policy elements: Resource \(p. 1276\)](#)

AWS managed policies with this suggestion

[AWS managed policies \(p. 495\)](#) enable you to get started with AWS by assigning permissions based on general AWS use cases.

Redundant resources do not affect the permissions granted by the policy. When using an AWS managed policy as a reference to create your customer managed policy, AWS recommends that you remove redundant resources from your policy.

Suggestion – Redundant statement

In the AWS Management Console, the finding for this check includes the following message:

```
Redundant statement: The statements are redundant because they provide identical permissions. Update the policy to remove the redundant statement.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The statements are redundant because they provide identical permissions. Update the policy to remove the redundant statement."
```

Resolving the suggestion

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.

When you include the same statement more than once in a long policy, the statements are redundant. You can remove one of the statements without affecting the permissions granted by the policy. When someone edits a policy, they might change one of the statements without updating the duplicate. This might result in more permissions than intended.

- [IAM JSON policy elements: Statement \(p. 1262\)](#)

Suggestion – Wildcard in service name

In the AWS Management Console, the finding for this check includes the following message:

Wildcard in service name: Avoid using wildcards (*, ?) in the service name because it might grant unintended access to other AWS services with similar names.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

"findingDetails": "Avoid using wildcards (*, ?) in the service name because it might grant unintended access to other AWS services with similar names."

Resolving the suggestion

When you include the name of an AWS service in a policy, AWS recommends that you do not include wildcards (*, ?). This might add permissions for future services that you do not intend. For example, there are more than a dozen AWS services with the word *code* in their name.

"Resource": "arn:aws:*code*:111122223333:*

- [IAM JSON policy elements: Resource \(p. 1276\)](#)

Suggestion – Allow with unsupported tag condition key for service

In the AWS Management Console, the finding for this check includes the following message:

Allow with unsupported tag condition key for service: Using the effect Allow with the tag condition key {{conditionKeyName}} and actions for services with the following prefixes does not affect the policy: {{serviceNames}}. Actions for the listed service are not allowed by this statement. We recommend that you move these actions to a different statement without this condition key.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

"findingDetails": "Using the effect Allow with the tag condition key {{conditionKeyName}} and actions for services with the following prefixes does not affect the policy: {{serviceNames}}. Actions for the listed service are not allowed by this statement. We recommend that you move these actions to a different statement without this condition key."

Resolving the suggestion

Using unsupported tag condition keys in the Condition element of a policy with "Effect": "Allow" does not affect the permissions granted by the policy, because the condition is ignored for that service action. AWS recommends that you remove the actions for services that don't support the condition key and create another statement to allow access to specific resources in that service.

If you use the aws:ResourceTag condition key and it's not supported by a service action, then the key is not included in the request context. In this case, the condition in the Allow statement always returns false and the action is never allowed. This happens even if the resource is tagged correctly.

When a service supports the `aws:ResourceTag` condition key, you can use tags to control access to that service's resources. This is known as [attribute-based access control \(ABAC\) \(p. 15\)](#). Services that don't support these keys require you to control access to resources using [resource-based access control \(RBAC\) \(p. 16\)](#).

Note

Some services allow support for the `aws:ResourceTag` condition key for a subset of their resources and actions. IAM Access Analyzer returns findings for the service actions that are not supported. For example, Amazon S3 supports `aws:ResourceTag` for a subset of its resources. To view all of the resource types available in Amazon S3 that support the `aws:ResourceTag` condition key, see [Resource types defined by Amazon S3](#) in the Service Authorization Reference.

For example, assume that you want to allow team members to view details for specific resources that are tagged with the key-value pair `team=BumbleBee`. Also assume that AWS Lambda allows you to tag resources, but doesn't support the `aws:ResourceTag` condition key. To allow view actions for AWS App Mesh and AWS Backup if this tag is present, use the `aws:ResourceTag` condition key. For Lambda, use a resource naming convention that includes the team name as a prefix. Then include a separate statement that allows viewing resources with that naming convention.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowViewSupported",  
            "Effect": "Allow",  
            "Action": [  
                "appmesh:DescribeMesh",  
                "backup:GetBackupPlan"  
            ],  
            "Resource": "*",  
            "Condition": {  
                "StringEquals": {  
                    "aws:ResourceTag/team": "BumbleBee"  
                }  
            }  
        },  
        {  
            "Sid": "AllowViewUnsupported",  
            "Effect": "Allow",  
            "Action": "lambda:GetFunction",  
            "Resource": "arn:aws:lambda:*:123456789012:function:team-BumbleBee*"  
        }  
    ]  
}
```

Warning

Do not use the Not [version of the condition operator \(p. 1281\)](#) with "Effect": "Allow" as a workaround for this finding. These condition operators provide negated matching. This means that after the condition is evaluated, the result is negated. In the previous example, including the `lambda:GetFunction` action in the `AllowViewSupported` statement with the `StringNotEquals` operator always allows the action, regardless of whether the resource is tagged.

Do not use the [...IfExists \(p. 1288\)](#) version of the condition operator as a workaround for this finding. This means "Allow the action if the key is present in the request context and the values match. Otherwise, allow the action." In the previous example, including the `lambda:GetFunction` action in the `AllowViewSupported` statement with the `StringEqualsIfExists` operator always allows the action. For that action, the key is not present in the context, and every attempt to view that resource type is allowed, regardless of whether the resource is tagged.

Related terms

- [Global condition keys \(p. 1338\)](#)
- [IAM JSON policy elements: Condition operators \(p. 1281\)](#)
- [Condition element \(p. 1278\)](#)
- [Overview of JSON policies \(p. 490\)](#)

Suggestion – Allow NotAction with unsupported tag condition key for service

In the AWS Management Console, the finding for this check includes the following message:

Allow NotAction with unsupported tag condition key for service: Using the effect Allow with NotAction and the tag condition key {{conditionKeyName}} allows only service actions that support the condition key. The condition key doesn't apply to some service actions. We recommend that you use Action instead of NotAction.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Using the effect Allow with NotAction and the tag condition key {{conditionKeyName}} allows only service actions that support the condition key. The condition key doesn't apply to some service actions. We recommend that you use Action instead of NotAction."
```

Resolving the suggestion

Using unsupported tag condition keys in the Condition element of a policy with the element NotAction and "Effect": "Allow" does not affect the permissions granted by the policy. The condition is ignored for service actions that don't support the condition key. AWS recommends that you rewrite the logic to allow a list of actions.

If you use the aws:ResourceTag condition key with NotAction, any new or existing service actions that don't support the key are not allowed. AWS recommends that you explicitly list the actions that you want to allow. IAM Access Analyzer returns a separate finding for listed actions that don't support the aws:ResourceTag condition key. For more information, see [Suggestion – Allow with unsupported tag condition key for service \(p. 1149\)](#).

When a service supports the aws:ResourceTag condition key, you can use tags to control access to that service's resources. This is known as [attribute-based access control \(ABAC\) \(p. 15\)](#). Services that don't support these keys require you to control access to resources using [resource-based access control \(RBAC\) \(p. 16\)](#).

Related terms

- [Global condition keys \(p. 1338\)](#)
- [Comparing ABAC to RBAC \(p. 16\)](#)
- [IAM JSON policy elements: Condition operators \(p. 1281\)](#)
- [Condition element \(p. 1278\)](#)
- [Overview of JSON policies \(p. 490\)](#)

Suggestion – Recommended condition key for service principal

In the AWS Management Console, the finding for this check includes the following message:

Recommended condition key for service principal: To restrict access to the service principal {{servicePrincipalPrefix}} operating on your behalf, we recommend aws:SourceArn or aws:SourceAccount instead of {{key}}.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "To restrict access to the service principal {{servicePrincipalPrefix}} operating on your behalf, we recommend aws:SourceArn or aws:SourceAccount instead of {{key}}."
```

Resolving the suggestion

You can specify AWS services in the Principal element of a resource-based policy using a *service principal*, which is an identifier for the service. You should use the aws:SourceAccount or aws:SourceArn condition keys when granting access to service principals instead of other condition keys, such as aws:Referer. This helps you prevent a security issue called the *confused deputy problem*.

Related terms

- [Service principals](#)
- [AWS global condition keys: aws:SourceAccount](#)
- [AWS global condition keys: aws:SourceArn](#)
- [The confused deputy problem](#)

Suggestion – Irrelevant condition key in policy

In the AWS Management Console, the finding for this check includes the following message:

Irrelevant condition key in policy: The condition key {{condition-key}} is not relevant for the {{resource-type}} policy. Use this key in an identity-based policy to govern access to this resource.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The condition key {{condition-key}} is not relevant for the {{resource-type}} policy. Use this key in an identity-based policy to govern access to this resource."
```

Resolving the suggestion

Some condition keys aren't relevant for resource-based policies. For example, the s3:ResourceAccount condition key isn't relevant for the resource-based policy attached to an Amazon S3 bucket or Amazon S3 access point resource type.

You should use the condition key in an identity-based policy to control access to the resource.

Related terms

- [Identity-based policies and resource-based policies \(p. 511\)](#)

Suggestion – Redundant principal in role trust policy

In the AWS Management Console, the finding for this check includes the following message:

Redundant principal in role trust policy: The assumed-role principal {{redundant_principal}} is redundant with its parent role {{parent_role}}. Remove the assumed-role principal.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The assumed-role principal {{redundant_principal}} is redundant with its parent role {{parent_role}}. Remove the assumed-role principal."
```

Resolving the suggestion

If you specify both an assumed-role principal and its parent role in the Principal element of a policy, it does not allow or deny any different permissions. For example, it is redundant if you specify the Principal element using the following format:

```
"Principal": {  
    "AWS": [  
        "arn:aws:iam::AWS-account-ID:role/rolename",  
        "arn:aws:iam::AWS-account-ID:assumed-role/rolename/rolesessionname"  
    ]  
}
```

We recommend removing the assumed-role principal.

Related terms

- [Role session principals](#)

Suggestion – Confirm audience claim type

In the AWS Management Console, the finding for this check includes the following message:

Confirm audience claim type: The 'aud' (audience) claim key identifies the recipients that the JSON web token is intended for. Audience claims can be multivalued or single-valued. If the claim is multivalued, use a ForAllValues or ForAnyValue qualifier. If the claim is single-valued, do not use a qualifier.

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The 'aud' (audience) claim key identifies the recipients that the JSON web token is intended for. Audience claims can be multivalued or single-valued. If the claim is multivalued, use a ForAllValues or ForAnyValue qualifier. If the claim is single-valued, do not use a qualifier."
```

Resolving the suggestion

The aud (audience) claim key is a unique identifier for your app that is issued to you when you register your app with the IdP and identifies the recipients that the JSON web token is intended for. Audience claims can be multivalued or single-valued. If the claim is multivalued, use a ForAllValues or ForAnyValue condition set operator. If the claim is single-valued, do not use a condition set operator.

Related terms

- [Creating a role for web identity or OpenID Connect Federation \(console\)](#)

- [Multivalued context keys](#)
- [Single-valued vs. multivalued condition keys](#)

IAM Access Analyzer policy generation

As an administrator or developer, you might grant permissions to IAM entities (users or roles) beyond what they require. IAM provides several options to help you refine the permissions that you grant. One option is to generate an IAM policy that is based on access activity for an entity. IAM Access Analyzer reviews your AWS CloudTrail logs and generates a policy template that contains the permissions that the entity used in your specified date range. You can use the template to create a policy with fine-grained permissions that grant only the permissions that are required to support your specific use case.

Topics

- [How policy generation works \(p. 1154\)](#)
- [Service and action-level information \(p. 1154\)](#)
- [Things to know about generating policies \(p. 1155\)](#)
- [Permissions required to generate a policy \(p. 1155\)](#)
- [Generate a policy based on CloudTrail activity \(console\) \(p. 1157\)](#)
- [Generate a policy using AWS CloudTrail data in another account \(p. 1160\)](#)
- [Generate a policy based on CloudTrail activity \(AWS CLI\) \(p. 1162\)](#)
- [Generate a policy based on CloudTrail activity \(AWS API\) \(p. 1162\)](#)
- [IAM Access Analyzer policy generation and IAM action last accessed support \(p. 1163\)](#)

How policy generation works

IAM Access Analyzer analyzes your CloudTrail events to identify actions and services that have been used by an IAM entity (user or role). It then generates an IAM policy that is based on that activity. You can refine an entity's permissions when you replace a broad permissions policy attached to the entity with the generated policy. The following is a high-level overview of the policy generation process.

- **Set up for policy template generation** – You specify a time period of up to 90 days for IAM Access Analyzer to analyze your historical AWS CloudTrail events. You must specify an existing service role or create a new one. The service role gives IAM Access Analyzer access to your CloudTrail trail and service last accessed information to identify the services and actions that were used. You must specify the CloudTrail trail that is logging events for the account before you can generate a policy. For more information about IAM Access Analyzer quotas for CloudTrail data, see [IAM Access Analyzer quotas](#).
- **Generate policy** – IAM Access Analyzer generates a policy based on the access activity in your CloudTrail events.
- **Review and customize policy** – After the policy is generated, you can review the services and actions that were used by the entity during the specified date range. You can further customize the policy by adding or removing permissions, specifying resources, and adding conditions to the policy template.
- **Create and attach policy** – You have the option to save the generated policy by creating a managed policy. You can attach the policy that you create to the user or role whose activity was used to generate the policy.

Service and action-level information

When IAM Access Analyzer generates an IAM policy, information is returned to help you to further customize the policy. Two categories of information can be returned when a policy is generated:

- **Policy with action-level information** – For some AWS services, such as Amazon EC2, IAM Access Analyzer can identify the actions found in your CloudTrail events and lists the actions used in the policy it generates. For a list of supported services, see [IAM Access Analyzer policy generation and IAM action last accessed support \(p. 1163\)](#). For some services, IAM Access Analyzer prompts you to add actions for the services to the generated policy.
- **Policy with service-level information** – IAM Access Analyzer uses [last accessed](#) information to create a policy template with all of the recently used services. When using the AWS Management Console, we prompt you to review the services and add actions to complete the policy.

For a list of actions in each service, see [Actions, Resources, and Condition Keys for AWS Services](#) in the Service Authorization Reference.

Things to know about generating policies

Before you generate a policy, review the following important details.

- **Enable a CloudTrail trail** – You must have a CloudTrail trail enabled for your account to generate a policy based on access activity. When you create a CloudTrail trail, CloudTrail sends events related to your trail to an Amazon S3 bucket that you specify. To learn how to create a CloudTrail trail, see [Creating a trail for your AWS account](#) in the *AWS CloudTrail User Guide*.
- **Data events not available** – IAM Access Analyzer does not identify action-level activity for data events, such as Amazon S3 data events, in generated policies.
- **PassRole** – The `iam:PassRole` action is not tracked by CloudTrail and is not included in generated policies.
- **Reduce policy generation time** – To generate a policy faster, reduce the date range that you specify during setup for policy generation.
- **Use CloudTrail for auditing** – Do not use policy generation for auditing purposes; use CloudTrail instead. For more information about using CloudTrail, see [Logging IAM and AWS STS API calls with AWS CloudTrail](#).
- **Denied actions** – Policy generation reviews all CloudTrail events, including denied actions.
- **One policy IAM console** – You can have one generated policy at a time in the IAM console.
- **Generated policy availability IAM console** – You can review a generated policy in the IAM console for up to 7 days after it is generated. After 7 days, you must generate a new policy.
- **Policy generation quotas** – For additional information about IAM Access Analyzer policy generation quotas, see [IAM Access Analyzer quotas](#).
- **Amazon S3 standard rates apply** – When you use the policy generation feature, IAM Access Analyzer reviews CloudTrail logs in your S3 bucket. There are no additional storage charges to access your CloudTrail logs for policy generation. AWS charges standard Amazon S3 rates for requests and data transfer of CloudTrail logs stored in your S3 bucket.
- **AWS Control Tower support** – Policy generation does not support AWS Control Tower for generating policies.

Permissions required to generate a policy

The permissions that you need to generate a policy for the first time differ from those that you need to generate a policy for subsequent uses.

First-time setup

When you generate a policy for the first time, you must choose a suitable existing [service role](#) in your account or create a new service role. The service role gives IAM Access Analyzer access to CloudTrail and service last accessed information in your account. Only administrators should have the permissions

necessary to create and configure roles. Therefore, we recommend that an administrator creates the service role during the first-time setup. To learn more about the permissions required to create service roles, see [Creating a role to delegate permissions to an AWS service](#).

Permissions required for service role

When you create a service role, you configure two policies for the role. You attach an *IAM permissions policy* to the role that specifies what the role can do. You also attach a *role trust policy* to the role that specifies the principal who can use the role.

The first example policy shows the permissions policy for the service role that is required to generate a policy. The second example policy shows the role trust policy that is required for the service role. You can use these policies to help you create a service role when you use the AWS API or AWS CLI to generate a policy. When you use the IAM console to create a service role as part of the policy generation process, we generate these policies for you.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "cloudtrail:GetTrail",  
            "Resource": "*"  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "iam:GetServiceLastAccessedDetails",  
                "iam:GenerateServiceLastAccessedDetails"  
            ],  
            "Resource": "*"  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:GetObject",  
                "s3>ListBucket"  
            ],  
            "Resource": [  
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET",  
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*"  
            ]  
        }  
    ]  
}
```

The following example policy shows the role trust policy with the permissions that allows IAM Access Analyzer to assume the role.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "access-analyzer.amazonaws.com"  
            },  
            "Action": "sts:AssumeRole"  
        }  
    ]  
}
```

Subsequent uses

To generate policies in the AWS Management Console, an IAM user must have a permissions policy that allows them to pass the service role that is used for policy generation to IAM Access Analyzer. `iam:PassRole` is usually 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 `AccessAnalyzerMonitorServiceRole*`. To learn more about passing IAM roles to AWS services, see [Granting a user permissions to pass a role to an AWS service](#).

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowUserToPassRole",  
            "Effect": "Allow",  
            "Action": [  
                "iam:GetRole",  
                "iam:PassRole"  
            ],  
            "Resource": "arn:aws:iam::123456789012:role/service-role/  
AccessAnalyzerMonitorServiceRole*"  
        }  
    ]  
}
```

You must also have the following IAM Access Analyzer permissions to generate policies in the AWS Management Console, AWS API, or AWS CLI as shown in the following policy statement.

```
{  
    "Sid": "AllowUserToGeneratePolicy",  
    "Effect": "Allow",  
    "Action": [  
        "access-analyzer:CancelPolicyGeneration",  
        "access-analyzer:GetGeneratedPolicy",  
        "access-analyzer>ListPolicyGenerations",  
        "access-analyzer:StartPolicyGeneration"  
    ],  
    "Resource": "*"  
}
```

For both first-time and subsequent uses

When you use the AWS Management Console to generate a policy, you must have `cloudtrail>ListTrails` permission to list the CloudTrail trails in your account as shown in the following policy statement.

```
{  
    "Sid": "AllowUserToListTrails",  
    "Effect": "Allow",  
    "Action": [  
        "CloudTrail>ListTrails"  
    ],  
    "Resource": "*"  
}
```

Generate a policy based on CloudTrail activity (console)

You can generate a policy for an IAM user or role.

Step 1: Generate a policy based on CloudTrail activity

The following procedure explains how to generate a policy for a role using the AWS Management Console.

Generate a policy for an IAM 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 on the left, choose **Roles**.

Note

The steps to generate a policy based on activity for an IAM user are almost identical. To do this, choose **Users** instead of **Roles**.

3. In the list of roles in your account, choose the name of the role whose activity you want to use to generate a policy.
4. On the **Permissions** tab, in the **Generate policy based on CloudTrail events** section, choose **Generate policy**.
5. On the **Generate policy** page, specify the time period that you want IAM Access Analyzer to analyze your CloudTrail events for actions taken with the role. You can choose a range of up to 90 days. We recommend that you choose the shortest time period possible to reduce the policy generation time.
6. In the **CloudTrail access** section, choose a suitable existing role or create a new role if a suitable role does not exist. The role gives IAM Access Analyzer permissions to access your CloudTrail data on your behalf to review access activity to identify the services and actions that have been used. To learn more about the permissions required for this role, see [Permissions required to generate a policy \(p. 1155\)](#).
7. In the **CloudTrail trail to be analyzed** section, specify the CloudTrail trail that logs events for the account.

If you choose a CloudTrail trail that stores logs in a different account, an information box about cross-account access is displayed. Cross-account access requires additional set up. To learn more, see [Choose a role for cross-account access](#) later in this topic.

8. Choose **Generate policy**.
9. While policy generation is in progress, you are returned to the **Roles Summary** page on the **Permissions** tab. Wait until the status in the **Policy request details** section displays **Success**, and then choose **View generated policy**. You can view the generated policy for up to seven days. If you generate another policy, the existing policy is replaced with the new one that you generate.

Step 2: Review permissions and add actions for services used

Review the services and actions that IAM Access Analyzer identified that the role used. You can add actions for any services that were used to the generated policy template.

1. Review the following sections:
 - On the **Review permissions** page, review the list of **Actions included in the generated policy**. The list displays the services *and* actions that IAM Access Analyzer identified were used by the role in the specified date range.
 - The **Services used** section displays additional services that IAM Access Analyzer identified that were used by the role in the specified date range. Information about which actions were used might not be available for the services listed in this section. Use the menus for each service listed to manually choose the actions that you want to include in the policy.
2. When you are done adding actions, choose **Next**.

Step 3: Further customize the generated policy

You can further customize the policy by adding or removing permissions or specifying resources.

To customize the generated policy

1. Update the policy template. The policy template contains resource ARN placeholders for actions that support resource-level permissions, as shown in the following image. *Resource-level permissions* refers to the ability to specify which resources users are allowed to perform actions on. We recommend that you use [ARNs](#) to specify your individual resources in the policy for actions that support resource-level permissions. You can replace the placeholder resource ARNs with valid resource ARNs for your use case.

If an action does not support resource-level permissions, you must use a wildcard (*) to specify that all resources can be affected by the action. To learn which AWS services support resource-level permissions, see [AWS services that work with IAM](#). For a list of actions in each service, and to learn which actions support resource-level permissions, see [Actions, Resources, and Condition Keys for AWS Services](#).

Generated policy

Customize permissions

Review the following policy template. You must specify resources for actions that support resource-level permissions to continue creating the policy.

```
1 * {
2     "Version": "2012-10-17",
3     "Statement": [
4         {
5             "Effect": "Allow",
6             "Action": [
7                 "access-analyzer:ValidatePolicy",
8                 "iam:GetAccountPasswordPolicy",
9                 "iam:GetAccountSummary",
10                "iam:ListAccountAliases",
11                "iam:ListGroups",
12                "iam:ListPolicies",
13                "iam:ListRoles",
14                "iam:ListUsers"
15            ],
16            "Resource": "*"
17        },
18        {
19            "Effect": "Allow",
20            "Action": [
21                "iam:GetRole",
22                "iam:ListAttachedRolePolicies",
23                "iam:ListInstanceProfilesForRole",
24                "iam:ListRolePolicies",
25                "iam:ListRoleTags"
26            ],
27            "Resource": "arn:aws:iam:${Account}:role/${RoleNameWithPath}"
28        },
29        {
30            "Effect": "Allow",
31            "Action": [
32                "iam: GetUser",
33                "iam:ListAccessKeys",
34                "iam:ListAttachedUserPolicies",
35                "iam:ListGroupsForUser",
36                "iam:ListUserTags"
37            ],
38            "Resource": "arn:aws:iam:${Account}:user/${UserNameWithPath}"
39        }
40    ]
41}
```

Edit statement

Select a statement

Select an existing statement in the policy or add a new statement.

+ Add new statement

2. (Optional) Add, modify, or remove JSON policy statements in the template. To learn more about writing JSON policies, see [Creating IAM policies \(console\)](#).
 3. When you are done customizing the policy template, you have the following options:
 - (Optional) You can copy the JSON in the template to use separately outside of the **Generated policy** page. For example, if you want to use the JSON to create a policy in a different account. If the policy in your template exceeds the 6,144 character limit for JSON policies, the policy is split into multiple policies.
 - Choose **Next** to review and create a managed policy in the same account.

Step 4: Review and create a managed policy

If you have permissions to create and attach IAM policies, you can create a managed policy from the policy that was generated. You can then attach the policy to a user or role in your account.

To review and create a policy

1. On the **Review and create managed policy** page, enter a **Name** and **Description** (optional) for the policy that you are creating.

2. (Optional) In the **Summary** section, you can review the permissions that will be included in the policy.
3. (Optional) Add metadata to the policy by attaching tags as key-value pairs. For more information about using tags in IAM, see [Tagging IAM resources](#).
4. When you are finished, do one of the following:
 - You can attach the new policy directly to the role that was used to generate the policy. To do this, near the bottom of the page, select the check box next to the **Attach policy to YourRoleName**. Then choose **Create and attach policy**.
 - Otherwise, choose **Create policy**. You can find the policy that you created in the list of policies in the **Policies** navigation pane of the IAM console.
5. You can attach the policy that you created to an entity in your account. After you attach the policy, you can remove any other overly broad policies that might be attached to the entity. To learn how to attach a managed policy, see [Adding IAM identity permissions \(console\)](#).

Generate a policy using AWS CloudTrail data in another account

You might create CloudTrail trails that store data in central accounts to streamline governing activities. For example, you can use AWS Organizations to create a trail that logs all events for all of the AWS accounts in that organization. The trail belongs to a central account. If you want to generate a policy for a user or role in an account that is different from the account where your CloudTrail log data is stored, you must grant cross-account access. To do this, you need both a role and a bucket policy that grant IAM Access Analyzer permissions to your CloudTrail logs. For more information about creating Organizations trails, see [Creating a trail for an organization](#).

In this example, assume that you want to generate a policy for a user or role in account A. The CloudTrail trail in account A stores CloudTrail logs in a bucket in account B. Before you can generate a policy, you must make the following updates:

1. Choose an existing role, or create a new service role that grants IAM Access Analyzer access to the bucket in account B (where your CloudTrail logs are stored).
2. Verify your Amazon S3 bucket object ownership and bucket permissions policy in account B so that IAM Access Analyzer can access objects in the bucket.

Step 1: Choose or create a role for cross-account access

- On the **Generate policy** screen, the option to **Use an existing role** is pre-selected for you if a role with the required permissions exists in your account. Otherwise, choose **Create and use a new service role**. The new role is used to grant IAM Access Analyzer access to your CloudTrail logs in account B.

Step 2: Verify or update your Amazon S3 bucket configuration in account B

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the name of the bucket where your CloudTrail trail logs are stored.
3. Choose the **Permissions** tab and go to the **Object Ownership** section.

Use Amazon S3 Object Ownership bucket settings to control ownership of objects that you upload to your buckets. By default, when other AWS accounts upload objects to your bucket, the uploading account owns the objects. To generate a policy, the bucket owner must own all of the objects in the

bucket. Depending on your ACL use case, you might need to change the **Object Ownership** setting for your bucket. Set **Object Ownership** to one of the following options.

- **Bucket owner enforced** (recommended)
- **Bucket owner preferred**

Important

To successfully generate a policy, the objects in the bucket must be owned by the bucket owner. If you choose to use **Bucket owner preferred**, you can only generate a policy for the time period after the object ownership change was made.

To learn more about object ownership in Amazon S3, see [Controlling ownership of objects and disabling ACLs for your bucket](#) in the *Amazon S3 User Guide*.

4. Add permissions to your Amazon S3 bucket policy in account B to allow access for the role in account A.

The following example policy allows ListBucket and GetObject for the bucket named *DOC-EXAMPLE-BUCKET*. It allows access if the role accessing the bucket belongs to an account in your organization and has a name that starts with AccessAnalyzerMonitorServiceRole. Using [aws:PrincipalArn](#) as a Condition in the Resource element ensures that the role can only access activity for the account if it belongs to account A. You can replace *DOC-EXAMPLE-BUCKET* with your bucket name and *organization-id* with your organization ID.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "PolicyGenerationBucketPolicy",  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": "*"  
            },  
            "Action": [  
                "s3:GetObject",  
                "s3>ListBucket"  
            ],  
            "Resource": [  
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET",  
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET/organization-id/AWSLogs/  
${aws:PrincipalAccount}/*"  
            ],  
            "Condition": {  
                "StringEquals": {  
                    "aws:PrincipalOrgID": "organization-id"  
                },  
                "StringLike": {  
                    "aws:PrincipalArn": "arn:aws:iam::${aws:PrincipalAccount}:role/service-role/  
AccessAnalyzerMonitorServiceRole*"  
                }  
            }  
        }  
    ]  
}
```

5. If you encrypt your logs using AWS KMS, update your AWS KMS key policy in the account where you store the CloudTrail logs to grant IAM Access Analyzer access to use your key, as shown in the following policy example. Replace CROSS_ACCOUNT_ORG_TRAIL_FULL_ARN with the ARN for your trail and *organization-id* with your organization ID.

```
{
```

```
"Version": "2012-10-17",
"Statement": [
    {
        "Effect": "Allow",
        "Principal": {
            "AWS": "*"
        },
        "Action": "kms:Decrypt",
        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "kms:EncryptionContext:aws:cloudtrail:arn": "$CROSS_ACCOUNT_ORG_TRAIL_FULL_ARN",
                "aws:PrincipalOrgID": "organization-id"
            },
            "StringLike": {
                "kms:ViaService": "access-analyzer.*.amazonaws.com",
                "aws:PrincipalArn": "arn:aws:iam::${aws:PrincipalAccount}:role/service-role/AccessAnalyzerMonitorServiceRole"
            }
        }
    }
]
```

Generate a policy based on CloudTrail activity (AWS CLI)

You can use the following commands to generate a policy using the AWS CLI.

To generate a policy

- [aws accessanalyzer start-policy-generation](#)

To view a generated policy

- [aws accessanalyzer get-generated-policy](#)

To cancel a policy generation request

- [aws accessanalyzer cancel-policy-generation](#)

To view a list of policy generation requests

- [aws accessanalyzer list-policy-generations](#)

Generate a policy based on CloudTrail activity (AWS API)

You can use the following operations to generate a policy using the AWS API.

To generate a policy

- [StartPolicyGeneration](#)

To view a generated policy

- [GetGeneratedPolicy](#)

To cancel a policy generation request

- [CancelPolicyGeneration](#)

To view a list of policy generation requests

- [ListPolicyGenerations](#)

IAM Access Analyzer policy generation and IAM action last accessed support

The following table lists the AWS services for which IAM Access Analyzer generates policies with action-level information and the services that support [IAM action last accessed information](#). For a list of actions in each service, see [Actions, resources, and condition keys for AWS services](#) in the Service Authorization Reference.

Service	Service prefix	Policy generation action-level information	Action last accessed information
AWS IAM Access Analyzer	access-analyzer	✓ Yes	✗ No
AWS Account Management	account	✓ Yes	✗ No
Amazon Managed Workflows for Apache Airflow	airflow	✓ Yes	✗ No
Amazon MQ	mq	✓ Yes	✗ No
AWS Amplify	amplify	✓ Yes	✗ No
AWS Amplify UI Builder	amplifyuibuilder	✓ Yes	✗ No
Amazon AppIntegrations	app-integrations	✓ Yes	✗ No
Amazon AppFlow	appflow	✓ Yes	✗ No
AWS Application Cost Profiler Service	application-cost-profiler	✓ Yes	✗ No
AWS AppSync	appsync	✓ Yes	✗ No
Amazon Managed Service for Prometheus	aps	✓ Yes	✗ No
Amazon Athena	athena	✓ Yes	✗ No

Service	Service prefix	Policy generation action-level information	Action last accessed information
AWS Marketplace	aws-marketplace	✓ Yes	✗ No
AWS Batch	batch	✓ Yes	✗ No
Amazon Braket	braket	✓ Yes	✗ No
AWS Cloud9	cloud9	✓ Yes	✗ No
AWS CloudFormation	cloudformation	✓ Yes	✗ No
Amazon CloudSearch	cloudsearch	✓ Yes	✗ No
AWS CloudTrail	cloudtrail	✓ Yes	✗ No
AWS CodeArtifact	codeartifact	✓ Yes	✗ No
Amazon CodeGuru Profiler	codeguru-profiler	✓ Yes	✗ No
Amazon CodeGuru Reviewer	codeguru-reviewer	✓ Yes	✗ No
AWS CodeStar	codestar	✓ Yes	✗ No
AWS CodeStar Notifications	codestar-notifications	✓ Yes	✗ No
Amazon Cognito Identity	cognito-identity	✓ Yes	✗ No
Amazon Cognito User Pools	cognito-idp	✓ Yes	✗ No
Amazon Cognito Sync	cognito-sync	✓ Yes	✗ No
AWS Compute Optimizer	compute-optimizer	✓ Yes	✗ No
Amazon Connect	connect	✓ Yes	✗ No
AWS Cost and Usage Report	cur	✓ Yes	✗ No
AWS Glue DataBrew	databrew	✓ Yes	✗ No
AWS Data Exchange	dataexchange	✓ Yes	✗ No
AWS Data Pipeline	datapipeline	✓ Yes	✗ No
DynamoDB Accelerator	dax	✓ Yes	✗ No
AWS Device Farm	devicefarm	✓ Yes	✗ No

Service	Service prefix	Policy generation action-level information	Action last accessed information
Amazon DevOps Guru	devops-guru	✓ Yes	✗ No
AWS Direct Connect	directconnect	✓ Yes	✗ No
Amazon Data Lifecycle Manager	dlm	✓ Yes	✗ No
AWS Database Migration Service	dms	✓ Yes	✗ No
AWS Directory Service	ds	✓ Yes	✗ No
Amazon DynamoDB	dynamodb	✓ Yes	✗ No
Amazon Elastic Block Store	ebs	✓ Yes	✗ No
Amazon Elastic Compute Cloud	ec2	✓ Yes	✓ Yes
Amazon Elastic Container Registry	ecr	✓ Yes	✗ No
Amazon Elastic Container Registry Public	ecr-public	✓ Yes	✗ No
Amazon Elastic Container Service	ecs	✓ Yes	✗ No
Amazon Elastic Kubernetes Service	eks	✓ Yes	✗ No
Amazon Elastic Inference	elastic-inference	✓ Yes	✗ No
AWS Elastic Beanstalk	elasticbeanstalk	✓ Yes	✗ No
Amazon Elastic File System	elasticfilesystem	✓ Yes	✗ No
Elastic Load Balancing	elasticloadbalancing	✓ Yes	✗ No
Amazon Elastic Transcoder	elastictranscoder	✓ Yes	✗ No
Amazon EMR on EKS (EMR Containers)	emr-containers	✓ Yes	✗ No
Amazon OpenSearch Service	es	✓ Yes	✗ No
Amazon CloudWatch Evidently	evidently	✓ Yes	✗ No
Amazon FinSpace	finspace	✓ Yes	✗ No
Amazon Kinesis Firehose	firehose	✓ Yes	✗ No
AWS Fault Injection Simulator	fis	✓ Yes	✗ No
AWS Firewall Manager	fms	✓ Yes	✗ No

Service	Service prefix	Policy generation action-level information	Action last accessed information
Amazon Location	geo	✓ Yes	✗ No
Amazon Managed Grafana	grafana	✓ Yes	✗ No
AWS IoT Greengrass	greengrass	✓ Yes	✗ No
AWS Ground Station	groundstation	✓ Yes	✗ No
Amazon GuardDuty	guardduty	✓ Yes	✗ No
AWS HealthLake	healthlake	✓ Yes	✗ No
AWS Identity and Access Management	iam	✓ Yes	✓ Yes
AWS Identity Store	identitystore	✓ Yes	✗ No
EC2 Image Builder	imagebuilder	✓ Yes	✗ No
Amazon Inspector Classic	inspector	✓ Yes	✗ No
Amazon Inspector	inspector2	✓ Yes	✗ No
AWS IoT Core Device Advisor	iotdeviceadvisor	✓ Yes	✗ No
AWS IoT Fleet Hub	iotfleethub	✓ Yes	✗ No
AWS IoT TwinMaker	iottwinmaker	✓ Yes	✗ No
Amazon Interactive Video Service	ivs	✓ Yes	✗ No
Amazon Managed Streaming for Apache Kafka	kafka	✓ Yes	✗ No
Amazon Managed Streaming for Kafka Connect	kafkaconnect	✓ Yes	✗ No
Amazon Kinesis	kinesis	✓ Yes	✗ No
AWS Key Management Service	kms	✓ Yes	✗ No
AWS Lambda	lambda	✓ Yes	✓ Yes
Amazon Lightsail	lightsail	✓ Yes	✗ No
Amazon CloudWatch Logs	logs	✓ Yes	✗ No
Amazon Lookout for Equipment	lookoutequipment	✓ Yes	✗ No
Amazon Lookout for Metrics	lookoutmetrics	✓ Yes	✗ No
Amazon Lookout for Vision	lookoutvision	✓ Yes	✗ No

Service	Service prefix	Policy generation action-level information	Action last accessed information
Amazon Managed Blockchain	managedblockchain	✓ Yes	✗ No
AWS Elemental MediaConnect	mediaconnect	✓ Yes	✗ No
AWS Elemental MediaConvert	mediaconvert	✓ Yes	✗ No
AWS Elemental MediaLive	medialive	✓ Yes	✗ No
AWS Elemental MediaTailor	mediatailor	✓ Yes	✗ No
Amazon MemoryDB for Redis	memorydb	✓ Yes	✗ No
AWS Application Migration Service	mgn	✓ Yes	✗ No
AWS Migration Hub	mgh	✓ Yes	✗ No
AWS Migration Hub Strategy Recommendations	migrationhub-strategy	✓ Yes	✗ No
Amazon CloudWatch	cloudwatch	✓ Yes	✗ No
AWS Network Manager	networkmanager	✓ Yes	✗ No
Amazon Nimble Studio	nimble	✓ Yes	✗ No
AWS OpsWorks	opsworks	✓ Yes	✗ No
AWS Outposts	outposts	✓ Yes	✗ No
AWS Panorama	panorama	✓ Yes	✗ No
AWS Performance Insights	pi	✓ Yes	✗ No
Amazon Pinpoint	mobiletargeting	✓ Yes	✗ No
Amazon Polly	polly	✓ Yes	✗ No
Amazon Connect Customer Profiles	profile	✓ Yes	✗ No
Amazon QLDB	qldb	✓ Yes	✗ No
AWS Resource Access Manager	ram	✓ Yes	✗ No
AWS Recycle Bin	rbin	✓ Yes	✗ No
Amazon Relational Database Service	rds	✓ Yes	✗ No
Amazon Redshift Data API	redshift-data	✓ Yes	✗ No

Service	Service prefix	Policy generation action-level information	Action last accessed information
AWS Resource Groups	resource-groups	✓ Yes	✗ No
Amazon Route 53 Recovery Controls	route53-recovery-control-config	✓ Yes	✗ No
Amazon Route 53 Recovery Readiness	route53-recovery-readiness	✓ Yes	✗ No
Amazon Route 53 Resolver	route53resolver	✓ Yes	✗ No
AWS CloudWatch RUM	rum	✓ Yes	✗ No
Amazon S3	s3	✓ Yes	✓ Yes
Amazon S3 on Outposts	s3-outposts	✓ Yes	✗ No
Savings Plans	savingsplans	✓ Yes	✗ No
Amazon EventBridge Schemas	schemas	✓ Yes	✗ No
Amazon SimpleDB	sdb	✓ Yes	✗ No
AWS Secrets Manager	secretsmanager	✓ Yes	✗ No
AWS Security Hub	securityhub	✓ Yes	✗ No
AWS Cloud Map	servicediscovery	✓ Yes	✗ No
Service Quotas	servicequotas	✓ Yes	✗ No
Amazon Simple Email Service	ses	✓ Yes	✗ No
AWS Shield	shield	✓ Yes	✗ No
AWS Signer	signer	✓ Yes	✗ No
AWS Server Migration Service	sms	✓ Yes	✗ No
Amazon Pinpoint SMS and Voice Service	sms-voice	✓ Yes	✗ No
AWS Snowball	snowball	✓ Yes	✗ No
Amazon Simple Queue Service	sq	✓ Yes	✗ No
AWS Systems Manager	ssm	✓ Yes	✗ No

Service	Service prefix	Policy generation action-level information	Action last accessed information
AWS Systems Manager Incident Manager	ssm-incidents	✓ Yes	✗ No
AWS Security Token Service	sts	✓ Yes	✗ No
Amazon Simple Workflow Service	swf	✓ Yes	✗ No
Amazon CloudWatch Synthetics	synthetics	✓ Yes	✗ No
Amazon Resource Group Tagging API	tag	✓ Yes	✗ No
Amazon Textract	textract	✓ Yes	✗ No
Amazon Timestream	timestream	✓ Yes	✗ No
Amazon Transcribe	transcribe	✓ Yes	✗ No
Amazon Translate	translate	✓ Yes	✗ No
AWS Well-Architected Tool	wellarchitected	✓ Yes	✗ No
Amazon Connect Wisdom	wisdom	✓ Yes	✗ No
Amazon WorkLink	worklink	✓ Yes	✗ No

IAM Access Analyzer quotas

IAM 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 ¹
Maximum archive rules per analyzer	100 Each archive rule can have up to 20 values per criterion.	1,000 ¹
Maximum number of access previews per analyzer per hour	1,000	1,000
AWS CloudTrail log files processed per policy generations	100,000	100,000
Concurrent policy generations	1	1

Resource	Default quota	Maximum quota
Policy generation AWS CloudTrail data size	25 GB	25 GB
Policy generation AWS CloudTrail time range	90 days	90 days
Policy generations per day	Africa (Cape Town): 5 Asia Pacific (Hong Kong): 5 Europe (Milan): 5 Middle East (Bahrain): 5 All other supported regions: 50 Note Canceled policy generation requests apply to the daily quota.	Africa (Cape Town): 5 Asia Pacific (Hong Kong): 5 Europe (Milan): 5 Middle East (Bahrain): 5 All other supported regions: 50

¹Some quotas are customer-configurable using [Service Quotas](#).

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 \(p. 1171\)](#)
- [Troubleshooting access denied error messages \(p. 1174\)](#)
- [Troubleshooting IAM policies \(p. 1180\)](#)
- [Troubleshooting FIDO security keys \(p. 1194\)](#)
- [Troubleshooting IAM roles \(p. 1195\)](#)
- [Troubleshooting IAM and Amazon EC2 \(p. 1201\)](#)
- [Troubleshooting IAM and Amazon S3 \(p. 1204\)](#)
- [Troubleshooting SAML 2.0 federation with AWS \(p. 1204\)](#)

Troubleshooting general IAM issues

Use the information here to help you diagnose and fix common issues when you work with AWS Identity and Access Management (IAM).

Issues

- [I can't sign in to my AWS account \(p. 1171\)](#)
- [I lost my access keys \(p. 1171\)](#)
- [Policy variables aren't working \(p. 1172\)](#)
- [Changes that I make are not always immediately visible \(p. 1172\)](#)
- [I am not authorized to perform: iam>DeleteVirtualMFADevice \(p. 1172\)](#)
- [How do I securely create IAM users? \(p. 1173\)](#)
- [Additional resources \(p. 1173\)](#)

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 sign-in issues](#) in the *AWS Sign-In User Guide*.

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 \(p. 1220\)](#), 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 \(p. 112\)](#).

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 \(p. 1261\)](#). To learn more about policy versions, see [the section called "Versioning IAM policies" \(p. 606\)](#).

- Verify that your policy variables are in the right case. For details, see [IAM policy elements: Variables and tags \(p. 1298\)](#).

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), including tags used in [attribute-based access control \(ABAC\)](#), 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 User 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 a virtual MFA device for the user in IAM but never assigns it to the user. You must delete the existing virtual MFA device before you can create a new virtual MFA device with the same device name.

To fix this issue, an administrator should **not** edit policy permissions. Instead, the administrator must use the AWS CLI or AWS API to delete the existing but unassigned virtual MFA device.

To delete an existing but unassigned virtual 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 MFA device for the user you are trying to fix.
3. Delete the virtual MFA 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 IAM Identity Center 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 \(p. 533\)](#).

Additional resources

The following resources can help you troubleshoot as you work with AWS.

- [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. For more information, see [Logging IAM and AWS STS API calls with AWS CloudTrail \(p. 471\)](#).

- [AWS Knowledge Center](#) – Find FAQs and links to other resources to help you troubleshoot issues.
- [AWS Support Center](#) – Get technical support.
- [AWS Premium Support Center](#) – Get premium technical support.

Troubleshooting access denied error messages

Access denied errors appear when AWS explicitly or implicitly denies an authorization request. An *explicit denial* occurs when a policy contains a Deny statement for the specific AWS action. An *implicit denial* occurs when there is no applicable Deny statement and also no applicable Allow statement. Because an IAM policy denies an IAM principal by default, the policy must explicitly allow the principal to perform an action. Otherwise, the policy implicitly denies access. For more information, see [The difference between explicit and implicit denies \(p. 1316\)](#).

If multiple policies of the same policy type deny an authorization request, then AWS doesn't specify the number of policies in the access denied error message. If multiple policy types deny an authorization request, AWS includes only one of those policy types in the error message.

I get "access denied" when I make a request to an AWS service

- Check if the error message includes the type of policy responsible for denying access. For example, if the error mentions that access is denied due to a Service Control Policy (SCP), then you can focus on troubleshooting SCP issues. When you know the policy type, you can also check for a deny statement or a missing allow on the specific action in policies of that policy type. For more information, see [Troubleshooting access denied error messages \(p. 1174\)](#). If the error message doesn't mention the policy type responsible for denying access, use the rest of the guidelines in this section to troubleshoot further.
- 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 \(p. 581\)](#).
- If the AWS Management Console returns a message stating that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator provided you with your sign-in credentials or sign-in link.

The following example error occurs when the mateojackson IAM user attempts to use the console to view details about a fictional `my-example-widget` resource but does not have the fictional widgets:`GetWidget` permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:  
widgets:GetWidget on resource: my-example-widget
```

In this case, Mateo must ask his administrator to update his policies to allow access to the `my-example-widget` resource using the widgets:`GetWidget` action.

- Are you trying to access a service that supports [resource-based policies \(p. 511\)](#), 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 \(p. 1224\)](#).

- If your policy includes a condition with a key–value pair, review it carefully. Examples include the [aws:RequestTag/tag-key \(p. 1338\)](#) 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 \(p. 1278\)](#).
- If you have a [permissions boundary \(p. 501\)](#), 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 \(p. 1306\)](#).
- 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 \(p. 1174\)](#).
- Verify that the service accepts temporary security credentials, see [AWS services that work with IAM \(p. 1224\)](#).
- 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 \(p. 438\)](#).
- Verify that your temporary security credentials haven't expired. For more information, see [Temporary security credentials in IAM \(p. 426\)](#).
- 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 \(p. 441\)](#).
- If you assumed a role, your role session might be limited by session policies. When you [request temporary security credentials \(p. 428\)](#) programmatically using AWS STS, you can optionally pass inline or managed [session policies \(p. 487\)](#). 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 \(p. 433\)](#). 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 \(p. 487\)](#).

- 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/*"]  
    }]  
}
```

Access denied error message examples

Most access denied error messages appear in the format User *user* is not authorized to perform *action* on *resource* because *context*. In this example, *user* is the [Amazon Resource Name \(ARN\)](#) that doesn't receive access, *action* is the service action that the policy denies, and *resource* is the ARN of the resource on which the policy acts. The *context* field represents additional context about the policy type that explains why the policy denied access.

When a policy explicitly denies access because the policy contains a Deny statement, then AWS includes the phrase with an explicit deny in a *type* policy in the access denied error message. When the policy implicitly denies access, then AWS includes the phrase because no *type* policy allows the *action* action in the access denied error message.

Note

Some AWS services do not support this access denied error message format. The content of access denied error messages can vary depending on the service making the authorization request.

The following examples show the format for different types of access denied error messages.

Access denied due to a Service Control Policy – implicit denial

- Check for a missing Allow statement for the action in your Service Control Policies (SCPs). For the following example, the action is codecommit>ListRepositories.
- Update your SCP by adding the Allow statement. For more information, see [Updating an SCP](#) in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*.

```
User: arn:aws:iam::777788889999:user/JohnDoe is not authorized to perform:  
codecommit>ListRepositories because no service control policy allows the  
codecommit>ListRepositories action
```

Access denied due to a Service Control Policy – explicit denial

- Check for a Deny statement for the action in your Service Control Policies (SCPs). For the following example, the action is codecommit>ListRepositories.
- Update your SCP by removing the Deny statement. For more information, see [Updating an SCP](#) in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*.

```
User: arn:aws:iam::777788889999:user/JohnDoe is not authorized to perform:
```

```
codecommit:ListRepositories with an explicit deny in a service control policy
```

Access denied due to a VPC endpoint policy – implicit denial

1. Check for a missing Allow statement for the action in your Virtual Private Cloud (VPC) endpoint policies. For the following example, the action is `codecommit:ListRepositories`.
2. Update your VPC endpoint policy by adding the Allow statement. For more information, see [Update a VPC endpoint policy](#) in the *AWS PrivateLink Guide*.

```
User: arn:aws:iam::123456789012:user/JohnDoe is not authorized to perform:  
codecommit:ListRepositories because no VPC endpoint policy allows the  
codecommit:ListRepositories action
```

Access denied due to a VPC endpoint policy – explicit denial

1. Check for an explicit Deny statement for the action in your Virtual Private Cloud (VPC) endpoint policies. For the following example, the action is `codedeploy>ListDeployments`.
2. Update your VPC endpoint policy by removing the Deny statement. For more information, see [Update a VPC endpoint policy](#) in the *AWS PrivateLink Guide*.

```
User: arn:aws:iam::123456789012:user/JohnDoe is not authorized to perform:  
codedeploy>ListDeployments on resource: arn:aws:codedeploy:us-  
east-1:123456789012:deploymentgroup:* with an explicit deny in a VPC endpoint policy
```

Access denied due to a permissions boundary – implicit denial

1. Check for a missing Allow statement for the action in your permissions boundary. For the following example, the action is `codedeploy>ListDeployments`.
2. Update your permissions boundary by adding the Allow statement to your IAM policy. For more information, see [Permissions boundaries for IAM entities \(p. 501\)](#) and [Editing IAM policies \(p. 609\)](#).

```
User: arn:aws:iam::123456789012:user/JohnDoe is not authorized to perform:  
codedeploy>ListDeployments on resource: arn:aws:codedeploy:us-  
east-1:123456789012:deploymentgroup:* because no permissions boundary allows the  
codedeploy>ListDeployments action
```

Access denied due to a permissions boundary – explicit denial

1. Check for an explicit Deny statement for the action in your permissions boundary. For the following example, the action is `sagemaker>ListModels`.
2. Update your permissions boundary by removing the Deny statement from your IAM policy. For more information, see [Permissions boundaries for IAM entities \(p. 501\)](#) and [Editing IAM policies \(p. 609\)](#).

```
User: arn:aws:iam::777788889999:user/JohnDoe is not authorized to perform:  
sagemaker>ListModels with an explicit deny in a permissions boundary
```

Access denied due to session policies – implicit denial

1. Check for a missing Allow statement for the action in your session policies. For the following example, the action is `codecommit>ListRepositories`.
2. Update your session policy by adding the Allow statement. For more information, see [Session policies](#) and [Editing IAM policies \(p. 609\)](#).

```
User: arn:aws:iam::123456789012:user/JohnDoe is not authorized to perform:  
codecommit>ListRepositories because no session policy allows the  
codecommit>ListRepositories action
```

Access denied due to session policies – explicit denial

1. Check for an explicit Deny statement for the action in your session policies. For the following example, the action is `codedeployListDeployments`.
2. Update your session policy by removing the Deny statement. For more information, see [Session policies](#) and [Editing IAM policies \(p. 609\)](#).

```
User: arn:aws:iam::123456789012:user/JohnDoe is not authorized to perform:  
codedeployListDeployments on resource: arn:aws:codedeploy:us-  
east-1:123456789012:deploymentgroup:* with an explicit deny in a sessions policy
```

Access denied due to resource-based policies – implicit denial

1. Check for a missing Allow statement for the action in your resource-based policy. For the following example, the action is `secretsmanagerGetSecretValue`.
2. Update your policy by adding the Allow statement. For more information, see [Resource-based policies](#) and [Editing IAM policies \(p. 609\)](#).

```
User: arn:aws:iam::123456789012:user/JohnDoe is not authorized to perform:  
secretsmanagerGetSecretValue because no resource-based policy allows the  
secretsmanagerGetSecretValue action
```

Access denied due to resource-based policies – explicit denial

1. Check for an explicit Deny statement for the action in your resource-based policy. For the following example, the action is `secretsmanagerGetSecretValue`.
2. Update your policy by removing the Deny statement. For more information, see [Resource-based policies](#) and [Editing IAM policies \(p. 609\)](#).

```
User: arn:aws:iam::123456789012:user/JohnDoe is not authorized to perform:  
secretsmanagerGetSecretValue on resource: arn:aws:secretsmanager:us-  
east-1:123456789012:secret:* with an explicit deny in a resource-based policy
```

Access denied due to role trust policies – implicit denial

1. Check for a missing Allow statement for the action in your role trust policy. For the following example, the action is `stsAssumeRole`.

2. Update your policy by adding the Allow statement. For more information, see [Resource-based policies](#) and [Editing IAM policies \(p. 609\)](#).

```
User: arn:aws:iam::123456789012:user/JohnDoe is not authorized to perform:  
sts:AssumeRole because no role trust policy allows the sts:AssumeRole action
```

Access denied due to role trust policies – explicit denial

1. Check for an explicit Deny statement for the action in your role trust policy. For the following example, the action is sts:AssumeRole.
2. Update your policy by removing the Deny statement. For more information, see [Resource-based policies](#) and [Editing IAM policies \(p. 609\)](#).

```
User: arn:aws:iam::777788889999:user/JohnDoe is not authorized to perform:  
sts:AssumeRole with an explicit deny in the role trust policy
```

Access denied due to identity-based policies – implicit denial

1. Check for a missing Allow statement for the action in identity-based policies attached to the identity. For the following example, the action is codecommit>ListRepositories attached to user JohnDoe.
2. Update your policy by adding the Allow statement. For more information, see [Identity-based policies](#) and [Editing IAM policies \(p. 609\)](#).

```
User: arn:aws:iam::123456789012:user/JohnDoe is not authorized to perform:  
codecommit>ListRepositories because no identity-based policy allows the  
codecommit>ListRepositories action
```

Access denied due to identity-based policies – explicit denial

1. Check for an explicit Deny statement for the action in identity-based policies attached to the identity. For the following example, the action is codedeploy>ListDeployments attached to user JohnDoe.
2. Update your policy by removing the Deny statement. For more information, see [Identity-based policies](#) and [Editing IAM policies \(p. 609\)](#).

```
User: arn:aws:iam::123456789012:user/JohnDoe is not authorized to perform:  
codedeploy>ListDeployments on resource: arn:aws:codedeploy:us-  
east-1:123456789012:deploymentgroup:* with an explicit deny in an identity-based policy
```

Access denied when a VPC request fails due to another policy

1. Check for an explicit Deny statement for the action in your Service Control Policies (SCPs). For the following example, the action is SNS:Publish.
2. Update your SCP by removing the Deny statement. For more information, see [Updating an SCP](#) in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*.

```
User: arn:aws:sts::111122223333:assumed-role/role-name/role-session-name is not authorized  
to perform:
```

SNS:Publish on resource: arn:aws:sns:us-east-1:444455556666:*role-name-2*
with an explicit deny in a VPC endpoint policy transitively through a service control
policy

Troubleshooting IAM policies

A [policy \(p. 485\)](#) 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 both **Visual** and **JSON** editor options. 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 \(p. 1323\)](#).

Troubleshooting IAM Policy Topics

- [Troubleshoot using the visual editor \(p. 1181\)](#)
 - [Policy restructuring \(p. 1181\)](#)
 - [Choosing a resource ARN in the visual editor \(p. 1181\)](#)
 - [Denying permissions in the visual editor \(p. 1182\)](#)
 - [Specifying multiple services in the visual editor \(p. 1182\)](#)
 - [Reducing the size of your policy in the visual editor \(p. 1182\)](#)
 - [Fixing unrecognized services, actions, or resource types in the visual editor \(p. 1183\)](#)
- [Troubleshoot using policy summaries \(p. 1184\)](#)
 - [Missing policy summary \(p. 1184\)](#)
 - [Policy summary includes unrecognized services, actions, or resource types \(p. 1184\)](#)
 - [Service does not support IAM policy summaries \(p. 1185\)](#)
 - [My policy does not grant the expected permissions \(p. 1186\)](#)
- [Troubleshoot policy management \(p. 1189\)](#)
 - [Attaching or detaching a policy in an IAM account \(p. 1190\)](#)
 - [Changing policies for your IAM identities based on their activity \(p. 1190\)](#)
- [Troubleshoot JSON policy documents \(p. 1190\)](#)
 - [Validate your policies \(p. 1190\)](#)
 - [I don't have permissions for policy validation in the JSON editor \(p. 1190\)](#)
 - [More than one JSON policy object \(p. 1190\)](#)
 - [More than one JSON statement element \(p. 1191\)](#)
 - [More than one effect, action, or resource element in a JSON statement element \(p. 1192\)](#)
 - [Missing JSON version element \(p. 1193\)](#)

Troubleshoot using the visual editor

When you create or edit a customer managed policy, you can use information in the **Visual** editor 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" \(p. 515\)](#).

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** editor. If you never make any changes in the **Visual** editor and choose **Next** from the **JSON** editor, the policy is less likely to be restructured. However, if you create a policy and use the **Visual** editor to make any modifications, or if you choose **Next** from the **Visual** editor option, 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:

Using this editor option	If you edit your policy	And then choose Next from this tab	When you choose Save changes
Visual	Edited	Visual	The policy is restructured
Visual	Edited	JSON	The policy is restructured
Visual	Not Edited	Visual	The policy is restructured
JSON	Edited	Visual	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 \(p. 520\)](#), 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 \(ARN\) \(p. 1213\)](#). 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 \(p. 1276\)](#).

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 \(p. 1306\)](#).

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 \(p. 1181\)](#) 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** editor option.

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** editor option. 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 limits \(p. 1221\)](#).

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 \(p. 589\)](#).

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** editor option 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** editor option 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** editor option 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 \(p. 1185\)](#).

- **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** editor option 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 \(p. 1185\)](#).

- **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. To avoid this, we recommend that you 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 **Next** 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 \(p. 635\)](#), the [service summary \(p. 645\)](#), and the [action summary \(p. 649\)](#). 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 \(p. 634\)](#) for any policies that are attached to an entity on the **Policy details** page for that policy. 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 \(p. 1260\)](#):
 - 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 \(p. 1260\)](#).

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 \(p. 589\)](#).

Policy summary includes unrecognized services, actions, or resource types

In the IAM console, if a [policy summary \(p. 634\)](#) 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 \(list of services\) \(p. 635\)](#).

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 \(p. 589\)](#).

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 \(p. 1185\)](#).
- **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 \(list of actions\) \(p. 645\)](#).
- **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 \(list of actions\) \(p. 645\)](#).
- **Typo** – AWS checks that the JSON is syntactically correct, and that the policy does not include typos or other errors as part of [policy validation \(p. 588\)](#).

Note

As a [best practice \(p. 1032\)](#), we recommend that you use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions. We recommend that you open your existing policies and review and resolve any policy validation recommendations.

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 **Feedback for IAM** 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 **Feedback for IAM** 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 \(p. 1260\)](#).

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 \(p. 1338\)](#) 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 \(p. 635\)](#) 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 \(p. 1186\)](#)
- [A resource is defined without an applicable action \(p. 1187\)](#)
- [A condition is defined without an applicable action \(p. 1188\)](#)

To view examples of policy summaries that include warnings, see [the section called “Policy summary \(list of services\)” \(p. 635\)](#).

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 \(p. 1213\)](#) in the policy's

[Resource \(p. 1276\)](#) 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 \(p. 1224\)](#).

```
{
    "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: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 \(p. 185\)](#) for that service. In the IAM console, when you view the **Policy details** 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 \(p. 1035\)](#) 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.

Validate your policies

When you create or edit a JSON policy, IAM can perform policy validation to help you create an effective policy. IAM identifies JSON syntax errors, while IAM Access Analyzer provides additional policy checks with recommendations to help you further refine your policies. To learn more about policy validation, see [Validating IAM policies \(p. 588\)](#). To learn more about IAM Access Analyzer policy checks and actionable recommendations, see [IAM Access Analyzer policy validation](#).

I don't have permissions for policy validation in the JSON editor

In the AWS Management Console, you might receive the following error if you do not have permissions to view IAM Access Analyzer policy validation results:

You need permissions. You do not have the permissions required to perform this operation. Ask your administrator to add permissions.

To fix this error, ask your administrator to add the `access-analyzer:ValidatePolicy` permission for you.

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. We recommend that you review any [correct any policy validation \(p. 588\)](#) recommendations for your policies.

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 \(p. 1261\)](#). To learn more about policy versions, see [the section called "Versioning IAM policies" \(p. 606\)](#).

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 FIDO security keys

Use the information here to help you diagnose common issues that you might encounter when working with FIDO2 security keys.

Topics

- [I can't enable my FIDO security key \(p. 1194\)](#)
- [I can't sign in using my FIDO security key \(p. 1195\)](#)
- [I lost or broke my FIDO security key \(p. 1195\)](#)
- [Other issues \(p. 1195\)](#)

I can't enable my FIDO 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 FIDO security key, check the following:

- Are you using a supported configuration?

For information on devices and browsers you can use with WebAuthn and AWS, see [Supported configurations for using FIDO security keys \(p. 125\)](#).

- Are you using Mozilla Firefox?

Current Firefox versions support WebAuthn by default. To enable support for WebAuthn 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 **webauthn**.
 3. Choose **security.webauthn.webauthn** and change its value to **true**.
- Are you using any browser plugins?

AWS does not support the use of plugins to add WebAuthn browser support. Instead, use a browser that offers native support of the WebAuthn standard.

Even if you're using a supported browser, you may have a plugin that is incompatible with WebAuthn. An incompatible plugin may prevent you from enabling and using your FIDO-compliant security key. You should disable any plugins that might be incompatible and restart your browser. Then retry enabling the FIDO 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 FIDO security keys despite using a supported configuration, make sure they have the appropriate permissions. For a detailed example, see [IAM tutorial: Permit users to manage their credentials and MFA settings \(p. 66\)](#).

I can't sign in using my FIDO security key

If you're an IAM user and you can't sign in to the AWS Management Console using your FIDO security key, first see [Supported configurations for using FIDO security keys \(p. 125\)](#). If you're using a supported configuration but cannot sign in, contact your system administrator for assistance.

I lost or broke my FIDO security key

Up to **eight** MFA devices of any combination of the [currently supported MFA types](#) can be assigned to a user. With multiple MFA devices, you only need one MFA device to sign in to the AWS Management Console. Replacing a FIDO security key is similar to replacing a hardware TOTP token. 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? \(p. 143\)](#).

Other issues

If you have an issue with FIDO 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 \(p. 1196\)](#)

- [A new role appeared in my AWS account \(p. 1197\)](#)
- [I can't edit or delete a role in my AWS account \(p. 1197\)](#)
- [I'm not authorized to perform: iam:PassRole \(p. 1198\)](#)
- [Why can't I assume a role with a 12-hour session? \(AWS CLI, AWS API\) \(p. 1198\)](#)
- [I receive an error when I try to switch roles in the IAM console \(p. 1198\)](#)
- [My role has a policy that allows me to perform an action, but I get "access denied" \(p. 1199\)](#)
- [The service did not create the role's default policy version \(p. 1199\)](#)
- [There is no use case for a service role in the console \(p. 1200\)](#)

I can't assume a role

Check the following:

- To allow users to assume the current role again within a role session, specify the role ARN or AWS account ARN as a principal in the role trust policy. AWS services that provide compute resources such as Amazon EC2, Amazon ECS, Amazon EKS, and Lambda provide temporary credentials and automatically rotate these credentials. This ensures that you always have a valid set of credentials. For these services, it's not necessary to assume the current role again to obtain temporary credentials. However, if you intend to pass [session tags \(p. 417\)](#) or a [session policy \(p. 487\)](#), you need to assume the current role again. To learn how to modify a role trust policy to add the principal role ARN or AWS account ARN, see [Modifying a role trust policy \(console\) \(p. 385\)](#).
- When you assume a role using the AWS Management Console, make sure to use the exact name of your role. Role names are case sensitive when you assume a role.
- When you assume a role using AWS STS API or AWS CLI, make sure to use the exact name of your role in the ARN. Role names are case sensitive when you assume a role.
- 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 IAM resources" \(p. 399\)](#).

```
"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": {
    "DateGreaterThanOrEqual": {"aws:CurrentTime": "2017-07-01T00:00:00Z"},
    "DateLessThan": {"aws:CurrentTime": "2017-12-31T23:59:59Z"}
}
```

- Source Identity** – Administrators can configure roles to require identities to pass a custom string that identifies the person or application that is performing actions in AWS, called *source identity*. Verify whether the role being assumed requires that a source identity is set. For more information about source identity, see [Monitor and control actions taken with assumed roles \(p. 444\)](#).

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 \(p. 185\)](#) 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 \(p. 242\)](#).

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 \(p. 1224\)](#) 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 \(p. 185\)](#) 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 \(p. 1224\)](#) 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 \(p. 1224\)](#). 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 \(p. 276\)](#).

If you use [role chaining \(p. 185\)](#) (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 \(p. 428\)](#) programmatically using AWS STS, you can optionally pass inline or managed [session policies \(p. 487\)](#). 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.amazonaws.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 \(console\) \(p. 610\)](#). For more information about policy versions, see [Versioning IAM policies \(p. 606\)](#).

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 **Entities attached** 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 using the JSON editor \(p. 583\)](#).
 - 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 \(p. 598\)](#).
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 \(p. 251\)](#) 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 \(p. 614\)](#).
9. [Delete the role \(p. 397\)](#).

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 \(p. 1269\)](#) 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 \(p. 1224\)](#).
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 \(p. 1269\)](#).

You can manually create a service role using [AWS CLI commands \(p. 258\)](#) or [AWS API operations \(p. 260\)](#). 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 \(p. 251\)](#).
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 \(p. 1201\)](#)
- [The credentials on my instance are for the wrong role \(p. 1202\)](#)
- [When I attempt to call the AddRoleToInstanceProfile, I get an AccessDenied error \(p. 1202\)](#)
- [Amazon EC2: When I attempt to launch an instance with a role, I get an AccessDenied error \(p. 1202\)](#)
- [I can't access the temporary security credentials on my EC2 instance \(p. 1202\)](#)
- [What do the errors from the info document in the IAM subtree mean? \(p. 1203\)](#)

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 \(p. 374\)](#). For information about adding permissions to a user, see [Managing IAM policies \(p. 581\)](#).

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 \(p. 381\)](#).

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.

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.

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 \(p. 374\)](#). For information about adding permissions to a user, see [Managing IAM policies \(p. 581\)](#).

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 \(p. 250\)](#).

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 \(p. 374\)](#). For information about adding permissions to a user, see [Managing IAM policies \(p. 581\)](#).

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 \(p. 374\)](#).

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? \(p. 1203\)](#) 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.

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.

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.

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.

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 User 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. \(p. 1205\)](#)
- [Error: RoleSessionName is required in AuthnResponse \(service: AWSSecurityTokenService; status code: 400; error code: InvalidIdentityToken\) \(p. 1205\)](#)
- [Error: Not authorized to perform sts:AssumeRoleWithSAML \(service: AWSSecurityTokenService; status code: 403; error code: AccessDenied\) \(p. 1206\)](#)
- [Error: RoleSessionName in AuthnResponse must match \[a-zA-Z_0-9+=,.@-\]{2,64} \(service: AWSSecurityTokenService; status code: 400; error code: InvalidIdentityToken\) \(p. 1206\)](#)
- [Error: Source Identity must match \[a-zA-Z_0-9+=,.@-\]{2,64} and not begin with "aws:" \(service: AWSSecurityTokenService; status code: 400; error code: InvalidIdentityToken\) \(p. 1207\)](#)
- [Error: Response signature invalid \(service: AWSSecurityTokenService; status code: 400; error code: InvalidIdentityToken\) \(p. 1207\)](#)
- [Error: Failed to assume role: Issuer not present in specified provider \(service: AWSOpenIdDiscoveryService; status code: 400; error code: AuthSamlInvalidSamlResponseException\) \(p. 1207\)](#)
- [Error: Could not parse metadata. \(p. 1207\)](#)
- [Error: Specified provider doesn't exist. \(p. 1208\)](#)
- [Error: Requested DurationSeconds exceeds MaxSessionDuration set for this role. \(p. 1208\)](#)
- [Error: Response does not contain the required audience. \(p. 1208\)](#)
- [How to view a SAML response in your browser for troubleshooting \(p. 1208\)](#)

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 \(p. 225\)](#). To view the SAML response in your browser, follow the steps listed in [How to view a SAML response in your browser for troubleshooting \(p. 1208\)](#).

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 \(p. 225\)](#). To view the SAML response in your browser, follow the steps listed in [How to view a SAML response in your browser for troubleshooting \(p. 1208\)](#).

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 \(p. 226\)](#), your trust policy must also include the `sts:TagSession` action. For more information about session tags, see [Passing session tags in AWS STS \(p. 417\)](#).

This error can occur if you do not have `sts:SetSourceIdentity` permissions in your role trust policy. If your SAML assertion is configured to use the [SourceIdentity \(p. 228\)](#) attribute, then your trust policy must also include the `sts:SetSourceIdentity` action. For more information about source identity, see [Monitor and control actions taken with assumed roles \(p. 444\)](#).

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 \(p. 225\)](#). To view the SAML response in your browser, follow the steps listed in [How to view a SAML response in your browser for troubleshooting \(p. 1208\)](#).

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 \(p. 225\)](#). To view the SAML response in your browser, follow the steps listed in [How to view a SAML response in your browser for troubleshooting \(p. 1208\)](#).

Error: Source Identity must match [a-zA-Z_0-9+=,.@-]{2,64} and not begin with "aws :" (service: AWSecurityTokenService; status code: 400; error code: InvalidIdentityToken)

This error can occur if the `sourceIdentity` attribute value is too long or contains invalid characters. The maximum valid length is 64 characters. For more information about source identity, see [Monitor and control actions taken with assumed roles \(p. 444\)](#).

For more information about creating SAML assertions, see [Configuring SAML assertions for the authentication response \(p. 225\)](#). To view the SAML response in your browser, follow the steps listed in [How to view a SAML response in your browser for troubleshooting \(p. 1208\)](#).

Error: Response signature invalid (service: AWSecurityTokenService; 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-IAMSAMLProvider` 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 you do not format your metadata file properly.

When you [create or manage a SAML identity provider \(p. 219\)](#) in the AWS Management Console, you must retrieve the SAML metadata document from your identity provider.

This metadata file includes the issuer name, expiration information, and keys that can be used to validate the SAML authentication response (assertions) received from the IdP. The metadata file must be encoded in UTF-8 format without a byte order mark (BOM). To remove the BOM, you can encode the file as UTF-8 using a text editing tool, such as Notepad++.

The x.509 certificate included as part of the SAML metadata document must use a key size of at least 1024 bits. Also, the x.509 certificate must also be free of any repeated extensions. You can use extensions, but the extensions can only appear once in the certificate. If the x.509 certificate does not meet either condition, IdP creation fails and returns an "Unable to parse metadata" error.

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 \(p. 218\)](#).

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 \(p. 276\)](#).

Error: Response does not contain the required audience.

This error can occur if there is a mismatch between the audience URL and the identity provider in the SAML configuration. Make sure that your identity provider (IdP) relying party identifier exactly matches the audience URL (entity ID) provided in the SAML configuration.

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 \(p. 1208\)](#)
- [Mozilla Firefox \(p. 1209\)](#)
- [Apple Safari \(p. 1209\)](#)
- [What to do with the Base64-encoded SAML response \(p. 1209\)](#)

Google Chrome

To view a SAML response in Chrome

These steps were tested using version 106.0.5249.103 (Official Build) (arm64) of Google Chrome. If you use another version, you might need to adapt the steps accordingly.

1. Press **F12** to start the **Developer Tools** console.
2. Select the **Network** tab, and then select **Preserve log** in the upper left of the **Developer Tools** window.

3. Reproduce the issue.
4. (Optional) If the **Method** column is not visible in the **Developer Tools Network** log pane, right-click on any column label and choose **Method** to add the column.
5. Look for a **SAML Post** in the **Developer Tools Network** log pane. Select that row, and then view the **Payload** tab at the top. Look for the **SAMLResponse** element that contains the encoded request. The associated value is the Base64-encoded response.

Mozilla Firefox

To view a SAML response in Firefox

This procedure was tested on version 105.0.3 (64-bit) of Mozilla Firefox. If you use another version, you might need to adapt the steps accordingly.

1. Press **F12** to start the **Web Developer Tools** console.
2. Select the **Network** tab.
3. In the upper right of the **Web Developer Tools** window, choose options (the small gear icon). Select **Persist logs**.
4. Reproduce the issue.
5. (Optional) If the **Method** column is not visible in the **Web Developer Tools Network** log pane, right-click on any column label and choose **Method** to add the column.
6. Look for a **POST SAML** in the table. Select that row, and then view the **Request** tab and find the **SAMLResponse** element. The associated value is the Base64-encoded response.

Apple Safari

To view a SAML response in Safari

These steps were tested using version 16.0 (17614.1.25.9.10, 17614) of Apple Safari. 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. Choose **Develop** in the menu bar, then select **Show Web Inspector**.
3. Select the **Network** tab.
4. In the upper left of the **Web Inspector** window, choose options (the small circle icon containing three horizontal lines). Select **Preserve Log**.
5. (Optional) If the **Method** column is not visible in the **Web Inspector Network** log pane, right-click on any column label and choose **Method** to add the column.
6. Reproduce the issue.
7. Look for a **POST SAML** in the table. Select that row, and then view the Headers tab.
8. Look for the **SAMLResponse** element that contains the encoded request. Scroll down to find Request Data with the name **SAMLResponse**. The associated value is the Base64-encoded response.

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

- [Amazon Resource Names \(ARNs\) \(p. 1211\)](#)
- [IAM identifiers \(p. 1213\)](#)
- [IAM and AWS STS quotas \(p. 1220\)](#)
- [AWS services that work with IAM \(p. 1224\)](#)
- [Signing AWS API requests \(p. 1244\)](#)
- [IAM JSON policy reference \(p. 1260\)](#)

Amazon Resource Names (ARNs)

Amazon Resource Names (ARNs) uniquely identify AWS resources. We require an ARN when you need to specify a resource unambiguously across all of AWS, such as in IAM policies, Amazon Relational Database Service (Amazon RDS) tags, and API calls.

ARN format

The following are the general formats for ARNs. The specific formats depend on the resource. To use an ARN, replace the *italicized* text with the resource-specific information. Be aware that the ARNs for some resources omit the Region, the account ID, or both the Region and the account ID.

```
arn:partition:service:region:account-id:resource-id
arn:partition:service:region:account-id:resource-type/resource-id
arn:partition:service:region:account-id:resource-type:resource-id
```

partition

The partition in which the resource is located. A *partition* is a group of AWS Regions. Each AWS account is scoped to one partition.

The following are the supported partitions:

- aws - AWS Regions
- aws-cn - China Regions
- aws-us-gov - AWS GovCloud (US) Regions

service

The service namespace that identifies the AWS product.

region

The Region code. For example, us-east-2 for US East (Ohio). For the list of Region codes, see [Regional endpoints](#) in the *AWS General Reference*.

account-id

The ID of the AWS account that owns the resource, without the hyphens. For example, 123456789012.

resource-type

The resource type. For example, vpc for a virtual private cloud (VPC).

resource-id

The resource identifier. This is the name of the resource, the ID of the resource, or a [resource path \(p. 1212\)](#). Some resource identifiers include a parent resource (sub-resource-type/parent-resource/sub-resource) or a qualifier such as a version (resource-type:resource-name:qualifier).

Examples

IAM user

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

SNS topic

```
arn:aws:sns:us-east-1:123456789012:example-sns-topic-name
```

VPC

```
arn:aws:ec2:us-east-1:123456789012:vpc/vpc-0e9801d129EXAMPLE
```

Look up the ARN format for a resource

To look up the ARN format for a specific AWS resource, open the [Service Authorization Reference](#), open the page for the service, and navigate to the resource types table.

Paths in ARNs

Resource ARNs can include a path. For example, in Amazon S3, the resource identifier is an object name that can include slashes (/) to form a path. Similarly, IAM user names and group names can include paths.

Paths can include a wildcard character, namely an asterisk (*). For example, if you are writing an IAM policy, you can specify all IAM users that have the path product_1234 using a wildcard as follows:

```
arn:aws:iam::123456789012:user/Development/product_1234/*
```

Similarly, you can specify user/* to mean all users or group/* to mean all groups, as in the following examples:

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

The following example shows ARNs for an Amazon S3 bucket in which the resource name includes a path:

```
arn:aws:s3:::my_corporate_bucket/*  
arn:aws:s3:::my_corporate_bucket/Development/*
```

Incorrect wildcard usage

You cannot use a wildcard in the portion of the ARN that specifies the resource type, such as the term `user` in an IAM ARN. For example, the following is not allowed.

```
arn:aws:iam::123456789012:u*    <== not allowed
```

IAM identifiers

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

Topics

- [Friendly names and paths \(p. 1213\)](#)
- [IAM ARNs \(p. 1213\)](#)
- [Unique identifiers \(p. 1218\)](#)

Friendly names and paths

When you create a user, a role, a user 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 use the IAM API or AWS Command Line Interface (AWS CLI) to create IAM resources, you can add an optional path. You can use a single path, or nest multiple paths as a folder structure. For example, you could use the nested path `/division_abc/subdivision_xyz/product_1234/engineering/` to match your company 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 \(p. 567\)](#). For information about how a friendly name can be specified, see [the User API documentation](#). For additional examples of how you might use paths, see [IAM ARNs \(p. 1213\)](#).

When you use AWS CloudFormation to create resources, you can specify a path for users, user groups, and roles, and customer managed policies.

If you have a user and user group in the same path, IAM doesn't automatically put the user in that user group. For example, you might create a Developers user group and specify the path as `/division_abc/subdivision_xyz/product_1234/engineering/`. If you create a user named Bob and add the same path to him, this doesn't automatically put Bob in the Developers user group. IAM doesn't enforce any boundaries between users or user groups based on their paths. Users with different paths can use the same resources if 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 AWS STS quotas \(p. 1220\)](#).

IAM ARNs

Most resources have a friendly name for example, a user named Bob or a user 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 for the resource. 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 \(p. 528\)](#) between accounts in different partitions.

- `service` identifies the AWS product. IAM resources always use `iam`.
- `region` identifies the Region of the resource. For IAM resources, this is always kept blank.
- `account` specifies the AWS account ID with no hyphens.
- `resource` 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:root
arn:aws:iam::account:user/user-name-with-path
arn:aws:iam::account:group/group-name-with-path
arn:aws:iam::account:role/role-name-with-path
arn:aws:iam::account:policy/policy-name-with-path
arn:aws:iam::account:instance-profile/instance-profile-name-with-path
arn:aws:sts::account:federated-user/user-name
arn:aws:sts::account:assumed-role/role-name/role-session-name
arn:aws:iam::account:mfa/virtual-device-name-with-path
arn:aws:iam::account:u2f/u2f-token-id
arn:aws:iam::account:server-certificate/certificate-name-with-path
arn:aws:iam::account:saml-provider/provider-name
arn:aws:iam::account: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/RDSAccess
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:

Note

Each IAM user name is unique. The user name is case-insensitive for the user, such as during the sign in process, but is case-sensitive when you use it in a policy or as part of an ARN.

```
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 user group:

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

- An IAM user 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 \(p. 185\)](#):

```
arn:aws:iam::123456789012:role/aws-service-role/access-analyzer.amazonaws.com/AWSServiceRoleForAccessAnalyzer
```

- A [service role \(p. 185\)](#):

```
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 \(p. 1264\)](#) of a resource-based policy.

- The AWS account displays the following:

```
arn:aws:iam::123456789012:root
```

The following example shows a policy you could assign to Richard to allow him to manage his 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 \(p. 1298\)](#)

Using wildcards and paths in ARNs

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

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

If 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, user 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/*
```

If you specify the following ARN for a user `arn:aws:iam::111122223333:user/*` it matches both of the following examples.

```
arn:aws:iam::111122223333:user/JohnDoe
arn:aws:iam::111122223333:user/division_abc/subdivision_xyz/JaneDoe
```

But, if you specify the following ARN for a user `arn:aws:iam::111122223333:user/division_abc*` it matches the second example, but not the first.

```
arn:aws:iam::111122223333:user/JohnDoe
arn:aws:iam::111122223333:user/division_abc/subdivision_xyz/JaneDoe
```

Don't use a wildcard in the `user/`, `group/`, or `policy/` part of the ARN. For example, IAM does not allow the following:

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

Example Example use of paths and ARNs for a project-based user 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` user group creates a project-based user group within the `/marketing/` path. Jules assigns users from different parts of the company to the user group. This example illustrates that a user's path isn't related to the user groups the user is in.

The marketing group has a new product they'll be launching, so Jules creates a new user group in the `/marketing/` path called `Widget_Launch`. Jules then assigns the following policy to the user group, which gives the user group access to objects in 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 user 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, user group, role, policy, instance profile, or server certificate, it assigns a unique ID to each resource. The unique ID looks like this:

AIDAJQABLZS4A3QDU576Q

For the most part, you use friendly names and [ARNs \(p. 1213\)](#) 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 reuses friendly names in your AWS account. Within your account, a friendly name for a user, user group, role, or policy must be unique. For example, you might create an IAM user named John. Your company uses Amazon S3 and has a bucket with folders for each employee. IAM user John is a member of an IAM user group named User-S3-Access with permissions that allows users access only to their own folders in the bucket. For an example of how you might create an identity-based policy that allows IAM users to access their own bucket object in S3 using the friendly name of users, see [Amazon S3: Allows IAM users access to their S3 home directory, programmatically and in the console \(p. 578\)](#).

Suppose that the employee named John leaves your company and you delete the corresponding IAM user named John. But later another employee named John starts, and you create a new IAM user named John. You add the new IAM user named John to the existing IAM user group User-S3-Access. If the policy associated to the user group specifies the friendly IAM user name John, the policy allows the new John to access information that was left by the former John.

In general, we recommend that you specify the ARN for the resource in your policies instead of its unique ID. However, every IAM user has a unique ID, even if you create a new IAM user that reuses a friendly name you deleted before. In the example, the old IAM user John and the new IAM user John have different unique IDs. You can create resource-based policies 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.

The following example shows how you might specify unique IDs in the [Principal element \(p. 1264\)](#) of a resource-based policy.

```
"Principal": {  
    "AWS": [  
        "arn:aws:iam::111122223333:role/role-name",  
        "AIDACKCEVSQ6C2EXAMPLE",  
        "AROADBQP57FF2AEXAMPLE"  
    ]  
}
```

The following example shows how you might specify unique IDs in the [Condition element \(p. 1278\)](#) of a policy using global condition key [aws:userid \(p. 1366\)](#).

```
"Condition": {  
    "StringLike": {  
        "aws:userId": [  
            "AIDACKCEVSQ6C2EXAMPLE",  
            "AROADBQP57FF2AEXAMPLE:role-session-name",  
            "AROA1234567890EXAMPLE:*",  
            "111122223333"  
        ]  
    }  
}
```

Another example where user IDs can be useful is if you maintain your own database (or other store) of IAM user or role information. The unique ID can provide a unique identifier for each IAM user or role you create. This is the case when you have IAM users or roles 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. 467)
ACCA	Context-specific credential
AGPA	User 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 AWS 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 aren't distinguished by case, for example, you can't 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 \(p. 191\)](#).
- Path names must begin and end with a forward slash (/).
- Policy names for [inline policies \(p. 494\)](#) must be unique to the user, group, or role they're embedded in. The names can contain any Basic Latin (ASCII) characters except for the following reserved characters: backward slash (\), forward slash (/), asterisk (*), question mark (?), and white space. These characters are reserved according to [RFC 3986, section 2.2](#).
- 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 can't contain two consecutive hyphens, and it can't 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

Quotas, also referred to as limits in AWS, are the maximum values for the resources, actions, and items in your AWS account. Use Service Quotas to manage your IAM quotas.

For the list of IAM service endpoints and service quotas, see [AWS Identity and Access Management endpoints and quotas](#) in the [AWS General Reference](#).

To request a quota increase

1. Follow the sign-in procedure appropriate to your user type as described in the topic [How to sign in to AWS](#) in the *AWS Sign-In User Guide* to sign in to the AWS Management Console.
2. Open the Service Quotas console.
3. In the navigation pane, choose **AWS services**.
4. On the navigation bar, choose the **US East (N. Virginia)** Region. Then search for **IAM**.
5. 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.](#)

You can request an increase to default quotas for adjustable IAM quotas. Requests up to the [maximum quota](#) are automatically approved and completed within a few minutes.

The following table lists the resources for which quota increases area can be automatically approved..

Adjustable quotas for IAM resources

Resource	Default quota	Maximum quota
Customer managed policies per account	1500	5000
Groups per account	300	500
Instance profiles per account	1000	5000
Managed policies per role	10	20
Managed policies per user	10	20
Role trust policy length	2048 characters	4096 characters
Roles per account	1000	5000
Server certificates per account	20	1000

IAM Access Analyzer quotas

For the list of IAM Access Analyzer service endpoints and service quotas, see [IAM Access Analyzer endpoints and quotas](#) in the *AWS General Reference*.

IAM Roles Anywhere quotas

For the list of IAM Roles Anywhere service endpoints and service quotas, see [AWS Identity and Access Management Roles Anywhere endpoints and quotas](#) in the *AWS General Reference*.

IAM and STS character limits

The following are the maximum character counts and size limits for IAM and AWS STS. You can't request an increase for the following limits.

Description	Limit
Alias for an AWS account ID	3–63 characters
For inline policies (p. 494)	<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 can't exceed the following limits:</p> <ul style="list-style-type: none"> User policy size can't exceed 2,048 characters. Role policy size can't exceed 10,240 characters. Group policy size can't exceed 5,120 characters. <p>Note IAM doesn't count white space when calculating the size of a policy against these limits.</p>
For managed policies (p. 494)	<ul style="list-style-type: none"> The size of each managed policy can't exceed 6,144 characters. <p>Note IAM doesn't count white space when calculating the size of a policy against this limit.</p>
Group name	128 characters
Instance profile name	128 characters
Password for a login profile	1–128 characters
Path	512 characters
Policy name	128 characters
Role name	<p>64 characters</p> <p>Important If you intend to use a role with the Switch Role feature in the AWS Management Console, then the combined Path and RoleName can't exceed 64 characters.</p>
Role session duration	<p>12 hours</p> <p>When you assume a role from the AWS CLI or API, you can use the duration-seconds CLI parameter or the DurationSeconds API parameter to request a longer role session. You 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 DurationSeconds parameter, your security credentials are valid</p>

Description	Limit
	for one hour. IAM users who switch roles in the console are granted the maximum session duration, or the remaining time in the user's session, whichever is less. The maximum session duration setting doesn't 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. 276) .
Role session name	64 characters
Role session policies (p. 487)	<ul style="list-style-type: none"> The size of the passed JSON policy document and all passed managed policy ARN characters combined can't exceed 2,048 characters. 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. Additionally, an AWS conversion compresses the passed session policies and session tags into a packed binary format that has a separate limit. The PackedPolicySize response element indicates by percentage how close the policies and tags for your request are to the upper size limit. 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. 417)	<ul style="list-style-type: none"> Session tags must meet the tag key limit of 128 characters and the tag value limit 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 limit. You can pass session tags using the AWS CLI or AWS API. The PackedPolicySize response element indicates by percentage how close the policies and tags for your request are to the upper size limit.
SAML authentication response base64 encoded	<p>100,000 characters</p> <p>This character limit applies to assume-role-with-saml CLI or AssumeRoleWithSAML API operation.</p>
Tag key	<p>128 characters</p> <p>This character limit applies to tags on IAM resources and session tags (p. 417).</p>

Description	Limit
Tag value	<p>256 characters</p> <p>This character limit applies to tags on IAM resources and session tags (p. 417).</p> <p>Tag values can be empty which means tag values can have a length of 0 characters.</p>
Unique IDs created by IAM	<p>128 characters. For example:</p> <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 isn't 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>
User name	64 characters

AWS services that work with IAM

The AWS services listed below are grouped alphabetically 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 \(p. 584\)](#). 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 \(p. 1213\)](#) 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 \(p. 584\)](#). 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 **Partial** 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 \(p. 511\)](#).
- **ABAC (authorization based on tags)** – To control access based on tags, you provide tag information in the [condition element](#) of a policy using the aws:ResourceTag/*key-name*, aws:RequestTag/*key-name*, or aws:TagKeys condition keys. If a service supports all three condition keys for every resource type, then the value is Yes for the service. If a service supports all three condition keys for only some resource types, then the value is Partial. For more information about defining permissions based on attributes such as tags, see [What is ABAC for AWS? \(p. 15\)](#). To view a tutorial with steps for setting up ABAC, see [Use attribute-based access control \(ABAC\)](#).

- Temporary credentials** – You can use short-term credentials that you obtain when you sign in using IAM Identity Center, 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 \(p. 426\)](#).
- Service-linked roles** – A [service-linked role \(p. 185\)](#) is a special type of service role that gives the service permission to access resources in other services on your behalf. Choose the **Yes** or **Partial** 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 \(p. 242\)](#).
- 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.

Services that work with IAM

Service	Actions	Resource-level permission	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
AWS Account Management	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
AWS Activate Console	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Amplify Admin	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
AWS Amplify	✓ Yes	✓ Yes	✗ No	⚠ Partial	✓ Yes	✗ No
AWS Amplify UI Builder	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Apache Kafka APIs for Amazon MSK clusters	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
Amazon API Gateway	✓ Yes	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes
Amazon API Gateway Management	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
API Gateway Management V2 Amazon API Gateway Management V2	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS App2Container	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS AppConfig	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS AppFabric	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon AppFlow	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon AppIntegrations	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Application Auto Scaling	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Application Cost Profiler	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No

Service	Actions	Resource-level permission	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
AWS Application Discovery Arsenal	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Application Discovery Service	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✓ Yes
AWS Application Migration Service	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS App Mesh	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS App Mesh Preview	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✓ Yes
AWS App Runner	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon AppStream 2.0	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS AppSync	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Artifact	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
Amazon Athena	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Audit Manager	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Auto Scaling	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✓ Yes
AWS Backup	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes
AWS Backup Gateway	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Backup storage	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Batch	✓ Yes	⚠ Partial	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Bedrock	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Billing and Cost Management	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Billing Conductor	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Braket	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Budget Service	✓ Yes	✓ Yes	✗ No	✗ No	✗ No	✗ No
AWS BugBust	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Certificate Manager (ACM)	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Chatbot	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✓ Yes
Amazon Chime	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes

Service	Actions	Resource-level permission	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
AWS Clean Rooms	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Client VPN	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✓ Yes
AWS Cloud9	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes
AWS Cloud Control API	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
Amazon Cloud Directory	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
AWS CloudFormation	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon CloudFront	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	⚠ Partial (Info (p. 1242))
AWS CloudHSM	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Cloud Map	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon CloudSearch	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
AWS CloudShell	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
AWS CloudTrail	✓ Yes	✓ Yes	⚠ Partial (Info (p. 1242))	✓ Yes	✓ Yes	✓ Yes
AWS CloudTrail Data	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon CloudWatch	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	⚠ Partial (Info (p. 1242))
Amazon CloudWatch Application Insights	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
Amazon CloudWatch Evidently	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon CloudWatch Internet Monitor	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon CloudWatch Logs	✓ Yes	✓ Yes	✓ Yes	⚠ Partial	✓ Yes	✓ Yes
Amazon CloudWatch Observability Access Manager	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon CloudWatch RUM	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon CloudWatch Synthetics	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS CodeArtifact	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✗ No

Service	Actions	Resource-level permission	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
AWS CodeBuild	✓ Yes	✓ Yes	✓ Yes <small>(Info (p. 1242))</small>	⚠ Partial <small>(Info (p. 1242))</small>	✓ Yes	✗ No
Amazon CodeCatalyst	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS CodeCommit	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS CodeDeploy	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS CodeDeploy secure host commands service	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
Amazon CodeGuru Profiler	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon CodeGuru Reviewer	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon CodeGuru Security	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS CodePipeline	✓ Yes	⚠ Partial	✗ No	✓ Yes	✓ Yes	✗ No
AWS CodeStar	✓ Yes	⚠ Partial	✗ No	✓ Yes	✓ Yes	✗ No
AWS CodeStar Connections	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS CodeStar Notifications	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon CodeWhisperer	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Cognito	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Cognito Sync	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✓ Yes
Amazon Cognito user pools	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Comprehend	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Comprehend Medical	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Compute Optimizer	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✓ Yes
AWS Config	✓ Yes	⚠ Partial <small>(Info (p. 1242))</small>	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Connect	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Connect Cases	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No

Service	Actions	Resource-level permission	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
Amazon Connect Customer Profiles	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Connect High-volume outbound communications	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Connect Voice ID	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Connect Wisdom	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Console Mobile Application	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
AWS Consolidated Billing	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Control Tower	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Cost and Usage Report	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
AWS Cost Explorer	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Customer Verification Service	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Database Migration Service	✓ Yes	✓ Yes	✗ No <small>(Info (p. 1242))</small>	✓ Yes	✓ Yes	✓ Yes
Database Query Metadata Service	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Data Exchange	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Data Lifecycle Manager	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Data Pipeline	✓ Yes	✓ Yes	✗ No	⚠ Partial	✓ Yes	✗ No
AWS DataSync	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon DataZone	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
Amazon DataZone Control	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS DeepComposer	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS DeepLens	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
AWS DeepRacer	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Detective	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Device Farm	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon DevOps Guru	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✓ Yes

Service	Actions	Resource-level permission	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
AWS Direct Connect	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Directory Service	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon DocumentDB Elastic Clusters	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon DynamoDB Accelerator (DAX)	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✓ Yes
Amazon DynamoDB	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
Amazon Elastic Compute Cloud (Amazon EC2)	✓ Yes	⚠ Partial	✗ No	✓ Yes	✓ Yes	⚠ Partial (Info (p. 1242))
Amazon EC2 Auto Scaling	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
EC2 Image Builder	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon EC2 Instance Connect	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✓ Yes
Amazon ElastiCache	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Elastic Beanstalk	✓ Yes	⚠ Partial	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Elastic Block Store (Amazon EBS)	✓ Yes	⚠ Partial	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Elastic Container Registry (Amazon ECR)	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes
Amazon Elastic Container Registry Public (Amazon ECR Public)	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Elastic Container Service (Amazon ECS)	✓ Yes	⚠ Partial (Info (p. 1242))	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Elastic Disaster Recovery	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Elastic File System (Amazon EFS)	✓ Yes	✓ Yes	✓ Yes	⚠ Partial	✓ Yes	✓ Yes
Amazon Elastic Inference	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
Amazon Elastic Kubernetes Service (Amazon EKS)	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Elastic Load Balancing	✓ Yes	⚠ Partial	✗ No	⚠ Partial	✓ Yes	✓ Yes

Service	Actions	Resource-level permission	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
Amazon Elastic Transcoder	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
AWS Elemental Appliances and Software Activation Service	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Elemental Appliances and Software	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Elemental MediaConnect	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✓ Yes
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	⚠ Partial (Info (p. 1243))
AWS Elemental MediaPackage V2	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Elemental MediaPackage VOD	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	⚠ Partial (Info (p. 1243))
AWS Elemental MediaStore	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✗ No
AWS Elemental MediaTailor	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Elemental Support Cases	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Elemental Support Content	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
Amazon EMR	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon EMR on EKS	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon EMR Serverless	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Entity Resolution	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon EventBridge	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✗ No
Amazon EventBridge Pipes	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon EventBridge Scheduler	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon EventBridge Schemas	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✗ No
AWS Fault Injection Simulator	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon FinSpace	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon FinSpace API	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No

Service	Actions	Resource-level permission	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
AWS Firewall Manager	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	⚠ Partial
Fleet Hub for AWS IoT Device Management	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Forecast	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Fraud Detector	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
FreeRTOS	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Free Tier	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
Amazon FSx	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon GameLift	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon GameSparks	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Global Accelerator	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Glue	✓ Yes	✓ Yes	✓ Yes	⚠ Partial	✓ Yes	✗ No
AWS Glue DataBrew	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Ground Station	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Ground Truth Labeling	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
Amazon GuardDuty	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Health APIs And Notifications	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
AWS HealthImaging	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS HealthLake	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS HealthOmics	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Honeycode	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
AWS IAM Identity Center (successor to AWS Single Sign-On)	✓ Yes	✓ Yes	✗ No	⚠ Partial	✓ Yes	✓ Yes
IAM Identity Center Directory	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
IAM Identity Center Identity Store	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No

Service	Actions	Resource-level permission	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
AWS Identity and Access Management (IAM)	✓ Yes	✓ Yes	⚠ Partial (Info (p. 1243))	⚠ Partial (Info (p. 1243))	⚠ Partial (Info (p. 1243))	✗ No
AWS Identity and Access Management Access Analyzer	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	⚠ Partial
AWS Identity and Access Management Roles Anywhere	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Identity Store Auth	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Identity Sync	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
AWS Import/Export	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
Amazon Inspector	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Inspector Classic	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✓ Yes
Amazon Interactive Video Service	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Interactive Video Service Chat	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Invoicing	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS IoT 1-Click	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS IoT Analytics	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS IoT	✓ Yes	✓ Yes (Info (p. 1243))	⚠ Partial (Info (p. 1243))	✓ Yes	✓ Yes	✗ No
AWS IoT Core Device Advisor	✓ 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 FleetWise	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS IoT Greengrass	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS IoT Greengrass V2	✓ Yes	✓ Yes	✗ No	⚠ Partial (Info (p. 1243))	✓ Yes	✗ No
AWS IoT Jobs DataPlane	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
AWS IoT RoboRunner	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No

Service	Actions	Resource-level permission	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
AWS IoT SiteWise	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS IoT TwinMaker	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS IoT Wireless	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS IQ	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
AWS IQ Permissions	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
Amazon Kendra	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Kendra Intelligent Ranking	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Key Management Service (AWS KMS)	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes
Amazon Keyspaces (for Apache Cassandra)	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Kinesis Data Analytics	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Kinesis Data Analytics V2	✓ 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
Amazon Kinesis Video Streams	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Lake Formation	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✓ Yes
AWS Lambda	✓ Yes	✓ Yes	✓ Yes	⚠ Partial (Info (p. 1243))	✓ Yes	⚠ Partial (Info (p. 1243))
AWS Launch Wizard	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
Amazon Lex	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Lex V2	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes
AWS License Manager	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS License Manager Linux Subscriptions Manager	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS License Manager User Subscriptions	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✓ Yes

Service	Actions	Resource-level permission	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
Amazon Lightsail	✓ Yes	⚠ Partial (Info (p. 1243))	✗ No	⚠ Partial (Info (p. 1243))	✓ Yes	✓ Yes
Amazon Location Service	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Lookout for Equipment	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Lookout for Metrics	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Lookout for Vision	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Machine Learning	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
Amazon Macie	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Mainframe Modernization	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Managed Blockchain	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Managed Blockchain Query	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
Amazon Managed Grafana	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Managed Service for Prometheus	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Managed Streaming for Apache Kafka (MSK)	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Managed Streaming for Kafka Connect	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✓ Yes
Amazon Managed Workflows for Apache Airflow	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Marketplace	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✓ Yes
AWS Marketplace Catalog	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Marketplace Commerce Analytics	✓ Yes	✗ No	✗ No	✗ No	✗ No	✗ No
AWS Marketplace Discovery	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Marketplace Management Portal	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Marketplace Metering Service	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Marketplace Private Marketplace	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No

Service	Actions	Resource-level permission	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
AWS Marketplace Seller Reporting	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
AWS Marketplace Vendor Insights	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Mechanical Turk	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
Amazon MediaImport	✓ Yes	✗ No	✗ No	✗ No	✗ No	✗ No
Amazon MemoryDB for Redis	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Message Delivery Service	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Microservice Extractor for .NET	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Migration Acceleration Program Credits	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
AWS Migration Hub	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✓ Yes
AWS Migration Hub Orchestrator	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Migration Hub Refactor Spaces	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes
AWS Migration Hub Strategy Recommendations	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✓ Yes
Amazon Monitron	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon MQ	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Neptune	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✓ Yes
AWS Network Firewall	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Network Manager	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes (Info (p. 1243))
Amazon Nimble Studio	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon OpenSearch Ingestion	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon OpenSearch Serverless	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon OpenSearch Service	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes
AWS OpsWorks	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
AWS OpsWorks Configuration Management	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No

Service	Actions	Resource-level permission	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
AWS Organizations	✓ Yes	✓ Yes	✗ No	✓ Yes	✗ No	✓ Yes
AWS Outposts	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Panorama	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Payment Cryptography	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Payments	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Performance Insights	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
Amazon Personalize	✓ Yes	✓ Yes	✗ No	✗ No	✓ 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 Pinpoint SMS and Voice Service V2	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Polly	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
AWS Price List	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Private 5G	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Private Certificate Authority (AWS Private CA)	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✗ No
AWS Proton	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Purchase Orders Console	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
Amazon Quantum Ledger Database (Amazon QLDB)	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon QuickSight	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon RDS Data API	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
Amazon RDS IAM Authentication	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
AWS Recycle Bin	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Redshift	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Redshift Data API	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No

Service	Actions	Resource-level permission	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
Amazon Redshift Serverless	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✗ No
Amazon Rekognition	✓ Yes	✓ Yes	⚠ Partial (Info (p. 1243))	✓ Yes	✓ Yes	✗ No
Amazon Relational Database Service (Amazon RDS) (Info (p. 1243))	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Resilience Hub	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Resource Access Manager (AWS RAM)	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Resource Explorer	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Resource Groups	✓ Yes	✓ Yes	✗ No	✓ Yes	⚠ Partial (Info (p. 1243))	✗ No
AWS Resource Groups Tagging API	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
Amazon RHEL Knowledgebase Portal	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS RoboMaker	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Route 53	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
Amazon Route 53 Application Recovery Controller - Zonal Shift	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
Amazon Route 53 Domains	✓ Yes	✗ No	✗ No	✗ No	✗ No	✗ No
Amazon Route 53 Recovery Cluster	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
Amazon Route 53 Recovery Control Config	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Route 53 Recovery Readiness	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Route 53 Resolver	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon S3 Glacier	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✗ No
Amazon SageMaker	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	⚠ Partial (Info (p. 1244))
Amazon SageMaker geospatial capabilities	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No

Service	Actions	Resource-level permission	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
Amazon SageMaker Ground Truth Synthetic	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Savings Plans	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Secrets Manager	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✗ No
AWS Security Hub	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Security Lake	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✓ Yes
AWS Security Token Service (AWS STS)	✓ Yes	⚠ Partial (Info (p. 1244))	✗ No	✓ Yes	⚠ Partial (Info (p. 1244))	✗ No
AWS Serverless Application Repository	✓ Yes	✓ Yes	✓ Yes	✗ No	✓ Yes	✗ No
AWS Service Catalog	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Service Quotas	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Session Manager Message Gateway Service	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Shield	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Signer	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✗ No
Amazon SimpleDB	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
Amazon Simple Email Service (Amazon SES) v2	✓ Yes	⚠ Partial (Info (p. 1244))	✓ Yes	✓ Yes	⚠ Partial (Info (p. 1244))	✗ No
Amazon Simple Notification Service (Amazon SNS)	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✗ No
Amazon Simple Queue Service (Amazon SQS)	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✗ No
Amazon Simple Storage Service (Amazon S3)	✓ Yes	✓ Yes	✓ Yes	⚠ Partial (Info (p. 1244))	✓ Yes	⚠ Partial (Info (p. 1244))
Amazon Simple Storage Service (Amazon S3) Object Lambda	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
Amazon Simple Storage Service (Amazon S3) on AWS Outposts	✓ Yes	✓ Yes	✓ Yes	✗ No	✓ Yes	✗ No
Amazon Simple Workflow Service (Amazon SWF)	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No

Service	Actions	Resource-level permission	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
AWS SimSpace Weaver	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Site-to-Site VPN	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✓ Yes
AWS Snowball	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Snowball Edge	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Snow Device Management	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS SQL Workbench	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Step Functions	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Storage Gateway	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Supply Chain	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Support App in Slack	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Support	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✓ Yes
AWS Support Plans	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Sustainability	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Systems Manager	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Systems Manager for SAP	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Systems Manager GUI Connect	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Systems Manager Incident Manager	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes
AWS Systems Manager Incident Manager Contacts	✓ Yes	✓ Yes	✓ Yes	✗ No	✓ Yes	✗ No
Tag Editor	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Tax Settings	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Telco Network Builder	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Textract	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
Amazon Timestream	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Tiros API (for Reachability Analyzer)	✓ Yes	✗ No	✗ No	✗ No	✗ No	✗ No

Service	Actions	Resource-level permission	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
Amazon Transcribe	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Transfer Family	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Translate	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Trusted Advisor	⚠ Partial (Info (p. 1244))	✓ Yes	✗ No	✗ No	⚠ Partial (Info (p. 1244))	✓ Yes
AWS User Notifications	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS User Notifications Contacts	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Verified Access	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
Amazon Verified Permissions	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
Amazon Virtual Private Cloud (Amazon VPC)	✓ Yes	⚠ Partial (Info (p. 1244))	⚠ Partial (Info (p. 1244))	✓ Yes	✓ Yes	⚠ Partial (Info (p. 1244))
Amazon VPC Lattice	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon VPC Lattice Services	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
AWS WAF	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS WAF Classic	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS WAF Regional	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Well-Architected Tool	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Wickr	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon WorkDocs	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
Amazon WorkMail	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon WorkMail Message Flow	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
Amazon WorkSpaces	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon WorkSpaces Application Manager	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
Amazon WorkSpaces Web	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes

Service	Actions	Resource-level permission	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
AWS X-Ray	Yes	⚠ Partial (Info (p. 1244))	No	⚠ Partial (Info (p. 1244))	Yes	No

More information

Amazon CloudFront

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.

AWS CloudTrail

CloudTrail supports resource-based policies only on CloudTrail channels used for [CloudTrail Lake integrations with event sources outside of AWS](#).

Amazon CloudWatch

CloudWatch service-linked roles cannot be created using the AWS Management Console, and support only the [Alarm Actions](#) feature.

AWS CodeBuild

CodeBuild supports cross-account resource sharing using AWS RAM.

CodeBuild supports ABAC for project-based actions.

AWS Config

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 the [AWS Config API Guide](#).

AWS Database Migration Service

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](#).

Amazon Elastic Compute Cloud

Amazon EC2 service-linked roles can be used only for the following features: [Spot Instance Requests](#), [Spot Fleet Requests](#), [Amazon EC2 Fleets](#), and [Fast launching for Windows instances](#).

Amazon Elastic Container Service

Only some Amazon ECS actions [support resource-level permissions](#).

AWS Elemental MediaPackage

MediaPackage supports service-linked roles for publishing customer access logs to CloudWatch but not for other API actions.

AWS Identity and Access Management

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 \(p. 277\)](#).

IAM supports tag-based access control for most IAM resources. For more information, see [Tagging IAM resources \(p. 399\)](#).

Only some of the API actions for IAM can be called with temporary credentials. For more information, see [Comparing your API options](#).

AWS IoT

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](#).

AWS Lambda

Lambda supports attribute-based access control (ABAC) for API actions that use a Lambda function as the required resource. Layers, event source mappings, and code signing config resources are not supported.

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](#).

Amazon Lightsail

Lightsail partially supports resource-level permissions and ABAC. For more information, see [Actions, resources, and condition keys for Amazon Lightsail](#).

AWS Network Manager

AWS Cloud WAN also supports service-linked roles. For more information, see [AWS Cloud WAN service-linked roles](#) in the [Amazon VPC AWS Cloud WAN Guide](#).

Amazon Relational Database Service

Amazon Aurora is a fully managed relational database engine that's compatible with MySQL and PostgreSQL. You can choose the Aurora MySQL or Aurora PostgreSQL as the DB engine option when setting up new database servers through Amazon RDS. For more information, see [Identity and access management for Amazon Aurora](#) in the [Amazon Aurora User Guide](#).

Amazon Rekognition

Resource-based policies are only supported for copying Amazon Rekognition Custom Labels models.

AWS Resource Groups

Users can assume a role with a policy that allows Resource Groups operations.

Amazon SageMaker

Service-linked roles are currently available for SageMaker Studio and SageMaker training jobs.

AWS Security Token Service

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](#).

Only some of the API operations for AWS STS support calling with temporary credentials. For more information, see [Comparing your API options](#).

Amazon Simple Email Service

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 *.

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.

Amazon Simple Storage Service

Amazon S3 supports tag-based authorization for only object resources.

Amazon S3 supports service-linked roles for Amazon S3 Storage Lens.

AWS Trusted Advisor

API access to Trusted Advisor is through the AWS Support API and is controlled by AWS Support IAM policies.

Amazon Virtual Private Cloud

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 [Identity and access management for VPC endpoints and VPC endpoint services](#) in the *AWS PrivateLink Guide*.

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 [Control access to services using endpoint policies](#) in the *AWS PrivateLink Guide*.

Amazon VPC doesn't have service-linked roles, but AWS Transit Gateway does. For more information, see [Use service-linked roles for transit gateway](#) in the *Amazon VPC AWS Transit Gateway Guide*.

AWS X-Ray

X-Ray does not support resource-level permissions for all actions.

X-Ray supports tag-based access control for groups and sampling rules.

Signing AWS API requests

Important

If you use an AWS SDKs (see [Sample Code and Libraries](#)) or AWS command line (CLI) tool to send API requests to AWS, you can skip this section because the SDK and CLI clients authenticate your

requests by using the access keys that you provide. Unless you have a good reason not to, we recommend that you always use an SDK or the CLI.

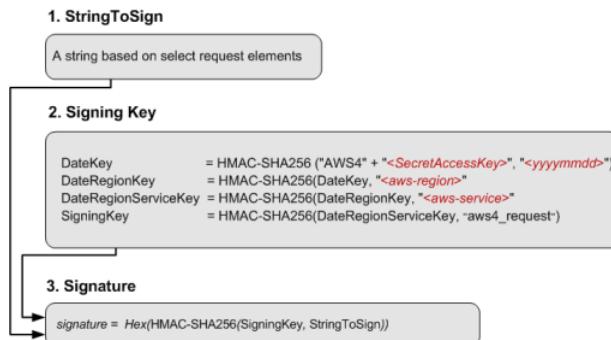
In Regions that support multiple signature versions, manually signing requests mean you must specify which signature version is used. When you supply requests to Multi-Region Access Points, SDKs and the CLI automatically switch to using Signature Version 4A without additional configuration.

Authentication information that you send in a request must include a signature. To calculate a signature, you first concatenate select request elements to form a string, referred to as the *string to sign*. You then use a signing key to calculate the hash-based message authentication code (HMAC) of the *string to sign*.

In AWS Signature Version 4, you don't use your secret access key to sign the request. Instead, you first use your secret access key to derive a signing key. The derived signing key is specific to the date, service, and Region. For more information about how to derive a signing key in different programming languages, see [Request signature examples \(p. 1258\)](#).

Signature Version 4 is the AWS signing protocol. AWS also supports an extension, Signature Version 4A, which supports signatures for multi-Region API requests. For more information, see the [sigv4a-signing-examples](#) project on GitHub.

The following diagram illustrates the general process of computing a signature.



- The **string to sign** depends on the request type. For example, when you use the HTTP Authorization header or the query parameters for authentication, you use a varying combination of request elements to create the string to sign. For an HTTP POST request, the POST policy in the request is the string you sign.
- For **signing key**, the diagram shows series of calculations, where result of each step you feed into the next step. The final step is the signing key.
- When an AWS service receives an authenticated request, it recreates the **signature** using the authentication information contained in the request. If the signatures match, the service processes the request. Otherwise, it rejects the request.

Contents

- [When to sign requests \(p. 1246\)](#)
- [Why requests are signed \(p. 1246\)](#)
- [Elements of an AWS API request signature \(p. 1246\)](#)
- [Authentication methods \(p. 1247\)](#)
- [Create a signed AWS API request \(p. 1250\)](#)
- [Request signature examples \(p. 1258\)](#)
- [Troubleshoot signed requests for AWS APIs \(p. 1258\)](#)

When to sign requests

When you write custom code that sends API requests to AWS, you must include code that signs the requests. You might write custom code because:

- You are working with a programming language for which there is no AWS SDK.
- You need complete control over how requests are sent to AWS.

Why requests are signed

The signing process helps secure requests in the following ways:

- **Verify the identity of the requester**

Authenticated requests require a signature that you create by using your access keys (access key ID, secret access key). If you are using temporary security credentials, the signature calculations also require a security token. For more information, see [AWS security credentials programmatic access](#).

- **Protect data in transit**

To prevent tampering with a request while it's in transit, some of the request elements are used to calculate a hash (digest) of the request, and the resulting hash value is included as part of the request. When an AWS service receives the request, it uses the same information to calculate a hash and matches it against the hash value in your request. If the values don't match, AWS denies the request.

- **Protect against potential replay attacks**

In most cases, a request must reach AWS within five minutes of the time stamp in the request. Otherwise, AWS denies the request.

Elements of an AWS API request signature

Important

Unless you are using the AWS SDKs or CLI, you must write code to calculate signatures that provide authentication information in your requests. Signature calculation in AWS Signature Version 4 can be a complex undertaking, and we recommend that you use the AWS SDKs or CLI whenever possible.

Each HTTP/HTTPS request that uses Signature Version 4 signing must contain these elements.

Elements

- [Endpoint specification \(p. 1246\)](#)
- [Action \(p. 1247\)](#)
- [Action parameters \(p. 1247\)](#)
- [Date \(p. 1247\)](#)
- [Authentication information \(p. 1247\)](#)

Endpoint specification

Specifies the DNS name of the endpoint to which you send the request. This name usually contains the service code and the Region. For example, the endpoint for Amazon DynamoDB in the us-east-1 Region is dynamodb.us-east-1.amazonaws.com.

For HTTP/1.1 requests, you must include the Host header. For HTTP/2 requests, you can include the :authority header or the Host header. Use only the :authority header for compliance with the HTTP/2 specification. Not all services support HTTP/2 requests.

For the endpoints supported by each service, see [Service endpoints and quotas](#) in the *AWS General Reference*.

Action

Specifies an API action for the service. For example, the DynamoDB CreateTable action or the Amazon EC2 DescribeInstances action.

For the actions supported by each service, see the [Service Authorization Reference](#).

Action parameters

Specifies the parameters for the action specified in the request. Each AWS API action has a set of required and optional parameters. The API version is usually a required parameter.

For the parameters supported by an API action, see the [API Reference](#) for the service.

Date

Specifies the date and time of the request. Including the date and time in a request helps prevent third parties from intercepting your request and resubmitting it later. The date that you specify in the credential scope must match the date of your request.

The time stamp must be in UTC and use the following ISO 8601 format: YYYYMMDDTHHMMSSZ. For example, 20220830T123600Z. Do not include milliseconds in the time stamp.

You can use a date or an x-amz-date header, or include x-amz-date as a query parameter. If we can't find an x-amz-date header, then we look for a date header.

Authentication information

Each request that you send must include the following information. AWS uses this information to ensure the validity and authenticity of the request.

- Algorithm – Use AWS4-HMAC-SHA256 to specify Signature Version 4 with the HMAC-SHA256 hash algorithm.
- Credential – A string that consists of your access key ID, the date in YYYYMMDD format, the Region code, the service code, and the aws4_request termination string, separated by slashes (/). The Region code, service code, and termination string must use lowercase characters.

AKIAIOSFODNN7EXAMPLE/YYYYMMDD/*region/service*/aws4_request

- Signed headers – The HTTP headers to include in the signature, separated by semicolons (;). For example, host;x-amz-date.
- Signature – A hexadecimal-encoded string that represents the calculated signature. You must calculate the signature using the algorithm that you specified in the Algorithm parameter.

Authentication methods

Important

Unless you are using the AWS SDKs or CLI, you must write code to calculate signatures that provide authentication information in your requests. Signature calculation in AWS Signature

Version 4 can be a complex undertaking, and we recommend that you use the AWS SDKs or CLI whenever possible.

You can express authentication information by using one of the following methods.

HTTP authorization header

The HTTP Authorization header is the most common method of authenticating a request. All REST API operations (except for browser-based uploads using POST requests) require this header. For more information about the authorization header value, and how to calculate signature and related options, see [Authenticating Requests: Using the Authorization Header \(AWS Signature Version 4\)](#) in the *Amazon S3 API Reference*.

The following is an example of the Authorization header value. Line breaks are added to this example for readability. In your code, the header must be a continuous string. There is no comma between the algorithm and Credential, but the other elements must be separated by commas.

```
Authorization: AWS4-HMAC-SHA256
Credential=AKIAIOSFODNN7EXAMPLE/20130524/us-east-1/s3/aws4_request,
SignedHeaders=host;range;x-amz-date,
Signature=fe5f80f77d5fa3beca038a248ff027d0445342fe2855ddc963176630326f1024
```

The following table describes the various components of the Authorization header value in the preceding example:

Component	Description
Authorization	The algorithm that was used to calculate the signature. You must provide this value when you use AWS Signature Version 4 for authentication. The string specifies AWS Signature Version 4 (AWS4) and the signing algorithm (HMAC-SHA256).
Credential	Your access key ID and the scope information, which includes the date, Region, and service that were used to calculate the signature. This string has the following form: <your-access-key-id>/<date>/<aws-region>/<aws-service>/aws4_request Where: <date> value is specified using YYYYMMDD format. <aws-service> value is s3 when sending request to Amazon S3.
SignedHeaders	A semicolon-separated list of request headers that you used to compute Signature. The list includes header names only, and the header names must be in lowercase. For example: host;range;x-amz-date
Signature	The 256-bit signature expressed as 64 lowercase hexadecimal characters. For example:fe5f80f77d5fa3beca038a248ff027d0445342fe28

Component	Description
	Note that the signature calculations vary depending on the option you choose to transfer the payload.

Query string parameters

You can use a query string to express a request entirely in a URL. In this case, you use query parameters to provide request information, including the authentication information. Because the request signature is part of the URL, this type of URL is often referred to as a presigned URL. You can use presigned URLs to embed clickable links in HTML, which can be valid for up to seven days. For more information, see [Authenticating Requests: Using Query Parameters \(AWS Signature Version 4\)](#) in the *Amazon S3 API Reference*.

The following is an example presigned URL. Line breaks are added to this example for readability:

```
https://s3.amazonaws.com/examplebucket/test.txt ?
X-Amz-Algorithm=AWS4-HMAC-SHA256 &
X-Amz-Credential=<your-access-key-id>/20130721/us-east-1/s3/aws4_request &
X-Amz-Date=20130721T201207Z &
X-Amz-Expires=86400 &
X-Amz-SignedHeaders=host &X-Amz-Signature=<signature-value>
```

Note

The X-Amz-Credential value in the URL shows the "/" character only for readability. In practice, it should be encoded as %2F. For example:

```
&X-Amz-Credential=<your-access-key-id>%2F20130721%2Fus-
east-1%2Fs3%2Faws4_request
```

The following table describes the query parameters in the URL that provide authentication information.

Query string parameter name	Description
X-Amz-Algorithm	Identifies the version of AWS Signature and the algorithm that you used to calculate the signature. For AWS Signature Version 4, you set this parameter value to AWS4-HMAC-SHA256. This string identifies AWS Signature Version 4 (AWS4) and the HMAC-SHA256 algorithm (HMAC-SHA256).
X-Amz-Credential	In addition to your access key ID, this parameter also provides scope (AWS Region and service) for which the signature is valid. This value must match the scope you use in signature calculations, discussed in the following section. The general form for this parameter value is as follows: <your-access-key-id>/<date>/<AWS Region>/<AWS-service>/aws4_request For example: AKIAIOSFODNN7EXAMPLE/20130721/us-east-1/s3/aws4_request

Query string parameter name	Description
	For a list of AWS regional strings, see Regional Endpoints in the AWS General Reference .
X-Amz-Date	The date and time format must follow the ISO 8601 standard, and must be formatted with the yyyyMMddTHHmmssZ format. For example if the date and time was "08/01/2016 15:32:41.982-700" then it must first be converted to UTC (Coordinated Universal Time) and then submitted as "20160801T223241Z".
X-Amz-Expires	Provides the time period, in seconds, for which the generated presigned URL is valid. For example, 86400 (24 hours). This value is an integer. The minimum value you can set is 1, and the maximum is 604800 (seven days). A presigned URL can be valid for a maximum of seven days because the signing key you use in signature calculation is valid for up to seven days.
X-Amz-SignedHeaders	<p>Lists the headers that you used to calculate the signature. The following headers are required in the signature calculations:</p> <ul style="list-style-type: none"> • The HTTP host header. • Any x-amz-* headers that you plan to add to the request. <p>For added security, you should sign all the request headers that you plan to include in your request.</p>
X-Amz-Signature	<p>Provides the signature to authenticate your request. This signature must match the signature the service calculates; otherwise, the service denies the request. For example, 733255ef022bec3f2a8701cd61d4b371f3f28c9f193a1f022</p> <p>Signature calculations are described in the following section.</p>
X-Amz-Security-Token	Optional credential parameter if using credentials sourced from the STS service.

Create a signed AWS API request

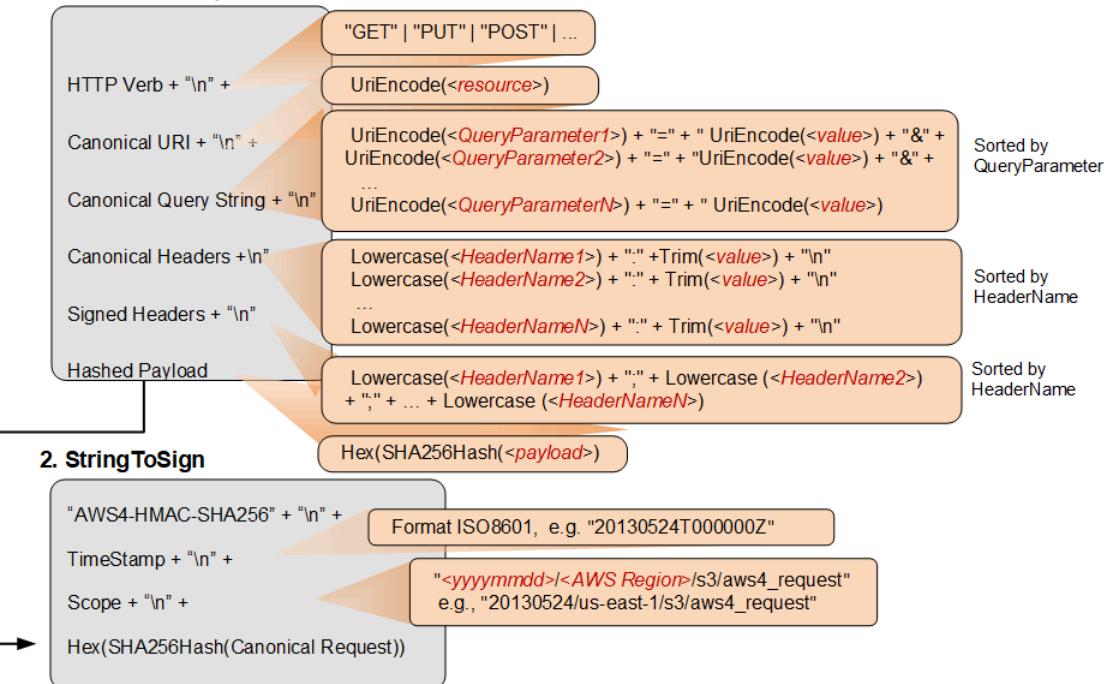
Important

If you use an AWS SDKs (see [Sample Code and Libraries](#)) or AWS command line (CLI) tool to send API requests to AWS, you can skip this section because the SDK and CLI clients authenticate your requests by using the access keys that you provide. Unless you have a good reason not to, we recommend that you always use an SDK or the CLI.

In Regions that support multiple signature versions, manually signing requests mean you must specify which signature version is used. When you supply requests to Multi-Region Access Points, SDKs and the CLI automatically switch to using Signature Version 4A without additional configuration.

The following is an overview of the process to create a signed request. To calculate a signature, you first need a string to sign. You then calculate a HMAC-SHA256 hash of the string to sign by using a signing key. The following diagram illustrates the process, including the various components of the string that you create for signing.

1. Canonical Request



2. StringToSign

3. Signing Key

DateKey	= HMAC-SHA256 ("AWS4" + "<SecretAccessKey>", "<yyyymmdd>")
DateRegionKey	= HMAC-SHA256(DateKey, "<aws-region>"")
DateRegionServiceKey	= HMAC-SHA256(DateRegionKey, "<aws-service>"")
SigningKey	= HMAC-SHA256(DateRegionServiceKey, "aws4_request")

4. Signature

```
signature = Hex(HMAC-SHA256 ( SigningKey, StringToSign))
```

The following table describes the functions that are shown in the diagram. You need to implement code for these functions. For more information, see the [code examples in the AWS SDKs \(p. 1257\)](#).

Function	Description
Lowercase()	Convert the string to lowercase.
Hex()	Lowercase base 16 encoding.
SHA256Hash()	Secure Hash Algorithm (SHA) cryptographic hash function.

Function	Description
HMAC-SHA256()	Computes HMAC by using the SHA256 algorithm with the signing key provided. This is the final signature.
Trim()	Remove any leading or trailing whitespace.
UriEncode()	<p>URI encode every byte. UriEncode() must enforce the following rules:</p> <ul style="list-style-type: none"> URI encode every byte except the unreserved characters: 'A'-'Z', 'a'-'z', '0'-'9', '!', '.', '_', and '~'. The space character is a reserved character and must be encoded as "%20" (and not as "+"). Each URI encoded byte is formed by a '%' and the two-digit hexadecimal value of the byte. Letters in the hexadecimal value must be uppercase, for example "%1A". Encode the forward slash character, '/', everywhere except in the object key name. For example, if the object key name is photos/Jan/sample.jpg, the forward slash in the key name is not encoded. <p>Important The standard UriEncode functions provided by your development platform may not work because of differences in implementation and related ambiguity in the underlying RFCs. We recommend that you write your own custom UriEncode function to ensure that your encoding will work.</p> <p>To see an example of a UriEncode function in Java, see Java Utilities on the GitHub website.</p>

Note

When signing your requests, you can use either AWS Signature Version 4 or AWS Signature Version 4A. The key difference between the two is determined by how the signature is calculated. With AWS Signature Version 4A, the signature does not include Region-specific information and is calculated using the AWS4-ECDSA-P256-SHA256 algorithm.

Temporary security credentials

Instead of using long-term credentials to sign a request, you can use temporary security credentials provided by AWS Security Token Service (AWS STS).

When you use temporary security credentials, you must add X-Amz-Security-Token to the Authorization header or the query string to hold the session token. Some services require that you add X-Amz-Security-Token to the canonical request. Other services require only that you add X-Amz-Security-Token at the end, after you calculate the signature. Check the documentation for each AWS service for details.

Summary of signing steps

Step 1: Create a canonical request

Arrange the contents of your request (host, action, headers, etc.) into a standard canonical format. The canonical request is one of the inputs used to create a string to sign. For details, see [Elements of an AWS API request signature \(p. 1246\)](#).

Step 2: Create a String to Sign

Create a string to sign with the canonical request and extra information such as the algorithm, request date, credential scope, and the digest (hash) of the canonical request.

Step 3: Create a hash of the canonical request

Derive a signing key by performing a succession of keyed hash operations (HMAC operations) on the request date, Region, and service, with your AWS secret access key as the key for the initial hashing operation.

Step 4: Calculate the signature

After you derive the signing key, you then calculate the signature by performing a keyed hash operation on the string to sign. Use the derived signing key as the hash key for this operation.

Step 5: Add the signature to the request

After you calculate the signature, add it to an HTTP header or to the query string of the request.

Step 1: Create a canonical request

Create a canonical request by concatenating the following strings, separated by newline characters. This helps ensure that the signature that you calculate and the signature that AWS calculates can match.

```
<HTTPMethod>\n<CanonicalURI>\n<CanonicalQueryString>\n<CanonicalHeaders>\n<SignedHeaders>\n<HashedPayload>
```

- **HTTPMethod** – The HTTP method, such as GET, PUT, HEAD, and DELETE.
- **CanonicalUri** – The URI-encoded version of the absolute path component URI, starting with the "/" that follows the domain name and up to the end of the string or to the question mark character ('?') if you have query string parameters. If the absolute path is empty, use a forward slash character (/). The URI in the following example, /examplebucket/myphoto.jpg, is the absolute path and you don't encode the "/" in the absolute path:

```
http://s3.amazonaws.com/examplebucket/myphoto.jpg
```

- **CanonicalQueryString** – The URI-encoded query string parameters. You URL-encode each name and values individually. You must also sort the parameters in the canonical query string alphabetically by key name. The sorting occurs after encoding. The query string in the following URI example is:

```
http://s3.amazonaws.com/examplebucket?prefix=somePrefix&marker=someMarker&max-keys=2
```

The canonical query string is as follows (line breaks are added to this example for readability):

```
UriEncode("marker")+"="+UriEncode("someMarker")+"&"+  
UriEncode("max-keys")+"="+UriEncode("20") + "&" +  
UriEncode("prefix")+"="+UriEncode("somePrefix")
```

When a request targets a subresource, the corresponding query parameter value will be an empty string (""). For example, the following URI identifies the ACL subresource on the examplebucket bucket:

```
http://s3.amazonaws.com/examplebucket?acl
```

The CanonicalQueryString in this case is as follows:

```
UriEncode("acl") + "=" + ""
```

If the URI does not include a '?', there is no query string in the request, and you set the canonical query string to an empty string (""). You will still need to include the "\n".

- **CanonicalHeaders** – A list of request headers with their values. Individual header name and value pairs are separated by the newline character ("\n"). The following is an example of a canonicalheader:

```
Lowercase(<HeaderName1>)+":"+Trim(<value>)+"\n"  
Lowercase(<HeaderName2>)+":"+Trim(<value>)+"\n"  
...  
Lowercase(<HeaderNameN>)+":"+Trim(<value>)+"\n"
```

CanonicalHeaders list must include the following:

- HTTP host header.
- If the Content-Type header is present in the request, you must add it to the **CanonicalHeaders** list.
- Any x-amz-* headers that you plan to include in your request must also be added. For example, if you are using temporary security credentials, you need to include x-amz-security-token in your request. You must add this header in the list of **CanonicalHeaders**.

Note

The x-amz-content-sha256 header is required for all AWS Signature Version 4 requests. It provides a hash of the request payload. If there is no payload, you must provide the hash of an empty string.

Each header name must:

- use lowercase characters.
- appear in alphabetical order.
- be followed by a colon (:).

For values, you must:

- trim any leading or trailing spaces.
- convert sequential spaces to a single space.
- separate the values for a multi-value header using commas.
- You must include the host header (HTTP/1.1) or the :authority header (HTTP/2), and any x-amz-* headers in the signature. You can optionally include other standard headers in the signature, such as content-type.

The Lowercase() and Trim() functions used in this example are described in the preceding section.

The following is an example CanonicalHeaders string. The header names are in lowercase and sorted.

```
host:s3.amazonaws.com
x-amz-content-sha256:e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
x-amz-date:20130708T220855Z
```

Note

For the purpose of calculating an authorization signature, only the host and any x-amz-* headers are required; however, in order to prevent data tampering, you should consider including all the headers in the signature calculation.

- **SignedHeaders** – An alphabetically sorted, semicolon-separated list of lowercase request header names. The request headers in the list are the same headers that you included in the CanonicalHeaders string. For example, for the previous example, the value of **SignedHeaders** would be as follows:

```
host;x-amz-content-sha256;x-amz-date
```

- **HashedPayload** – A string created using the payload in the body of the HTTP request as input to a hash function. This string uses lowercase hexadecimal characters.

```
Hex(SHA256Hash(<payload>))
```

If there is no payload in the request, you compute a hash of the empty string as follows:

```
Hex(SHA256Hash("")))
```

The hash returns the following value:

```
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
```

For example, when you upload an object by using a PUT request, you provide object data in the body. When you retrieve an object by using a GET request, you compute the empty string hash.

Step 2: Create a hash of the canonical request

Create a hash (digest) of the canonical request using the same algorithm that you used to create the hash of the payload. The hash of the canonical request is a string of lowercase hexadecimal characters.

Step 3: Create a string to sign

Create a string by concatenating the following strings, separated by newline characters. Do not end this string with a newline character.

```
Algorithm \n
RequestDateTime \n
CredentialScope \n
HashedCanonicalRequest
```

- *Algorithm* – The algorithm used to create the hash of the canonical request. For SHA-256, the algorithm is AWS4-HMAC-SHA256.
- *RequestDateTime* – The date and time used in the credential scope. This value is the current UTC time in ISO 8601 format (for example, 20130524T000000Z).
- *CredentialScope* – The credential scope. This restricts the resulting signature to the specified Region and service. The string has the following format: YYYYMMDD/*region/service*/aws4_request.
- *HashedCanonicalRequest* – The hash of the canonical request. This value is calculated in Step 2.

The following is an example string to sign.

```
"AWS4-HMAC-SHA256" + "\n" +
timeStampISO8601Format + "\n" +
<Scope> + "\n" +
Hex(SHA256Hash(<CanonicalRequest>))
```

Step 4: Calculate the signature

In AWS Signature Version 4, instead of using your AWS access keys to sign a request, you create a signing key that is scoped to a specific Region and service as the authentication information you'll add to your request.

```
DateKey = HMAC-SHA256("AWS4"+<SecretAccessKey>, "<YYYYMMDD>")
DateRegionKey = HMAC-SHA256(<DateKey>, "<aws-region>")
DateRegionServiceKey = HMAC-SHA256(<DateRegionKey>, "<aws-service>")
SigningKey = HMAC-SHA256(<DateRegionServiceKey>, "aws4_request")
```

For a list of Region strings, see [Regional Endpoints](#) in the *AWS General Reference*.

For each step, call the hash function with the required key and data. The result of each call to the hash function becomes the input for the next call to the hash function.

Required input

- A string, Key, that contains your secret access key
- A string, Date, that contains the date used in the credential scope, in the format YYYYMMDD
- A string, Region, that contains the Region code (for example, us-east-1)
- A string, Service, that contains the service code (for example, ec2)
- The string to sign that you created in the previous step.

To calculate the signature

1. Concatenate "AWS4" and the secret access key. Call the hash function with the concatenated string as the key and the date string as the data.

```
kDate = hash("AWS4" + Key, Date)
```

2. Call the hash function with the result of the previous call as the key and the Region string as the data.

```
kRegion = hash(kDate, Region)
```

3. Call the hash function with the result of the previous call as the key and the service string as the data.

```
kService = hash(kRegion, Service)
```

4. Call the hash function with the result of the previous call as the key and "aws4_request" as the data.

```
kSigning = hash(kService, "aws4_request")
```

5. Call the hash function with the result of the previous call as the key and the string to sign as the data. The result is the signature as a binary value.

```
signature = hash(kSigning, string-to-sign)
```

6. Convert the signature from binary to hexadecimal representation, in lowercase characters.

Step 5: Add the signature to the request

Example Example: Authorization header

The following example shows an Authorization header for the `DescribeInstances` action. For readability, this example is formatted with line breaks. In your code, this must be a continuous string. There is no comma between the algorithm and `Credential`. However, the other elements must be separated by commas.

```
Authorization: AWS4-HMAC-SHA256
Credential=AKIAIOSFODNN7EXAMPLE/20220830/us-east-1/ec2/aws4_request,
SignedHeaders=host;x-amz-date,
Signature=calculated-signature
```

Example Example: Request with authentication parameters in the query string

The following example shows a query for the `DescribeInstances` action that includes the authentication information. For readability, this example is formatted with line breaks and is not URL encoded. In your code, the query string must be a continuous string that is URL encoded.

```
https://ec2.amazonaws.com/?
Action=DescribeInstances&
Version=2016-11-15&
X-Amz-Algorithm=AWS4-HMAC-SHA256&
X-Amz-Credential=AKIAIOSFODNN7EXAMPLE/20220830/us-east-1/ec2/aws4_request&
X-Amz-Date=20220830T123600Z&
X-Amz-SignedHeaders=host;x-amz-date&
X-Amz-Signature=calculated-signature
```

Code examples in the AWS SDKs

The AWS SDKs include source code on GitHub that shows how to sign AWS API requests.

- AWS SDK for .NET – [AWS4Signer.cs](#)
- AWS SDK for C++ – [AWSAuthV4Signer.cpp](#)
- AWS SDK for Go – [v4.go](#)
- AWS SDK for Java – [BaseAws4Signer.java](#)
- AWS SDK for JavaScript – [v4.js](#)
- AWS SDK for PHP – [SignatureV4.php](#)
- AWS SDK for Python (Boto) – [signers.py](#)

- AWS SDK for Ruby – [signer.rb](#)

Request signature examples

The following examples of AWS signing requests show you how you can use SigV4 to sign requests sent without the AWS SDK or AWS command line tool.

Need help? [Troubleshoot SigV4 signature mismatch errors](#)

Browser based Amazon S3 upload using HTTP POST

[Authenticating Requests: Browser-Based Uploads](#) describes the signature and relevant information that Amazon S3 uses to calculate the signature upon receiving the request.

[Example: Browser-Based Upload using HTTP POST \(Using AWS Signature Version 4\)](#) provides more information with a sample POST policy and a form that you can use to upload a file. The example policy and fictitious credentials show you the workflow and resulting signature and policy hash.

VPC Lattice authenticated requests

[Examples for Signature Version 4 \(SigV4\) authenticated requests](#) provides Python and Java examples showing how you can perform request signing with and without custom interceptors.

Using Signature Version 4 with Amazon Translate

[Using Signature Version 4 with Amazon Translate](#) shows how to use a Python program to add authentication information to Amazon Translate requests. The example makes a POST request, creates a JSON structure that contains the text to be translated in the body (payload) of the request, and passes authentication information in an Authorization header.

Using Signature Version 4 with Neptune

[Example: Connecting to Neptune Using Python with Signature Version 4 Signing](#) shows how to make signed requests to Neptune using Python. This example includes variations for using an access key or temporary credentials.

Troubleshoot signed requests for AWS APIs

When you develop code that creates a signed request, you might receive HTTP 403 SignatureDoesNotMatch errors from the AWS services that you test against. This error means that the signature value in your HTTP request to AWS did not match the signature that the AWS service calculated.

Possible causes

- [Canonicalization errors \(p. 1258\)](#)
- [Credential scope errors \(p. 1259\)](#)
- [Key signing errors \(p. 1260\)](#)

Canonicalization errors

If you incorrectly calculated the canonical request or the string to sign, the signature verification step performed by the service fails with the following error message:

The request signature we calculated does not match the signature you provided

The error response includes the canonical request and the string to sign that the service calculated. You can compare these strings with the strings that you calculated.

You can also verify that you didn't send the request through a proxy that modifies the headers or the request.

Credential scope errors

The credential scope restricts a signature to a specific date, Region, and service. This string has the following format:

`YYYYMMDD/region/service/aws4_request`

Date

If the credential scope does not specify the same date as the `x-amz-date` header, the signature verification step fails with the following error message:

Date in Credential scope does not match YYYYMMDD from ISO-8601 version of date from HTTP

If the request specifies a time in the future, the signature verification step fails with the following error message:

Signature not yet current: `date` is still later than `date`

If the request has expired, the signature verification step fails with the following error message:

Signature expired: `date` is now earlier than `date`

Region

If the credential scope does not specify the same Region as the request, the signature verification step fails with the following error message:

Credential should be scoped to a valid Region, not `region-code`

Service

If the credential scope does not specify the same service as the host header, the signature verification step fails with the following error message:

Credential should be scoped to correct service: '`service`'

Termination string

If the credential scope does not end with `aws4_request`, the signature verification step fails with the following error message:

Credential should be scoped with a valid terminator: '`aws4_request`'

Key signing errors

Errors that are caused by incorrect derivation of the signing key or improper use of cryptography are more difficult to troubleshoot. After you verify that the canonical string and the string to sign are correct, you can also check for one of the following issues:

- The secret access key does not match the access key ID that you specified.
- There is a problem with your key derivation code.

To verify that the secret key matches the access key ID, you can test them with a known working implementation. For example, use an AWS SDK or the AWS CLI to make a request to AWS.

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 \(p. 490\)](#).

This reference includes the following sections.

- [IAM JSON policy elements reference \(p. 1260\)](#) — 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 \(p. 1306\)](#) — 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 \(p. 1323\)](#) — This section presents a formal grammar for the language that is used to create policies in IAM.
- [AWS managed policies for job functions \(p. 1329\)](#) — 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 \(p. 1338\)](#) — 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 \(p. 1367\)](#) — 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 \(p. 490\)](#).

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 \(p. 1224\)](#).

When you create or edit a JSON policy, IAM can perform policy validation to help you create an effective policy. IAM identifies JSON syntax errors, while IAM Access Analyzer provides additional policy checks with recommendations to help you further refine your policies. To learn more about policy validation, see [Validating IAM policies \(p. 588\)](#). To learn more about IAM Access Analyzer policy checks and actionable recommendations, see [IAM Access Analyzer policy validation](#).

Topics

- [IAM JSON policy elements: Version \(p. 1261\)](#)
- [IAM JSON policy elements: Id \(p. 1262\)](#)
- [IAM JSON policy elements: Statement \(p. 1262\)](#)
- [IAM JSON policy elements: Sid \(p. 1263\)](#)
- [IAM JSON policy elements: Effect \(p. 1263\)](#)
- [AWS JSON policy elements: Principal \(p. 1264\)](#)
- [AWS JSON policy elements: NotPrincipal \(p. 1272\)](#)
- [IAM JSON policy elements: Action \(p. 1273\)](#)
- [IAM JSON policy elements: NotAction \(p. 1274\)](#)
- [IAM JSON policy elements: Resource \(p. 1276\)](#)
- [IAM JSON policy elements: NotResource \(p. 1277\)](#)
- [IAM JSON policy elements: Condition \(p. 1278\)](#)
- [IAM policy elements: Variables and tags \(p. 1298\)](#)
- [IAM JSON policy elements: Supported data types \(p. 1305\)](#)

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" \(p. 606\)](#).

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 **outside** the `Statement` element in all of your policies.

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": "s3>ListAllMyBuckets",  
      "Resource": "*"  
    }  
  ]  
}
```

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 \(p. 1298\)](#) 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. 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. ID is allowed in resource-based policies, but not in identity-based policies.

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.

```
{  
    "Version": "2012-10-17",  
    "Id": "cd3ad3d9-2776-4ef1-a904-4c229d1642ee",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "s3>ListAllMyBuckets",  
            "Resource": "*"  
        }  
    ]  
}
```

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 \(p. 1298\)](#).

```
{  
    "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

You can provide **Sid** (statement ID) as an optional identifier for the policy statement. You can assign a **Sid** value to each statement in a statement array. You can use the **Sid** value as a description for the policy statement. 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 ID. In IAM, the **Sid** value must be unique within a JSON policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ExampleStatementID",
            "Effect": "Allow",
            "Action": "s3>ListAllMyBuckets",
            "Resource": "*"
        }
    ]
}
```

The **Sid** element supports ASCII uppercase letters (A-Z), lowercase letters (a-z), and numbers (0-9).

IAM does not expose the **Sid** 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 work 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**. The **Effect** value is case sensitive.

```
"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 \(p. 1306\)](#).

AWS JSON policy elements: Principal

Use the Principal element in a resource-based JSON policy to specify the principal that is allowed or denied access to a resource.

You must use the Principal element in [resource-based policies \(p. 511\)](#). Several services support resource-based policies, including IAM. The IAM resource-based policy type is a role trust policy. In IAM roles, use the Principal element in the role 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!

Other examples of resources that support resource-based policies include an Amazon S3 bucket or an AWS KMS key.

You cannot use the Principal element in an identity-based policy. Identity-based policies are permissions policies that you attach to IAM identities (users, groups, or roles). In those cases, the principal is implicitly the identity where the policy is attached.

Topics

- [Specifying a principal \(p. 1264\)](#)
- [AWS account principals \(p. 1265\)](#)
- [IAM role principals \(p. 1266\)](#)
- [Role session principals \(p. 1267\)](#)
- [IAM user principals \(p. 1268\)](#)
- [IAM Identity Center principals \(p. 1268\)](#)
- [AWS STS federated user session principals \(p. 1269\)](#)
- [AWS service principals \(p. 1269\)](#)
- [AWS service principals in opt-in Regions \(p. 1270\)](#)
- [All principals \(p. 1270\)](#)
- [More information \(p. 1271\)](#)

Specifying a principal

You specify a principal in the Principal element of a resource-based policy or in condition keys that support principals.

You can specify any of the following principals in a policy:

- AWS account and root user

- IAM roles
- Role sessions
- IAM users
- Federated user sessions
- AWS services
- All principals

You cannot identify a user group as a principal in a policy (such as a resource-based policy) because groups relate to permissions, not authentication, and principals are authenticated IAM entities.

You can specify more than one principal for each of the principal types in following sections using an array. Arrays can take one or more values. When you specify more than one principal in an element, you grant permissions to each principal. This is a logical OR and not a logical AND, because you authenticate as one principal at a time. If you include more than one value, use square brackets ([and]) and comma-delimit each entry for the array. The following example policy defines permissions for the 123456789012 account or the 555555555555 account.

```
"Principal" : {  
  "AWS": [  
    "123456789012",  
    "555555555555"  
  ]  
}
```

Note

You cannot use a wildcard to match part of a principal name or ARN.

AWS account principals

You can specify AWS account identifiers in the Principal element of a resource-based policy or in condition keys that support principals. This delegates authority to the account. When you allow access to a different account, an administrator in that account must then grant access to an identity (IAM user or role) in that account. When you specify an AWS account, you can use the account ARN (arn:aws:iam:*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" }
```

The account ARN and the shortened account ID behave the same way. Both delegate permissions to the account. Using the account ARN in the Principal element does not limit permissions to only the root user of the account.

Note

When you save a resource-based policy that includes the shortened account ID, the service might convert it to the principal ARN. This does not change the functionality of the policy.

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" }
```

You can also specify more than one AWS account, (or canonical user ID) as a principal using an array. For example, you can specify a principal in a bucket policy using all three methods.

```
"Principal": {  
  "AWS": [  
    "arn:aws:iam::123456789012:root",  
    "999999999999"  
  ],  
  "CanonicalUser": "79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be"  
}
```

IAM role principals

You can specify IAM role principal ARNs in the `Principal` element of a resource-based policy or in condition keys that support principals. IAM roles are identities. In IAM, identities are resources to which you can assign permissions. Roles trust another authenticated identity to assume that role. This includes a principal in AWS or a user from an external identity provider (IdP). When a principal or identity assumes a role, they receive temporary security credentials with the assumed role's permissions. When they use those session credentials to perform operations in AWS, they become a *role session principal*.

IAM roles are identities that exist in IAM. Roles trust another authenticated identity, such as a principal in AWS or a user from an external identity provider. When a principal or identity assumes a role, they receive temporary security credentials. They can then use those credentials as a role session principal to perform operations in AWS.

When you specify a role principal in a resource-based policy, the effective permissions for the principal are limited by any policy types that limit permissions for the role. This includes session policies and permissions boundaries. For more information about how the effective permissions for a role session are evaluated, see [Policy evaluation logic \(p. 1306\)](#).

To specify the role ARN in the `Principal` element, use the following format:

```
"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 transforms to the role unique principal ID when you save the policy. 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 IAM uses a reverse transformation back to the role ARN when the trust policy is displayed. However, if you delete the role, then you break the relationship. 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 appears in resource-based policies 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 in the policy to replace the principal ID with the correct ARN. The ARN once again transforms into the role's new principal ID when you save the policy.

Alternatively, you can specify the role principal as the principal in a resource-based policy or [create a broad-permission policy \(p. 1270\)](#) that uses the `aws:PrincipalArn` condition key. When you use this key, the role session principal is granted the permissions based on the ARN of role that was assumed, and not the ARN of the resulting session. Because AWS does not convert condition key ARNs to IDs, permissions granted to the role ARN persist if you delete the role and then create a new role with the same name. Identity-based policy types, such as permissions boundaries or session policies, do not limit

permissions granted using the `aws:PrincipalArn` condition key with a wildcard(*) in the `Principal` element, unless the identity-based policies contain an explicit deny.

Role session principals

You can specify role sessions in the `Principal` element of a resource-based policy or in condition keys that support principals. When a principal or identity assumes a role, they receive temporary security credentials with the assumed role's permissions. When they use those session credentials to perform operations in AWS, they become a *role session principal*.

The format that you use for a role session principal depends on the AWS STS operation that was used to assume the role.

Additionally, administrators can design a process to control how role sessions are issued. For example, they can provide a one-click solution for their users that creates a predictable session name. If your administrator does this, you can use role session principals in your policies or condition keys. Otherwise, you can specify the role ARN as a principal in the `aws:PrincipalArn` condition key. How you specify the role as a principal can change the effective permissions for the resulting session. For more information, see [IAM role principals \(p. 1266\)](#).

Assumed-role session principals

An *assumed-role session principal* is a session principal that results from using the AWS STS `AssumeRole` operation. For more information about which principals can assume a role using this operation, see [Comparing the AWS STS API operations \(p. 436\)](#).

To specify the assumed-role session ARN in the `Principal` element, use the following format:

```
"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.

Web identity session principals

A *web identity session principal* is a session principal that results from using the AWS STS `AssumeRoleWithWebIdentity` operation. You can use an external web identity provider (IdP) to sign in, and then assume an IAM role using this operation. This leverages identity federation and issues a role session. For more information about which principals can assume a role using this operation, see [Comparing the AWS STS API operations \(p. 436\)](#).

When you issue a role from a web identity provider, you get this special type of session principal that includes information about the web identity provider.

Use this principal type in your policy to allow or deny access based on the trusted web identity provider. To specify the web identity role session ARN in the `Principal` element of a role trust policy, use the following format:

```
"Principal": { "Federated": "cognito-identity.amazonaws.com" }
```

```
"Principal": { "Federated": "www.amazon.com" }
```

```
"Principal": { "Federated": "graph.facebook.com" }
```

```
"Principal": { "Federated": "accounts.google.com" }
```

SAML session principals

A *SAML session principal* is a session principal that results from using the AWS STS AssumeRoleWithSAML operation. You can use an external SAML identity provider (IdP) to sign in, and then assume an IAM role using this operation. This leverages identity federation and issues a role session. For more information about which principals can assume a role using this operation, see [Comparing the AWS STS API operations \(p. 436\)](#).

When you issue a role from a SAML identity provider, you get this special type of session principal that includes information about the SAML identity provider.

Use this principal type in your policy to allow or deny access based on the trusted SAML identity provider. To specify the SAML identity role session ARN in the Principal element of a role trust policy, use the following format:

```
"Principal": { "Federated": "arn:aws:iam::AWS-account-ID:saml-provider/provider-name" }
```

IAM user principals

You can specify IAM users in the Principal element of a resource-based policy or in condition keys that support principals.

Note

In a Principal element, the user name part of the [Amazon Resource Name \(ARN\) \(p. 1213\)](#) 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/user-name-2"  
    ]  
}
```

When you specify users in a Principal element, you cannot use a wildcard (*) to mean "all users". Principals must always name specific users.

Important

If your Principal element in a role trust policy contains an ARN that points to a specific IAM user, then IAM transforms the ARN to the user's unique principal ID when you save the policy. 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 you break the relationship. 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 appears in resource-based policies 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 Principal element, you must edit the role to replace the now incorrect principal ID with the correct ARN. IAM once again transforms ARN into the user's new principal ID when you save the policy.

IAM Identity Center principals

In IAM Identity Center, the principal in a resource-based policy must be defined as the AWS account principal. To specify access, reference the role ARN of the permission set in the condition block. For

details, see [Referencing permission sets in resource policies, Amazon EKS, and AWS KMS](#) in the *IAM Identity Center User Guide*.

AWS STS federated user session principals

You can specify *federated user sessions* in the `Principal` element of a resource-based policy or in condition keys that support principals.

Important

AWS recommends that you use AWS STS federated user sessions only when necessary, such as when [root user access is required](#). Instead, [use roles to delegate permissions \(p. 40\)](#).

An *AWS STS federated user session principal* is a session principal that results from using the AWS STS `GetFederationToken` operation. In this case, AWS STS uses [identity federation](#) as the method to obtain temporary access tokens instead of using IAM roles.

In AWS, IAM users or an AWS account root user can authenticate using long-term access keys. For more information about which principals can federate using this operation, see [Comparing the AWS STS API operations \(p. 436\)](#).

- **IAM federated user** – An IAM user federates using the `GetFederationToken` operation that results in a federated user session principal for that IAM user.
- **Federated root user** – A root user federates using the `GetFederationToken` operation that results in a federated user session principal for that root user.

When an IAM user or root user requests temporary credentials from AWS STS using this operation, they begin a temporary federated user session. This session's ARN is based on the original identity that was federated.

To specify the federated user session ARN in the `Principal` element, use the following format:

```
"Principal": { "AWS": "arn:aws:sts::AWS-account-ID:federated-user/user-name" }
```

AWS service principals

You can specify AWS services in the `Principal` element of a resource-based policy or in condition keys that support principals. A *service principal* is an identifier for a service.

IAM roles that can be assumed by an AWS service are called [service roles \(p. 185\)](#). Service roles must include a trust policy. *Trust policies* are resource-based policies attached to a role that defines 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.

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 \(p. 1224\)](#), 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.

```
"Principal": {  
    "Service": [  
        "ecs.amazonaws.com",  
        "elasticloadbalancing.amazonaws.com"  
    ]  
}
```

AWS service principals in opt-in Regions

You can launch resources in several AWS Regions and some of those Regions you must opt in to. For a complete list of Regions you must opt in to, see [Managing AWS Regions](#) in the *AWS General Reference* guide.

When an AWS service in an opt-in Region makes a request within the same Region, the service principal name format is identified as the non-regionalized version of their service principal name:

service-name.amazonaws.com

When an AWS service in an opt-in Region makes a cross-region request to another Region, the service principal name format is identified as the regionalized version of their service principal name:

service-name.{region}.amazonaws.com

For example, you have an Amazon SNS topic located in Region ap-southeast-1 and an Amazon S3 bucket located in opt-in Region ap-east-1. You want to configure S3 bucket notifications to publish messages to the SNS topic. To allow the S3 service to post messages to the SNS topic you must grant the S3 service principal sns:Publish permission via the resource-based access policy of the topic.

If you specify the non-regionalized version of the S3 service principal, s3.amazonaws.com, in the topic access policy, the sns:Publish request from the bucket to the topic will fail. The following example specifies the non-regionalized S3 service principal in the Principal policy element of the SNS topic access policy.

```
"Principal": { "Service": "s3.amazonaws.com" }
```

Since the bucket is located in an opt-in Region and the request is made outside of that same Region, the S3 service principal appears as the regionalized service principal name, s3.ap-east-1.amazonaws.com. You must use the regionalized service principal name when an AWS service in an opt-in Region makes a request to another Region. After you specify the regionalized service principal name, if the bucket makes an sns:Publish request to the SNS topic located in another Region, the request will be successful. The following example specifies the regionalized S3 service principal in the Principal policy element of the SNS topic access policy.

```
"Principal": { "Service": "s3.ap-east-1.amazonaws.com" }
```

Resource policies or service principal-based allow-lists for cross-Region requests from an opt-in Region to another Region will only be successful if you specify the regionalized service principal name.

Note

For IAM role trust policies, we recommend using the non-regionalized service principal name. IAM resources are global and therefore the same role can be used in any Region.

All principals

You can use a wildcard (*) to specify all principals in the Principal element of a resource-based policy or in condition keys that support principals. [Resource-based policies \(p. 486\)](#) grant permissions and [condition keys](#) are used to limit the conditions of a policy statement.

Important

We strongly recommend that you do not use a wildcard (*) in the Principal element of a resource-based policy with an Allow effect unless you intend to grant public or anonymous access. Otherwise, specify intended principals, services, or AWS accounts in the Principal element and then further restrict access in the Condition element. This is especially true for IAM role trust policies, because they allow other principals to become a principal in your account.

For resource-based policies, using a wildcard (*) with an Allow effect grants access to all users, including anonymous users (public access). For IAM users and role principals within your account, no other permissions are required. For principals in other accounts, they must also have identity-based permissions in their account that allow them to access your resource. This is called [cross-account access](#).

For anonymous users, the following elements are equivalent:

```
"Principal": "*"
```

```
"Principal" : { "AWS" : "*" }
```

You cannot use a wildcard to match part of a principal name or ARN.

The following example shows a resource-based policy that can be used instead of [Specifying NotPrincipal with Deny \(p. 1272\)](#) to explicitly deny all principals *except* for the ones specified in the Condition element.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "UsePrincipalArnInsteadOfNotPrincipalWithDeny",
      "Effect": "Deny",
      "Action": "s3:*",
      "Principal": "*",
      "Resource": [
        "arn:aws:s3:::BUCKETNAME/*",
        "arn:aws:s3:::BUCKETNAME"
      ],
      "Condition": {
        "ArnNotEquals": {
          "aws:PrincipalArn": "arn:aws:iam::44445556666:user/user-name"
        }
      }
    }
  ]
}
```

More information

For more information, see the following:

- [Bucket policy examples](#) in the *Amazon Simple Storage Service User 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 \(p. 199\)](#)

AWS JSON policy elements: NotPrincipal

You can use the `NotPrincipal` element to deny access to all principals *except* the IAM user, federated user, IAM role, AWS account, AWS service, or other principal specified in the `NotPrincipal` element.

You can use it in resource-based policies for some AWS services, including VPC endpoints. Resource-based policies are policies that you embed directly in a resource. You cannot use the `NotPrincipal` element in an IAM identity-based policy nor in an IAM role trust policy.

`NotPrincipal` must be used with `"Effect": "Deny"`. Using it with `"Effect": "Allow"` is not supported.

Important

Very few scenarios require the use of `NotPrincipal`. We recommend that you explore other authorization options before you decide to use `NotPrincipal`. When you use `NotPrincipal`, troubleshooting the effects of multiple policy types can be difficult. We recommend using the `aws:PrincipalArn` condition key instead. For more information, see [All principals \(p. 1270\)](#).

Specifying `NotPrincipal` with Deny

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/*"
            ]
        }
    ]
}

```

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 User 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": "sns: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": [ "sns:SendMessage", "sns: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": {
                "Bool": "false"
            }
        }
    ]
}
```

```
        "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 \(p. 543\)](#).

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 \(p. 1213\)](#).

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 you want to write a statement.

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.

Note

In the Resource element, the IAM user name is case sensitive.

```
"Resource": "arn:aws:iam::account-ID-without-hyphens:user/Bob"
```

Using wildcards in resource ARNs

You can use wildcards as part of the resource ARN. You can use wildcard characters (*) and (?) within ARN segments (the parts separated by colons) to represent any combination of characters with an asterisk (*) and any single character with a question mark (?). You can use multiple * or ? characters in each segment.

Note

You can't use a wildcard in the service segment that identifies the AWS product. For more information about ARN segments, see [Amazon Resource Names \(ARNs\) \(p. 1211\)](#)

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:::DOC-EXAMPLE-BUCKET/*"
```

The asterisk (*) character can expand to replace everything within a segment, including characters like a forward slash (/) that may otherwise appear to be a delimiter within a given service namespace. For example, consider the following Amazon S3 ARN as the same wildcard expansion logic applies to all services.

```
"Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*/test/*"
```

The wildcards in the ARN apply to all of the following objects in the bucket, not only the first object listed.

```
DOC-EXAMPLE-BUCKET/1/test/object.jpg  
DOC-EXAMPLE-BUCKET/1/2/test/object.jpg  
DOC-EXAMPLE-BUCKET/1/2/test/3/object.jpg  
DOC-EXAMPLE-BUCKET/1/2/3/test/4/object.jpg  
DOC-EXAMPLE-BUCKET/1///test///object.jpg  
DOC-EXAMPLE-BUCKET/1/test/.jpg  
DOC-EXAMPLE-BUCKET//test/object.jpg  
DOC-EXAMPLE-BUCKET/1/test/
```

Consider the last two objects in the previous list. An Amazon S3 object name can validly begin or end with the conventional delimiter forward slash (/) character. While "/" works as a delimiter, there is no specific significance when this character is used within a resource ARN. It is treated the same as any other valid character. The ARN would not match the following objects:

```
DOC-EXAMPLE-BUCKET/1-test/object.jpg  
DOC-EXAMPLE-BUCKET/test/object.jpg  
DOC-EXAMPLE-BUCKET/1/2/test.jpg
```

Specifying multiple resources

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"  
]
```

Using policy variables in resource ARNs

In the Resource element, you can use JSON [policy variables \(p. 1298\)](#) 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:table/${aws:username}"  
    }  
}
```

For more information about JSON policy variables, see [IAM policy elements: Variables and tags \(p. 1298\)](#).

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 \(p. 1281\)](#) (equal, less than, and others) to match the context keys and values in the policy against keys and values in the request context. To learn more about the request context, see [Request \(p. 8\)](#).

```
"Condition" : { "{condition-operator}" : { "{condition-key}" : "{condition-value}" }}
```

The context key that you specify in a policy condition can be a [global condition context key \(p. 1338\)](#) or a service-specific context key. Global condition context keys have the aws : prefix. Service-specific context keys have the service's prefix. For example, Amazon EC2 lets you write a condition using the ec2:InstanceType context key, which is unique to that service. To view service-specific IAM context keys with the iam: prefix, see [IAM and AWS STS condition context keys \(p. 1367\)](#).

Context key *names* are not case-sensitive. For example, including the aws:SourceIP context key is equivalent to testing for AWS:SourceIp. Case-sensitivity of context key *values* depends on the [condition operator \(p. 1281\)](#) that you use. For example, the following condition includes the StringEquals operator to make sure that only requests made by johndoe match. Users named JohnDoe are denied access.

```
"Condition" : { "StringEquals" : { "aws:username" : "johndoe" }}
```

The following condition uses the [StringEqualsIgnoreCase \(p. 1282\)](#) operator to match users named johndoe or JohnDoe.

```
"Condition" : { "StringEqualsIgnoreCase" : { "aws:username" : "johndoe" }}
```

Some context keys support key-value pairs that allow you to specify part of the key name. Examples include the [aws:RequestTag/tag-key \(p. 1355\)](#) context key, the AWS KMS [kms:EncryptionContext:encryption_context_key](#), and the [ResourceTag/tag-key \(p. 1361\)](#) context key supported by multiple services.

- If you use the ResourceTag/[tag-key](#) context 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 "aws: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. For example, you might tag an Amazon EC2 instance with ec2=test1 and EC2=test2. When you use a condition such as "aws:ResourceTag/EC2": "test1" 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 \(p. 1364\)](#) context 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 they work, see [Condition operators \(p. 1281\)](#).
- Unless otherwise specified, all context keys can have multiple values. For a description of how to handle context keys that have multiple values, see [Multivalued context keys \(p. 1294\)](#).
- For a list of all of the globally available context keys, see [AWS global condition context keys \(p. 1338\)](#).
- For condition context keys that are defined by each service, see [Actions, Resources, and Condition Keys for AWS Services](#).

The request context

When a [principal \(p. 7\)](#) makes a [request \(p. 8\)](#) 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 context keys against the request context. For

example, you can create a policy that uses the [aws:CurrentTime \(p. 1341\)](#) context key to [allow a user to perform actions within only a specific range of dates \(p. 532\)](#).

When a request is submitted, AWS evaluates each context key in the policy and returns a value of *true*, *false*, *not present*, and occasionally *null* (an empty data string). A context key that is not present in the request is considered a mismatch. For example, the following policy allows removing your own multi-factor 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"
            ],
            "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 context 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 \(p. 1294\)](#) condition operator with the [aws:TagKeys \(p. 1364\)](#) context key, you can experience unexpected results if the request context returns *null*. For more information, see [aws:TagKeys \(p. 1364\)](#) and [Multivalued context keys \(p. 1294\)](#).

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 context 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 \(p. 1281\)](#) (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 \(p. 1282\)](#) or [ARN operators \(p. 1288\)](#), you can also use a [policy variable \(p. 1298\)](#) in the context key value. The following example includes the `aws:username` variable.

```
"Condition": {"StringLike": {"s3:prefix": ["${aws:username}/*"]}}
```

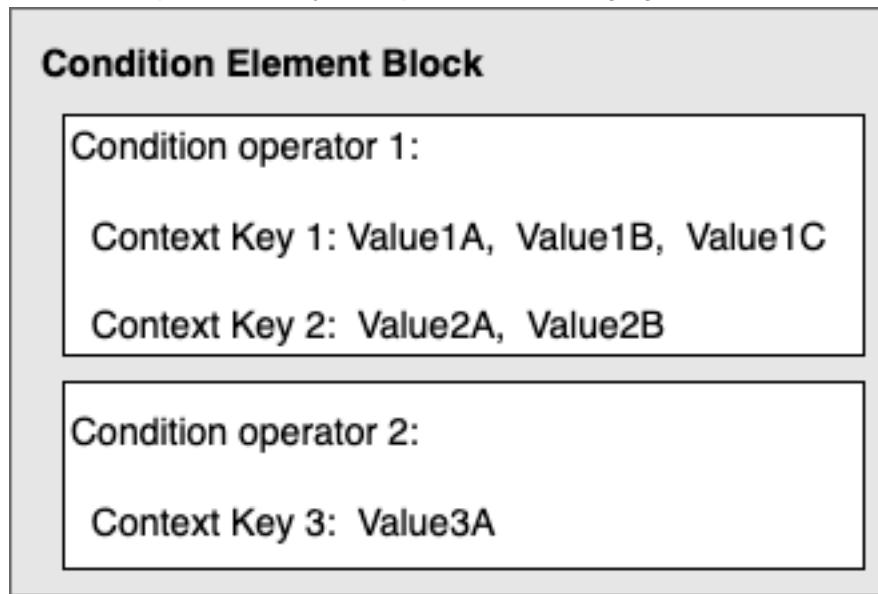
Under some circumstances, context keys can contain multiple values. For example, a request to Amazon DynamoDB might ask to return or update multiple attributes from a table. A policy for access to DynamoDB tables can include the `dynamodb:Attributes` context key, which contains all the attributes

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 [Multivalued context keys \(p. 1294\)](#).

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 condition operators, and each condition operator can contain multiple context key-value pairs. The following figure illustrates this.



For more information, see [Multivalued context keys \(p. 1294\)](#).

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 \(p. 1278\)](#).

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 \(p. 1338\)](#). 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 and the condition is *false*. If the policy condition requires that the key is *not* matched, such as StringNotLike or ArnNotLike, and the right key is not present, the condition is *true*. This logic applies to all condition operators except [...IfExists \(p. 1288\)](#) and [Null check \(p. 1290\)](#). These operators test whether the key is present (exists) in the request context.

The condition operators can be grouped into the following categories:

- [String \(p. 1282\)](#)

- [Numeric \(p. 1284\)](#)
- [Date and time \(p. 1285\)](#)
- [Boolean \(p. 1285\)](#)
- [Binary \(p. 1286\)](#)
- [IP address \(p. 1286\)](#)
- [Amazon Resource Name \(ARN\) \(p. 1288\)](#) (available for only some services.)
- [...IfExists \(p. 1288\)](#) (checks if the key value exists as part of another check)
- [Null check \(p. 1290\)](#) (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 multi-character match wildcards (*) and single-character match wildcards (?) anywhere in the string. You must specify wildcards to achieve partial string matches. Note If a key contains multiple values, StringLike can be qualified with set operators —ForAllValues:StringLike and ForAnyValue:StringLike. For more information, see Multivalued context keys (p. 1294) .
StringNotLike	Negated case-sensitive matching. The values can include multi-character match wildcards (*) or single-character match wildcards (?) anywhere in the string.

For example, the following statement contains a Condition element that uses `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: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 \(p. 1298\)](#) with the `String` condition operator.

The following example uses the `StringLike` condition operator to perform string matching with a [policy variable \(p. 1298\)](#) 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/${aws:username}/"
        ]}}
    },
    {
      "Effect": "Allow",
      "Action": "s3:*",
      "Resource": [
        "arn:aws:s3:::BUCKET-NAME/home/${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 \(p. 575\)](#).

Wildcard matching

String condition operators perform a patternless matching that does not enforce a predefined format. ARN and Date condition operators are a subset of string operators that enforce a structure on the condition key value. When you use `StringLike` or `StringNotLike` operators for partial string matches of an ARN or date, the matching ignores which portion of the structure is wildcarded.

For example, the following conditions search for a partial match of an ARN using different condition operators.

When `ArnLike` is used, the partition, service, account-id, resource-type, and partial resource-id portions of the ARN must have exact matching to the ARN in the request context. Only the region and resource path allow partial matching.

```
"Condition": {"ArnLike": {"aws:SourceArn": "arn:aws:cloudtrail:*:111122223333:trail/*"}}
```

When StringLike is used instead of ArnLike, matching ignores the ARN structure and allows partial matching, regardless of the portion that was wildcarded.

```
"Condition": {"StringLike": {"aws:SourceArn": "arn:aws:cloudtrail:*:111122223333:trail/*"}}
```

ARN	ArnLike	StringLike
arn:aws:cloudtrail:us-west-2:111122223333:trail/finance	Match	Match
arn:aws:cloudtrail:us-east-2:111122223333:trail/finance/archive	Match	Match
arn:aws:cloudtrail:us-east-2: 444455556666:user/111122223333:trail/finance	No match	Match

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
NumericEquals	Matching
NumericNotEquals	Negated matching
NumericLessThan	"Less than" matching
NumericLessThanEquals	"Less than or equals" matching
NumericGreaterThan	"Greater than" matching
NumericGreaterThanEquals	"Greater than or equals" matching

For example, the following statement 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 \(p. 1298\)](#) 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 [aws:CurrentTime](#) key or [aws:EpochTime](#) key. 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: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. The 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 \(p. 1298\)](#) 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, this identity-based policy uses the Bool condition operator with the [aws:SecureTransport](#) key to denies replicating objects and object tags to the destination bucket and its contents if the request is not over SSL.

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": "BooleanExample",
            "Action": "s3:ReplicateObject",
            "Effect": "Deny",
            "Resource": [
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET",
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*"
            ],
            "Condition": {
                "Bool": {
                    "aws:SecureTransport": "false"
                }
            }
        }
    ]
}
```

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` request context returns true or false.

You can use a [policy variable \(p. 1298\)](#) 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" : "QmluYXJ5VmFsdWVjbkJhc2U2NA=="
    }
}
```

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 \(p. 1298\)](#) 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
IpAddress	The specified IP address or range
NotIpAddress	All IP addresses except the specified IP address or range

For example, the following statement uses the IpAddress condition 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: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 \(p. 1298\)](#) with the IpAddress 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 update 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.

Condition operator	Description
ArnEquals, ArnLike	Case-sensitive matching of the ARN. Each of the six colon-delimited components of the ARN is checked separately and each can include multi-character match wildcards (*) or single-character match wildcards (?). The ArnEquals and ArnLike condition operators behave identically.
ArnNotEquals, ArnNotLike	Negated matching for ARN. The ArnNotEquals and ArnNotLike condition operators behave identically.

You can use a [policy variable \(p. 1298\)](#) 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. If you are using an "Effect": "Deny" element with a negated condition operator like StringNotEqualsIfExists, the request is still denied even if the tag is missing.

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.*",
        "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 requires accessing many resources in addition to the instance itself; for example, images, key pairs, security groups, and more. 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 policy 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 asterisk (*) in the condition key values, when used with the StringLikeIfExists condition operator, is interpreted as a wildcard to achieve partial string matches. The DescribeActions statement includes the actions required to view the instance in the console.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "RunInstance",
      "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 absent 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 \(p. 1298\)](#) 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"}}
  }
}
```

Conditions with multiple context keys or values

You can use the `Condition` element of a policy to test multiple context keys or multiple values for a single context 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. You can use context keys to test the values of the matching context keys in the request, with the context keys specified in the policy condition. To learn about information and data included in a request, see [The request context \(p. 1279\)](#).

Topics

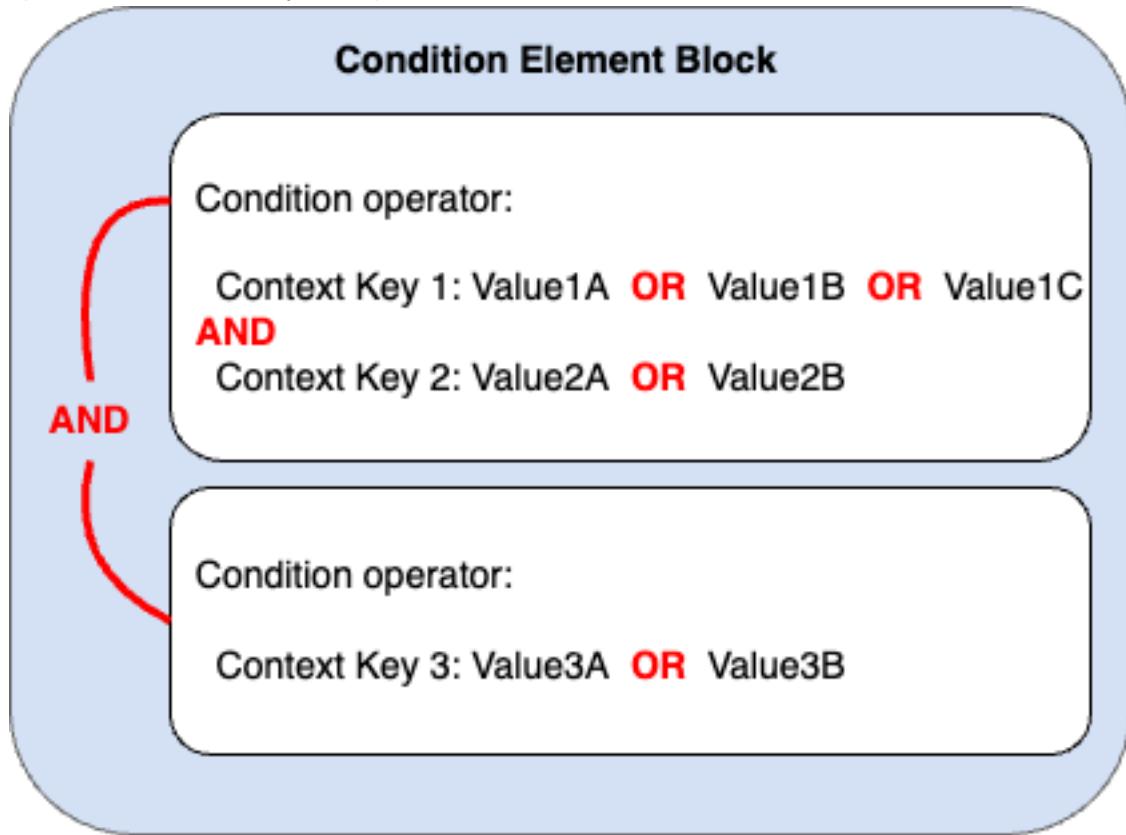
- [Evaluation logic for multiple context keys or values \(p. 1290\)](#)
- [Evaluation logic for negated matching condition operators \(p. 1292\)](#)

Evaluation logic for multiple context keys or values

A `Condition` element can contain multiple condition operators, and each condition operator can contain multiple context key-value pairs. Most context keys support using multiple values, unless otherwise specified.

- If your policy statement has multiple [condition operators \(p. 1281\)](#), the condition operators are evaluated using a logical AND.
- If your policy statement has multiple context keys attached to a single condition operator, the context keys are evaluated using a logical AND.
- If a single condition operator includes multiple values for a context key, those values are evaluated using a logical OR.
- If a single negated matching condition operator includes multiple values for a context key, those values are evaluated using a logical NOR.

All context keys in a condition element block must resolve to true to invoke the desired Allow or Deny effect. The following figure illustrates the evaluation logic for a condition with multiple condition operators and context key-value pairs.



For example, the following S3 bucket policy illustrates how the previous figure is represented in a policy. The condition block includes condition operators `StringEquals` and `ArnLike`, and context keys `aws:PrincipalTag` and `aws:PrincipalArn`. To invoke the desired Allow or Deny effect, all context keys in the condition block must resolve to true. The user making the request must have both principal tag keys, `department` and `role`, that include one of the tag key values specified in the policy. Also, the principal ARN of the user making the request must match one of the `aws:PrincipalArn` values specified in the policy to be evaluated as true.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ExamplePolicy",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::222222222222:root"
      },
      "Action": "s3>ListBucket",
      "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalTag/department": [
            "finance",
            "hr",
            "legal"
          ],
          "aws:PrincipalTag/role": [
            "Editor"
          ]
        }
      }
    }
  ]
}
```

```

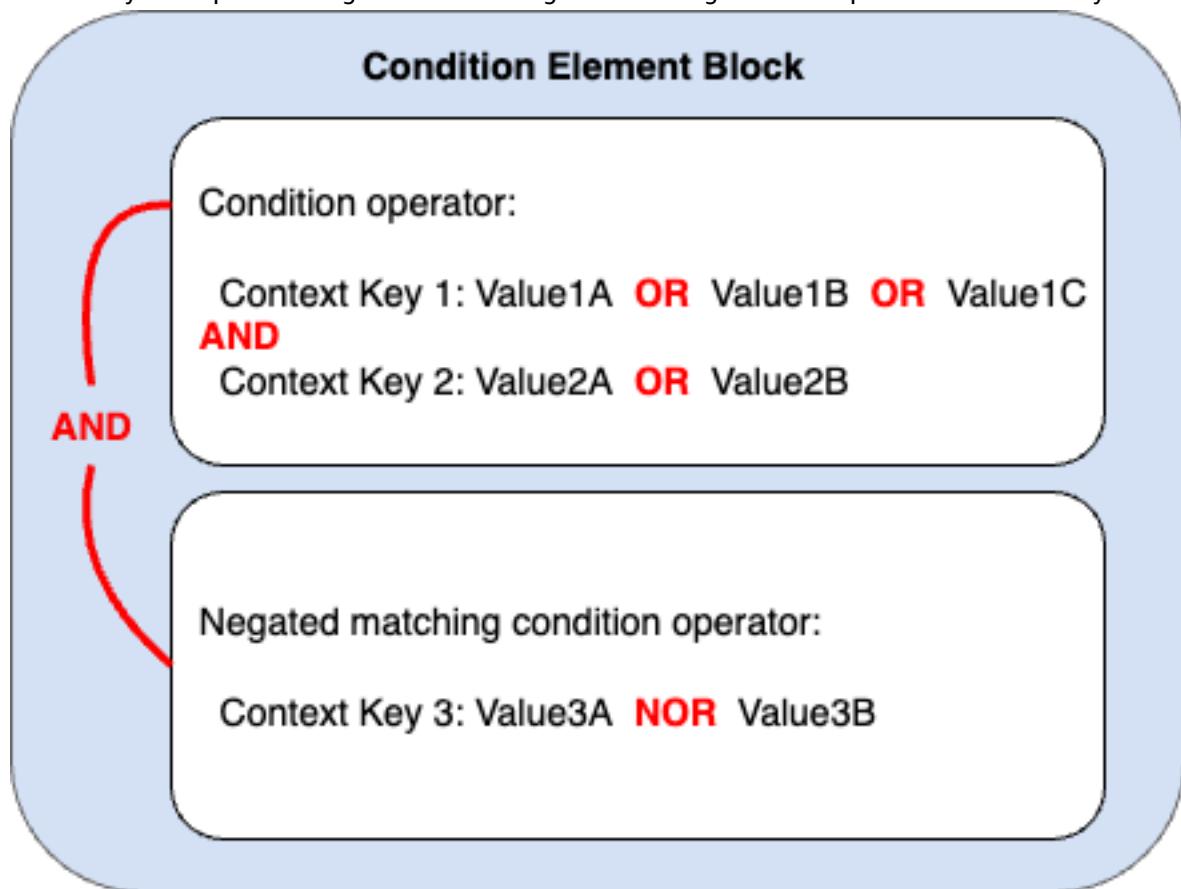
        "audit",
        "security"
    ],
},
"ArnLike": {
    "aws:PrincipalArn": [
        "arn:aws:iam::222222222222:user/Ana",
        "arn:aws:iam::222222222222:user/Mary"
    ]
}
}
]
}
}

```

Evaluation logic for negated matching condition operators

Some [condition operators](#), (p. 1281) such as `StringNotEquals` or `ArnNotLike`, use negated matching to compare the context key-value pairs in your policy against the context key-value pairs in a request. When multiple values are specified for a single context key in a policy with negated matching condition operators, the effective permissions work like a logical NOR. In negated matching, a logical NOR or NOT OR returns true only if all values evaluate to false.

The following figure illustrates the evaluation logic for a condition with multiple condition operators and context key-value pairs. The figure includes a negated matching condition operator for context key 3.



For example, the following S3 bucket policy illustrates how the previous figure is represented in a policy. The condition block includes condition operators `StringEquals` and `ArnNotLike`, and context keys `aws:PrincipalTag` and `aws:PrincipalArn`. To invoke the desired Allow or Deny effect, all context

keys in the condition block must resolve to true. The user making the request must have both principal tag keys, *department* and *role*, that include one of the tag key values specified in the policy. Since the `ArnNotLike` condition operator uses negated matching, the principal ARN of the user making the request must not match any of the `aws:PrincipalArn` values specified in the policy to be evaluated as true.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ExamplePolicy",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::222222222222:root"
      },
      "Action": "s3>ListBucket",
      "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalTag/department": [
            "finance",
            "hr",
            "legal"
          ],
          "aws:PrincipalTag/role": [
            "audit",
            "security"
          ]
        },
        "ArnNotLike": {
          "aws:PrincipalArn": [
            "arn:aws:iam::222222222222:user/Ana",
            "arn:aws:iam::222222222222:user/Mary"
          ]
        }
      }
    }
  ]
}
```

Single-valued vs. multivalued context keys

The difference between single-valued and multivalued context keys depends on the number of values in the [request context \(p. 8\)](#), not the number of values in the policy condition.

- *Single-valued* condition context keys have at most one value in the request context. For example, you can tag resources in AWS. Resource tags are stored as tag key-value pairs. A resource tag key can have a single tag value. Therefore, [the section called "ResourceTag" \(p. 1361\)](#) is a single-valued context key. Do not use a condition set operator with a single-valued context key.
- *Multivalued* condition context keys can have multiple values in the request context. For example, you can tag resources in AWS and include multiple tag key-value pairs in a request. Therefore, [the section called "TagKeys" \(p. 1364\)](#) is a multivalued context key. Multivalued context keys require a condition set operator.

Important

Multivalued context keys require a condition set operator. Do not use condition set operators `ForAllValues` or `ForAnyValue` with single-valued context keys. To learn more about condition set operators, see [Multivalued context keys \(p. 1294\)](#).

The *Single-valued* and *Multivalued* classifications are included in the description of each condition context key as *Value type* in the [AWS global condition context keys \(p. 1338\)](#) topic. The [Service Authorization Reference](#) uses a different value type classification for multivalued context keys in the following format: an `ArrayOf` prefix followed by the condition operator category type. For example, `ArrayOfString` or `ArrayOfARN`.

For example, a request can originate from at most one VPC endpoint, so [the section called "SourceVpce" \(p. 1364\)](#) is a single-valued context key. Since a service can have more than one service principal name that belongs to the service, [aws:PrincipalServiceNamesList \(p. 1351\)](#) is a multivalued context key.

You can use any available single-valued context key as a policy variable. You cannot use a multivalued context key as a policy variable. For more information about policy variables, see [IAM policy elements: Variables and tags \(p. 1298\)](#).

Multivalued context keys require condition set operators `ForAllValues` or `ForAnyValue`. Context keys that include key-value pairs such as [the section called "RequestTag" \(p. 1355\)](#) and [the section called "ResourceTag" \(p. 1361\)](#) can cause confusion because there can be multiple *tag-key* values. But since each *tag-key* can have only one value, `aws:RequestTag` and `aws:ResourceTag` are both single-valued context keys. Using condition set operators with single-valued context keys can lead to overly permissive policies.

Multivalued context keys

To compare your condition context key against a [request context \(p. 8\)](#) key with multiple values, you must use the `ForAllValues` or `ForAnyValue` set operators. These set operators are used to compare two sets of values, such as the set of tags in a request and the set of tags in a policy condition.

The `ForAllValues` and `ForAnyValue` qualifiers add set-operation functionality to the condition operator so that you can test request context keys with multiple values against multiple context key values in a policy condition. Additionally, if you include a multivalued string context key in your policy with a wildcard or a variable, you must also use the `StringLike` [condition operator \(p. 1282\)](#). For requests that include multiple values for a single context key, you must enclose the condition context key values within brackets like an [array \(p. 1324\)](#). For example, "Key2": ["Value2A", "Value2B"].

- `ForAllValues` – This qualifier tests whether the value of every member of the request set is a subset of the condition context key set. The condition returns true if every context key value in the request matches at least one context key value in the policy. It also returns true if there are no context keys in the request, or if the context key value resolves to a null dataset, such as an empty string. To prevent missing context keys or context keys with empty values from evaluating to true, you can include the [Null \(p. 1290\)](#) condition operator in your policy with a false value to check if the context key exists and its value is not null.

Important

Use caution if you use `ForAllValues` with an Allow effect because it can be overly permissive if the presence of missing context keys or context keys with empty values in the request context is unexpected. You can include the `Null` condition operator in your policy with a false value to check if the context key exists and its value is not null. For an example, see [Controlling access based on tag keys \(p. 525\)](#).

- `ForAnyValue` – This qualifier tests whether at least one member of the set of request context key values matches at least one member of the set of context key values in your policy condition. The context key returns true if any one of the context key values in the request matches any one of the context key values in the policy. For no matching context key or a null dataset, the condition returns false.

Note

The difference between single-valued and multivalued context keys depends on the number of values in the request context, not the number of values in the policy condition.

Condition policy examples

In IAM policies, you can specify multiple values for both single-valued and multivalued context keys for comparison against the request context. The following set of policy examples demonstrates policy conditions with multiple context keys and values.

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. For IAM identity-based policy examples, see [Example IAM identity-based policies \(p. 529\)](#).

Condition policy examples: Single-valued context keys

- Multiple condition blocks with single-valued context keys. ([View this example \(p. 1296\)](#).)
- One condition block with multiple single-valued context keys and values. ([View this example \(p. 1297\)](#).)

Condition policy examples: Multivalued context keys

- Deny policy with condition set operator ForAllValues. ([View this example \(p. 1295\)](#).)
- Deny policy with condition set operator ForAnyValue. ([View this example \(p. 1296\)](#).)

Multivalued context key examples

The following set of policy examples demonstrate how to create policy conditions with multivalued context keys.

Example: Deny policy with condition set operator ForAllValues

The following example identity-based policy denies the use of IAM tagging actions when specific tag key prefixes are included in the request. Each value for context key `aws:TagKeys` includes a wildcard (*) for partial string matching. The policy includes the `ForAllValues` set operator with context key `aws:TagKeys` because the request context key can include multiple values. In order for context key `aws:TagKeys` to return true, every value in the request must match at least one value in the policy.

The `ForAllValues` set operator also returns true if there are no context keys in the request, or if the context key value resolves to a null dataset, such as an empty string. To prevent missing context keys or context keys with empty values from evaluating to true, include the `Null` condition operator in your policy with a value of `false` to check if the context key in the request exists and its value is not null.

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": "DenyRestrictedTags",  
            "Effect": "Deny",  
            "Action": [  
                "iam:Tag*",  
                "iam:UnTag*"  
            ],  
            "Resource": [  
                "*"  
            ],  
            "Condition": {  
                "Null": {  
                    "aws:TagKeys": "  
                }  
            }  
        }  
    ]  
}
```

```
        "aws:TagKeys": "false"
    },
    "ForAllValues:StringLike": [
        "aws:TagKeys": [
            "key1*",
            "key2*",
            "key3*"
        ]
    }
}
]
```

Example: Deny policy with condition set operator ForAnyValue

The following identity-based policy example denies creating snapshots of EC2 instance volumes if any snapshots are tagged with one of the tag keys specified in the policy, environment or webserver. The policy includes the `ForAnyValue` set operator with context key `aws:TagKeys` because the request context key can include multiple values. If your tagging request includes any one of the tag key values specified in the policy, the `aws:TagKeys` context key returns true invoking the deny policy effect.

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": [  
        {  
            "Effect": "Deny",  
            "Action": [  
                "ec2:CreateSnapshot",  
                "ec2:CreateSnapshots"  
            ],  
            "Resource": "arn:aws:ec2:us-west-2::snapshot/*",  
            "Condition": {  
                "ForAnyValue:StringEquals": {  
                    "aws:TagKeys": ["environment", "webserver"]  
                }  
            }  
        }  
    ]  
}
```

Single-valued context key policy examples

The following set of policy examples demonstrate how to create policy conditions with single-valued context keys.

Example: Multiple condition blocks with single-valued context keys

When a condition block has multiple conditions, each with a single context key, all context keys must resolve to true for the desired Allow or Deny effect to be invoked. When you use negated matching condition operators, the evaluation logic of the condition value is reversed.

The following example lets users create EC2 volumes and apply tags to the volumes during volume creation. The request context must include a value for context key `aws:RequestTag/project`, and the value for context key `aws:ResourceTag/environment` can be anything except production.

{

```

    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "ec2:CreateVolume",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "ec2:CreateTags",
            "Resource": "arn:aws:ec2:::volume/*",
            "Condition": {
                "StringLike": {
                    "aws:RequestTag/project": "*"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "ec2:CreateTags",
            "Resource": "arn:aws:ec2:region:account:*/",
            "Condition": {
                "StringNotEquals": {
                    "aws:ResourceTag/environment": "production"
                }
            }
        }
    ]
}

```

The request context must include a project tag-value and cannot be created for a production resource to invoke the Allow effect. The following EC2 volume is successfully created because the project name is Feature3 with a QA resource tag.

```

aws ec2 create-volume \
    --availability-zone us-east-1a \
    --volume-type gp2 \
    --size 80 \
    --tag-specifications 'ResourceType=volume,Tags=[{Key=project,Value=Feature3}, \
    {Key=env,Value=QA}]'

```

Example: One condition block with multiple single-valued context keys and values

When a condition block contains multiple context keys and each context key has multiple values, each context key must resolve to true for at least one key value for the desired Allow or Deny effect to be invoked. When you use negated matching condition operators, the evaluation logic of the context key value is reversed.

The following example allows users to start and run tasks on Amazon Elastic Container Service clusters.

- The request context must include production **OR** pre-prod for the aws:RequestTag/ environment context key **AND**.
- The ecs:cluster context key makes sure that tasks are run on either the default1 **OR** default2 ARN ECS clusters.

```

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

```

```

    "Action": [
        "ecs:RunTask",
        "ecs:StartTask"
    ],
    "Resource": [
        "*"
    ],
    "Condition": {
        "StringEquals": {
            "aws:RequestTag/environment": [
                "production",
                "prod-backup"
            ]
        },
        "ArnEquals": {
            "ecs:cluster": [
                "arn:aws:ecs:us-east-1:111122223333:cluster/default1",
                "arn:aws:ecs:us-east-1:111122223333:cluster/default2"
            ]
        }
    }
}
]
}

```

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. 1288)

Topics

- [Introduction \(p. 1298\)](#)
- [Using variables in policies \(p. 1299\)](#)
- [Tags as policy variables \(p. 1301\)](#)
- [Where you can use policy variables \(p. 1301\)](#)
- [Policy variables with no value \(p. 1302\)](#)
- [Request information that you can use for policy variables \(p. 1303\)](#)
- [Specifying default values \(p. 1305\)](#)
- [For more information \(p. 1305\)](#)

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 users to list, read, and write objects in the S3 bucket `DOC-EXAMPLE-BUCKET` for marketing projects.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["s3>ListBucket"],

```

```

    "Resource": ["arn:aws:s3:::DOC-EXAMPLE-BUCKET"],
    "Condition": {"StringLike": {"s3:prefix": ["marketing/*"]}}
},
{
    "Effect": "Allow",
    "Action": [
        "s3:GetObject",
        "s3:PutObject"
    ],
    "Resource": ["arn:aws:s3:::DOC-EXAMPLE-BUCKET/marketing/*"]
}
]
}

```

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. Instead of creating a separate policy for each user, we recommend you create a single group policy that works for any user in that group.

Using variables in policies

You can define dynamic values inside policies by using *policy variables* that set placeholders in a policy.

Variables are marked using a \$ prefix followed by a pair of curly braces ({}) that include the variable name of the value from the request.

When the policy is evaluated, the policy variables are replaced with values that come from the conditional context keys passed in the request. Global condition context keys can be used as variables in requests across AWS services. Service specific condition keys can also be used as variables when interacting with AWS resources, but are only available when requests are made against resources which support them. For a list of context keys available for each AWS service and resource, see the [Service Authorization Reference](#).

Variables can be used in [identity-based policies](#), [resource policies](#), [service control policies](#), [session policies](#), and [VPC endpoint policies](#). Identity-based policies used as permissions boundaries also support policy variables.

Under certain circumstances, you can't populate global condition context keys with a value. To learn more about each key, see [AWS global condition context keys](#).

Important

- Key names are case-insensitive. For example, aws:CurrentTime is equivalent to AWS:currenttime.
- You can use any single-valued condition key as a variable. You can't use a multivalued condition key as a variable.

The following example shows a policy for an IAM role or user that replaces a specific resource name with a policy variable. You can reuse this policy by taking advantage of the aws:PrincipalTag condition key. When this policy is evaluated, \${aws:PrincipalTag/team} allows the actions only if the bucket name ends with a team name from the team principal tag.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["s3>ListBucket"],
            "Resource": ["arn:aws:s3:::DOC-EXAMPLE-BUCKET"],
            "Condition": {"StringLike": {"s3:prefix": ["${aws:PrincipalTag/team}/*"]}}
        }
    ]
}
```

```

        "Condition": {"StringLike": {"s3:prefix": ["${aws:PrincipalTag/team}/*"]}}
    },
    {
        "Effect": "Allow",
        "Action": [
            "s3:GetObject",
            "s3:PutObject"
        ],
        "Resource": ["arn:aws:s3:::DOC-EXAMPLE-BUCKET/${aws:PrincipalTag/team}/*"]
    }
]
}

```

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.

For details about this global condition key, see [aws:PrincipalTag/tag-key \(p. 1352\)](#) in the list of global condition keys.

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" \(p. 1261\)](#). To learn more about policy versions, see [the section called "Versioning IAM policies" \(p. 606\)](#).

A policy that allows a principal to get objects from the /David path of an S3 bucket looks like this:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["s3:GetObject"],
            "Resource": ["arn:aws:s3:::DOC-EXAMPLE-BUCKET/David/*"]
        }
    ]
}
```

If this policy is attached to user David, that user get objects from his own S3 bucket, but 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 policies that can be reused. The following policy allows a user to get objects from an Amazon S3 bucket if the tag-key value for `aws:PrincipalTag` matches the tag-key owner value passed in the request.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowUnlessOwnedBySomeoneElse",
            "Effect": "Allow",
            "Action": ["s3:GetObject"],
            "Resource": ["*"],
            "Condition": {

```

```

        "StringEquals": {
            "${s3:ExistingObjectTag/owner)": "${aws:PrincipalTag/owner}"
        }
    }
}
]
}
}

```

When you use a policy variable in place of a user like this, you don't have to have a separate policy for each individual user. In the following example, the policy is attached to an IAM role that is assumed by Product Managers using temporary security credentials. When a user makes a request to add an Amazon S3 object, IAM substitutes the dept tag value from the current request for the \${aws:PrincipalTag} variable and evaluates the policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowOnlyDeptS3Prefix",
            "Effect": "Allow",
            "Action": ["s3:GetObject"],
            "Resource": ["arn:aws:s3:::DOC-EXAMPLE-BUCKET/${aws:PrincipalTag/dept}/*"]
        }
    ]
}
```

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. These tags are key-value pairs. You define the tag key name and the value that is 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 resources \(p. 399\)](#). 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 resources 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 tags \(p. 521\)](#).

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 fifth colon (:). You can't use a variable to replace parts of the ARN before the fifth colon, such as the service or account. For more information about the ARN format, see [IAM ARNs \(p. 1213\)](#).

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}"]
```

Many AWS resources use ARNs that contain a user-created name. The following IAM policy ensures that only intended users with matching access-project, access-application, and access-environment tag values

can modify their resources. In addition, using [* wildcard matches](#), they are able to allow for custom resource name suffixes.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowAccessBasedOnArnMatching",
      "Effect": "Allow",
      "Action": [
        "sns:CreateTopic",
        "sns:DeleteTopic"],
      "Resource": ["arn:aws:sns:*:*:${aws:PrincipalTag/access-project}-
${aws:PrincipalTag/access-application}-${aws:PrincipalTag/access-environment}-*"]
    }
  ]
}
```

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 \(p. 1281\)](#).

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 users, but only if the tag `costCenter` is attached to the user. 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": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iam:*user*"
      ],
      "Resource": "*",
      "Condition": {
        "StringLike": {
          "iam:ResourceTag/costCenter": [ "12345", "67890" ]
        }
      }
    }
  ]
}
```

Policy variables with no value

When policy variables reference a condition context key that has no value or is not present in an authorization context for a request, the value is effectively null. There is no equal or like value. Condition context keys may not be present in the authorization context when:

- You are using service specific condition context keys in requests to resources that do not support that condition key.
- Tags on IAM principals, sessions, resources, or requests are not present.

- Other circumstances as listed for each global condition context key in [AWS global condition context keys \(p. 1338\)](#).

When you use a variable with no value in the condition element of an IAM policy, [IAM JSON policy elements: Condition operators \(p. 1281\)](#) like `StringEquals` or `StringLike` do not match, and the policy statement does not take effect.

Inverted condition operators like `StringNotEquals` or `StringNotLike` do match against a null value, as the value of the condition key they are testing against is not equal to or like the effectively null value.

In the following example, `aws:principalTag/Team` must be equal to `s3:ExistingObjectTag/Team` to allow access. Access is explicitly denied when `aws:principalTag/Team` is not set. If a variable that has no value in the authorization context is used as part of the `Resource` or `NotResource` element of a policy, the resource that includes a policy variable with no value will not match any resource.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Deny",  
            "Action": "s3:GetObject",  
            "Resource": "arn:aws:s3:::/example-bucket/*",  
            "StringNotEquals": {  
                "s3:ExistingObjectTag/Team": "${aws:PrincipalTag/Team}"  
            }  
        }  
    ]  
}
```

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 \(p. 1306\)](#) 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.

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 \(p. 1218\)](#)
 - `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 \(p. 203\)](#).

- 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 User Guide](#)
- [Common scenarios for temporary credentials \(p. 426\)](#)

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.

Specifying default values

To add a default value to a variable, surround the default value with single quotes (' '), and separate the variable text and the default value with a comma and space (,).

For example, if a principal is tagged with team=yellow, they can access ExampleCorp's Amazon S3 bucket named *DOC-EXAMPLE-BUCKET*-yellow. A policy with this resource allows team members to access their team bucket, but not those of other teams. For users without team tags, it sets a default value of company-wide for the bucket name. These users can access only the *DOC-EXAMPLE-BUCKET*-company-wide bucket where they can view broad information, such as instructions for joining a team.

```
"Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET-${aws:PrincipalTag/team, 'company-wide'}"
```

For more information

For more information about policies, see the following:

- [Policies and permissions in IAM \(p. 485\)](#)
- [Example IAM identity-based policies \(p. 529\)](#)
- [IAM JSON policy elements reference \(p. 1260\)](#)
- [Policy evaluation logic \(p. 1306\)](#)
- [About web identity federation \(p. 199\)](#)

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. 1306)** – AWS processes the information gathered in the request to determine which policies apply to the request.
3. **Evaluating policies within a single account (p. 1307)** – 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. 1309)** – 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

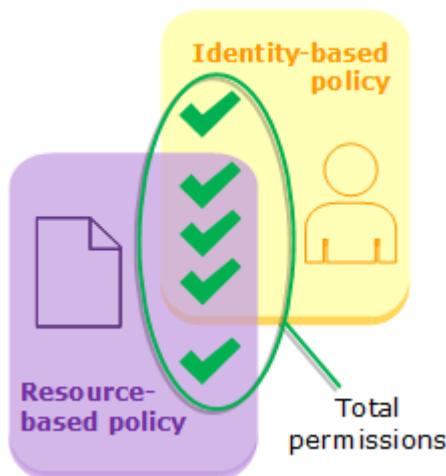
How AWS evaluates policies depends on the types of policies that apply to the request context. The following policy types, listed in order of frequency, are available for use within a single AWS account. For more information about these policy types, see [Policies and permissions in IAM \(p. 485\)](#). To learn how AWS evaluates policies for cross-account access, see [Cross-account policy evaluation logic \(p. 1317\)](#).

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, and session principals such as role sessions and IAM federated users) 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. When resource-based policies are evaluated, the principal ARN that is specified in the policy determines whether implicit denies in other policy types are applicable to the final decision.
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. In some cases, an implicit deny in a permissions boundary can limit the permissions granted by a resource-based policy. To learn more, see [Determining whether a request is allowed or denied within an account \(p. 1309\)](#) later in this topic.
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 the IAM user 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 \(p. 487\)](#).

Remember, an explicit deny in any of these policies overrides the allow.

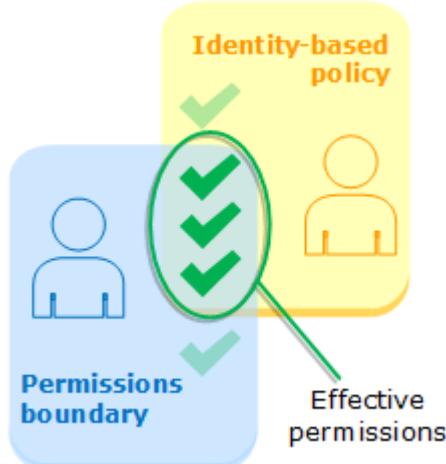
Evaluating identity-based policies with resource-based policies

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 \(p. 502\)](#).



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.



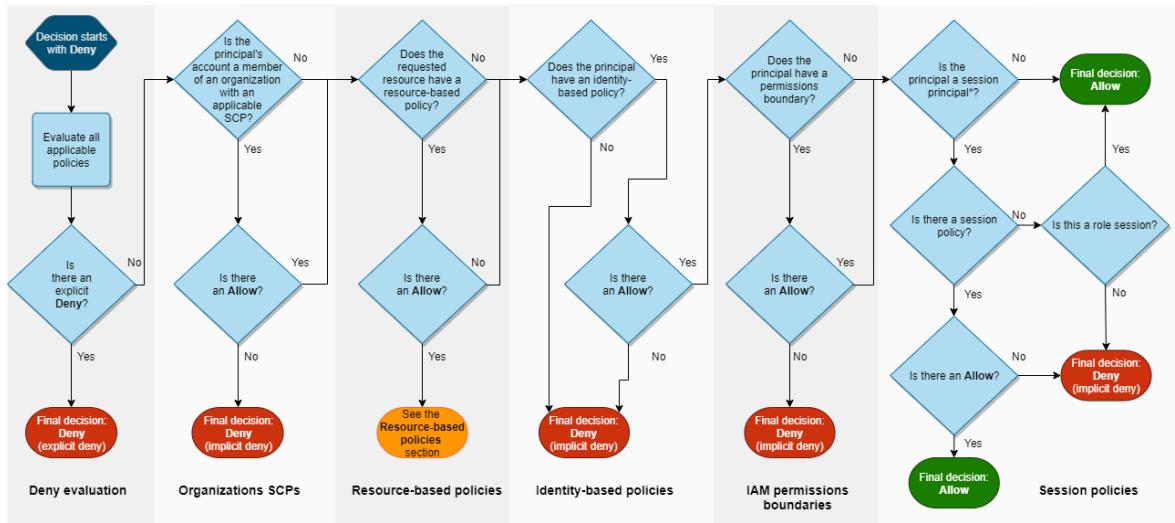
You can learn [whether your account is a member of an organization](#) in AWS Organizations. Organization members might be affected by an SCP. To view this data using the AWS CLI command or AWS API operation, you must have permissions for the `organizations:DescribeOrganization` action for your Organizations 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 evaluates all policies that are applicable to the request context. The following is a summary of the AWS evaluation logic for policies within a single account.

- By default, all requests are implicitly denied with the exception of the AWS account root user, which 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. This flow chart does not cover the impact of resource-based policies and implicit denies in other types of policies.



*A session principal is either a role session or an IAM federated user session.

- 1. Deny evaluation** – By default, all requests are denied. This is called an [implicit deny \(p. 1316\)](#). 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, identity-based policies, IAM permissions boundaries, and session policies. In all those policies, the enforcement code looks for a Deny statement that applies to the request. This is called an [explicit deny \(p. 1316\)](#). If the enforcement code finds even one explicit deny that applies, the code returns a final decision of Deny. If there is no explicit deny, the enforcement code evaluation continues.
- 2. Organizations SCPs** – Then the enforcement 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, the request is explicitly denied, even if the denial is implicit. The enforcement code returns a final decision of Deny. If there is no SCP, or if the SCP allows the requested action, the enforcement code evaluation continues.
- 3. Resource-based policies** – Within the same account, resource-based policies impact policy evaluation differently depending on the type of principal accessing the resource, and the principal that is allowed in the resource-based policy. Depending on the type of principal, an Allow in a resource-based policy can result in a final decision of Allow, even if an implicit deny in an identity-based policy, permissions boundary, or session policy is present.

For most resources, you only need an explicit allow for the principal in either an identity-based policy or a resource-based policy to grant access. [IAM role trust policies \(p. 528\)](#) and [KMS key policies](#) are exceptions to this logic, because they must explicitly allow access for [principals \(p. 1264\)](#).

Resource-based policy logic differs from other policy types if the specified principal is an IAM user, an IAM role, or a session principal. Session principals include IAM [role sessions \(p. 1267\)](#) or an [IAM federated user session \(p. 1269\)](#). If a resource-based policy grants permission directly to the IAM user or the session principal that is making the request, then an implicit deny in an identity-based policy, a permissions boundary, or a session policy does not impact the final decision.

The following table helps you understand the impact of resource-based policies for different principal types when implicit denies are present in identity-based policies, permissions boundaries, and session policies.

Resource-based policies and implicit denies in other policy types (same account)

Principal making the request	Resource-based policy	Identity-based policy	Permissions boundary	Session Policy	Result	Reason
IAM role	Not applicable	Not applicable	Not applicable	Not applicable	Not applicable	A role itself cannot make a request. Requests are made with the role session after a role is assumed.
IAM role session	Allows role ARN	Implicit deny	Implicit deny	Implicit deny	DENY	Permissions boundary and session policy are evaluated as part of the final decision. An implicit deny in either policy results in a DENY decision.
IAM role session	Allows role session ARN	Implicit deny	Implicit deny	Implicit deny	ALLOW	Permissions are granted directly to the session. Other policy types do not affect the decision.
IAM user	Allows IAM user ARN	Implicit deny	Implicit deny	Not applicable	ALLOW	Permissions are granted directly to the user. Other policy types do not affect the decision.

Principal making the request	Resource-based policy	Identity-based policy	Permissions boundary	Session Policy	Result	Reason
IAM federated user (<code>GetFederationToken</code>)	Allows IAM user ARN	Implicit deny	Implicit deny	Implicit deny	DENY	An implicit deny in either the permissions boundary or session policy results in a DENY.
IAM federated user (<code>GetFederationARNToken</code>)	Allows IAM federated user session ARN	Implicit deny	Implicit deny	Implicit deny	ALLOW	Permissions are granted directly to the session. Other policy types do not affect the decision.
root user	Allows root user ARN	Not applicable	Not applicable	Not applicable	ALLOW	The root user has complete, unrestricted access to all resources in your AWS account. To learn how to control access to the root user for accounts in AWS Organizations, see Service control policies (SCPs) in the Organizations User Guide .

Principal making the request	Resource-based policy	Identity-based policy	Permissions boundary	Session Policy	Result	Reason
AWS service principal	Allows an AWS service principal	Not applicable	Not applicable	Not applicable	ALLOW	When a resource-based policy grants permissions directly to an AWS service principal (p. 1269) , other policy types do not affect the decision.

- **IAM role** – Resource-based policies that grant permissions to an IAM role ARN are limited by an implicit deny in a permissions boundary or session policy.

Example role ARN

```
arn:aws:iam::111122223333:role/examplerole
```

- **IAM role session** – Within the same account, resource-based policies that grant permissions to an IAM role session ARN grant permissions directly to the assumed role session. Permissions granted directly to a session are not limited by an implicit deny in an identity-based policy, a permissions boundary, or session policy. When you assume a role and make a request, the principal making the request is the IAM role session ARN and not the ARN of the role itself.

Example role session ARN

```
arn:aws:sts::111122223333:assumed-role/examplerole/examplerolesessionname
```

- **IAM user** – Within the same account, resource-based policies that grant permissions to an IAM user ARN (that is not a federated user session) are not limited by an implicit deny in an identity-based policy or permissions boundary.

Example IAM user ARN

```
arn:aws:iam::111122223333:user/exampleuser
```

- **IAM federated user sessions** – An IAM federated user session is a session created by calling [GetFederationToken \(p. 433\)](#). When a federated user makes a request, the principal making the request is the federated user ARN and not the ARN of the IAM user who federated. Within the same account, resource-based policies that grant permissions to a federated user ARN grant permissions directly to the session. Permissions granted directly to a session are not limited by an implicit deny in an identity-based policy, a permissions boundary, or session policy.

However, if a resource-based policy grants permission to the ARN of the IAM user who federated, then requests made by the federated user during the session are limited by an implicit deny in a permission boundary or session policy.

Example IAM federated user session ARN

```
arn:aws:sts::111122223333:federated-user/exampleuser
```

4. **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 there are no identity-based policies or no statements in identity-based policies that allow the requested action, then the request is implicitly denied and the code returns a final decision of **Deny**. If any statement in any applicable identity-based policies allows the requested action, the code continues.
5. **IAM permissions boundaries** – The 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.
6. **Session policies** – The code then checks whether the principal is a session principal. Session principals include an IAM role session or an IAM federated user session. If the principal is not a session principal, the enforcement code returns a final decision of **Allow**.

For session principals, the code checks whether a session policy was passed in the request. You can pass a session policy while using the AWS CLI or AWS API to get temporary credentials for a role or an IAM federated user.

- If a 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, the code checks whether the principal is a role session. If the principal is a role session, then the request is **Allowed**. Otherwise, the request is implicitly denied and the code returns a final decision of **Deny**.
 - If a session policy is present and allows the requested action, then the enforcement code returns a final decision of **Allow**.
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. When access to a resource is requested, AWS evaluates all the permissions granted by the policies for **at least one Allow** within the same account. An explicit deny in any of the policies overrides the allow.

Important

If either the identity-based policy or the resource-based policy within the same account allows the request and the other doesn't, the request is still allowed.

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:GetBucketLocation",  
                "s3:GetAccountPublicAccessBlock",  
                "s3>ListAccessPoints",  
                "s3>ListAllMyBuckets"  
            ]  
        }  
    ]  
}
```

```

        ],
        "Resource": "arn:aws:s3:::*"
    },
    {
        "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*"
    }
]
}

```

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.

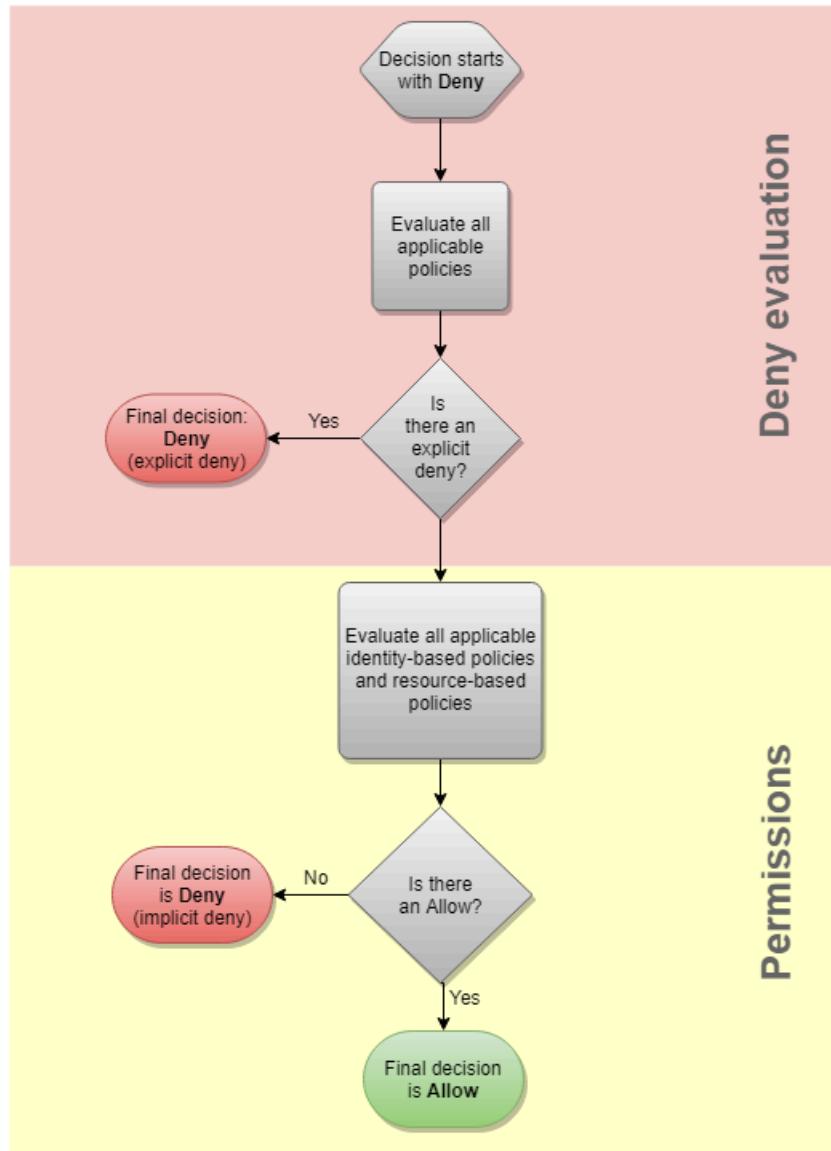
```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:user/carlossalazar"
            },
            "Action": "s3:*",
            "Resource": [
                "arn:aws:s3:::carlossalazar/*",
                "arn:aws:s3:::carlossalazar"
            ]
        }
    ]
}

```

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 principal 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 that includes allowed actions, implicitly denied actions, and explicitly denied actions. The AllowGetList statement **allows** read-only access to IAM actions that begin with the prefixes Get and List. All other actions in IAM, such as iam:CreatePolicy are **implicitly denied**. The DenyReports statement **explicitly denies** access to IAM reports by denying access to actions that include the Report suffix, such as iam:GenerateOrganizationsAccessReport. If someone adds another policy to this principal to grant them access to IAM reports, such as iam:GenerateCredentialReport, report-related requests are still denied because of this explicit deny.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowGetList",
            "Effect": "Allow",
            "Action": [
                "iam:Get*",
                "iam>List*"
            ],
            "Resource": "*"
        },
        {
            "Sid": "DenyReports",
            "Effect": "Deny",
            "Action": "iam:*Report",
            "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 as 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 \(p. 1264\)](#).

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 [Cross account resource access in IAM \(p. 526\)](#).

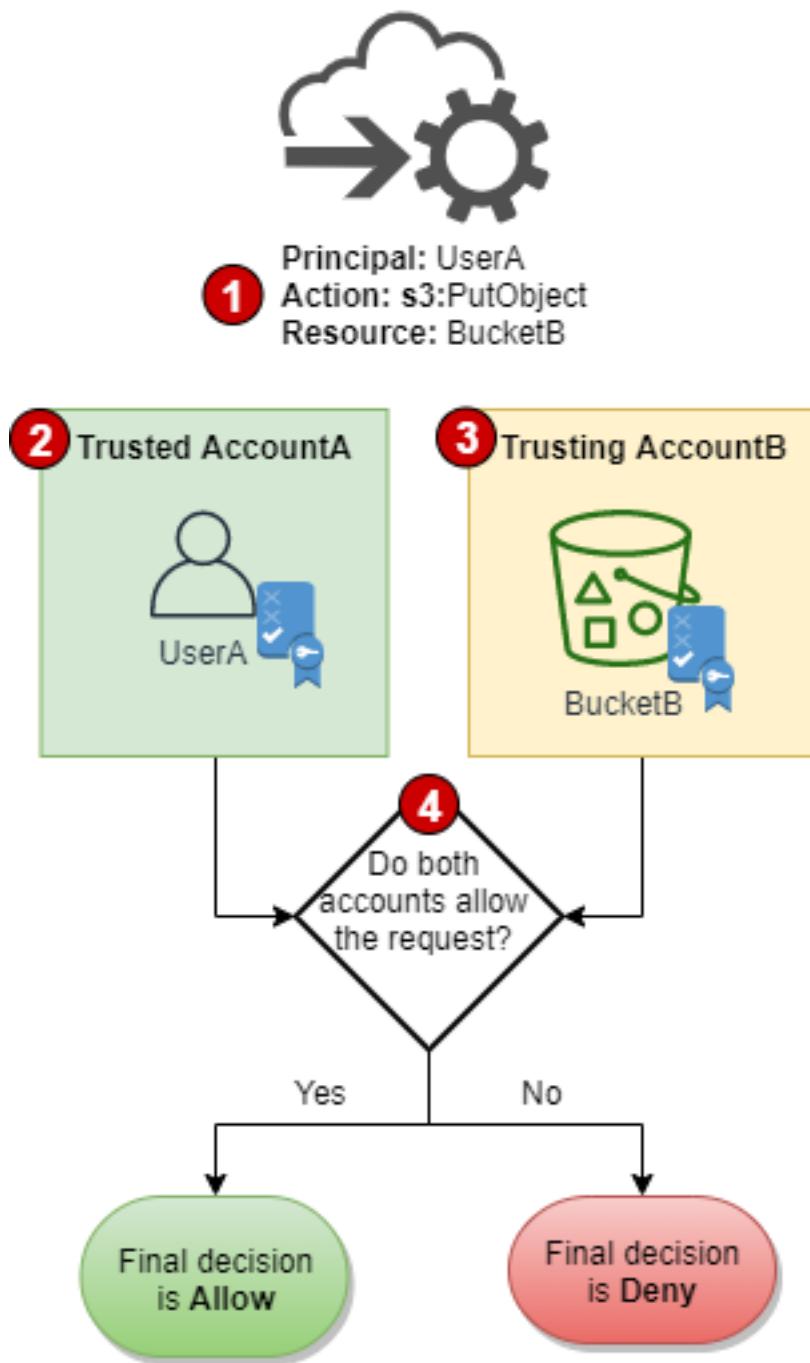
Important

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 \(p. 1309\)](#). 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 \(p. 1307\)](#).
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 \(p. 1307\)](#).

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.

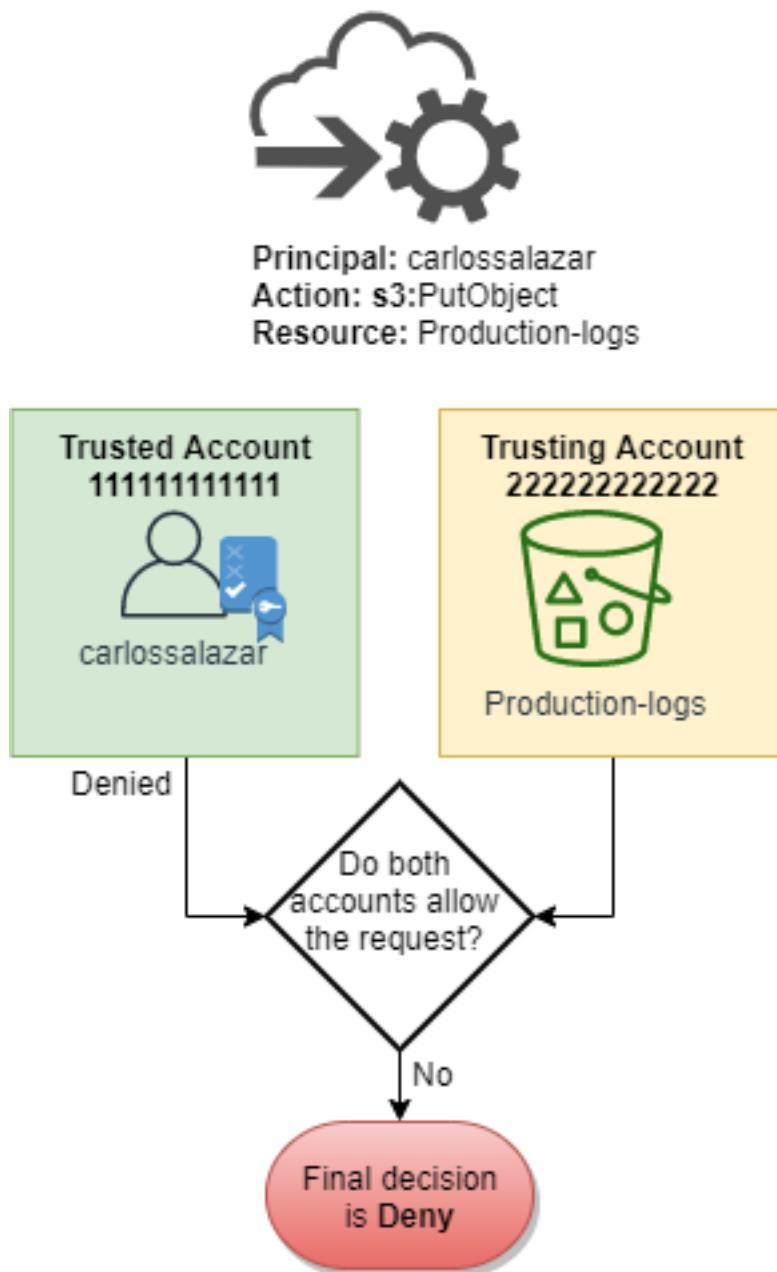
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject*",
                "s3:PutObject*",
                "s3:ReplicateObject",
                "s3:RestoreObject"
            ],
            "Principal": { "AWS": "arn:aws:iam::111111111111:user/carlossalazar" },
            "Resource": "arn:aws:s3:::Production/*"
        }
    ]
}
```

```
    ]  
}
```

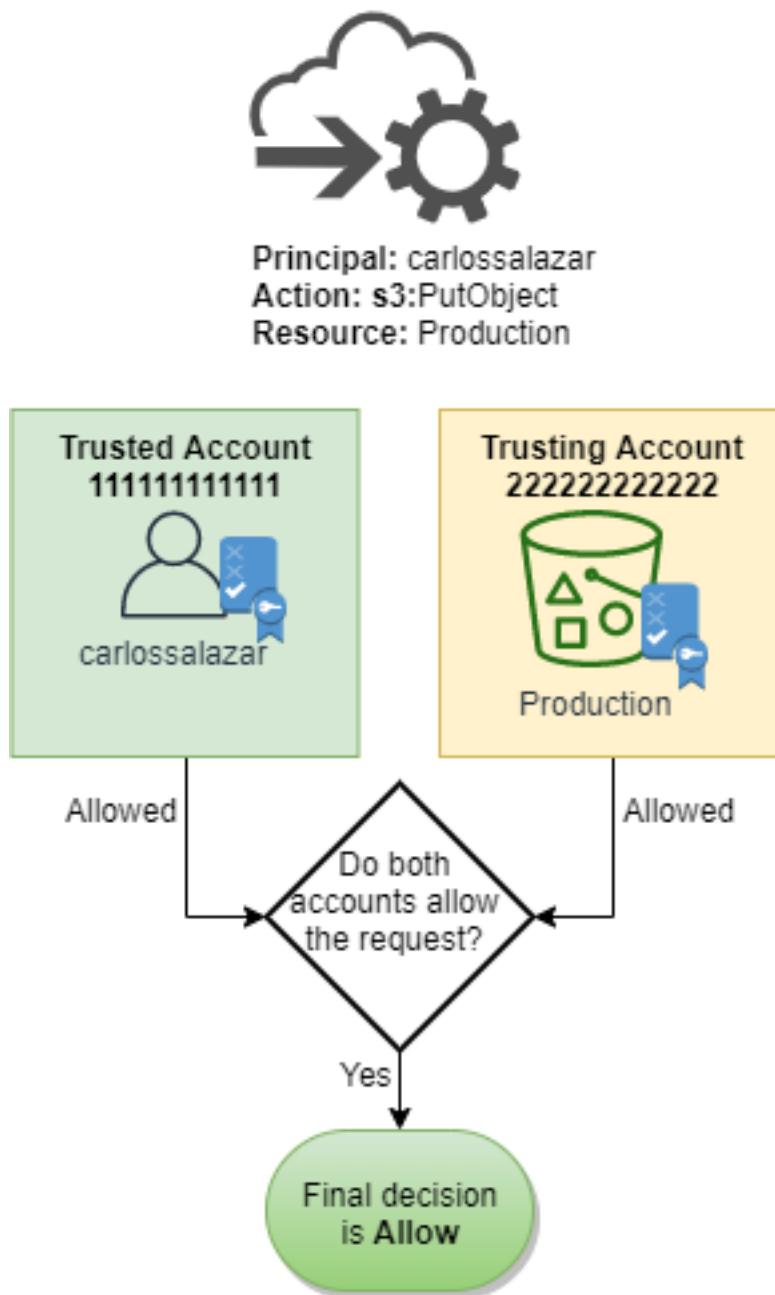
This policy allows the `carlossalazar` user to access objects in the `Production` bucket. He can create and edit, but not delete the objects in the bucket. He can't manage the bucket itself.

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 \(p. 1309\)](#).

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 \(p. 485\)](#)
- [Example IAM identity-based policies \(p. 529\)](#)
- [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 User Guide*.

For examples of policies used in other AWS services, go to the documentation for those services.

Topics

- [The policy language and JSON \(p. 1324\)](#)
- [Conventions used in this grammar \(p. 1324\)](#)
- [Grammar \(p. 1325\)](#)
- [Policy grammar notes \(p. 1326\)](#)

The policy language and JSON

Policies are expressed in JSON. When you create or edit a JSON policy, IAM can perform policy validation to help you create an effective policy. IAM identifies JSON syntax errors, while IAM Access Analyzer provides additional policy checks with recommendations to help you further refine your policies. To learn more about policy validation, see [Validating IAM policies \(p. 588\)](#). To learn more about IAM Access Analyzer policy checks and actionable recommendations, see [IAM Access Analyzer policy validation](#).

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](#).

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 \(p. 1326\)](#) 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 \(p. 1261\)](#). To learn more about policy versions, see [the section called "Versioning IAM policies" \(p. 606\)](#).

```
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>, <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> = (<condition_value_string> | <condition_value_string> |  
    <condition_value_string>)
```

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 AWS STS quotas \(p. 1220\)](#). 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 User 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 \(ARN\) \(p. 1213\)](#) 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 \(p. 1264\)](#).

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 \(p. 1213\)](#) (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 \(p. 1281\)](#).

```
"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 \(p. 1338\)](#). 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 User 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": {  
        "aws:ResourceTag/purpose": "test"  
    }  
}
```

condition_value_string

Identifies the value of the condition_key_string that determines whether the condition is met. For a complete list of valid values for a condition type, see [IAM JSON policy elements: Condition operators \(p. 1281\)](#).

```
"Condition":{  
    "ForAnyValue:StringEquals": {  
        "dynamodb:Attributes": [  
            "ID",  
            "PostDateTime"
```

}

AWS managed policies for job functions

We recommend using policies that [grant least privilege \(p. 1035\)](#), or granting only the permissions required to perform a task. The most secure way to grant least privilege is to write a custom policy with only the permissions needed by your team. You must create a process to allow your team to request more permissions when necessary. It takes time and expertise to [create IAM customer managed policies \(p. 582\)](#) that provide your team with only the permissions they need.

To get started adding permissions to your IAM identities (users, groups of users, and roles), you can use [AWS managed policies \(p. 495\)](#). AWS managed policies cover common use cases and are available in your AWS account. AWS managed policies don't grant least privilege permissions. You must consider the security risk of granting your principals more permissions than they need to do their job.

You can attach AWS managed policies, including job functions, to any IAM identity. To switch to least privilege permissions, you can run AWS Identity and Access Management Access Analyzer to monitor principals with AWS managed policies. After learning which permissions they are using, then you can write a custom policy or generate a policy with only the required permissions for your team. This is less secure, but provides more flexibility as you learn how your team is using AWS.

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 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.

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.

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 job function

AWS managed policy name: [AdministratorAccess](#)

Use case: This user has full access and can delegate permissions to every service and resource in AWS.

Policy updates: AWS maintains and updates this policy. For a history of changes for this policy, view the policy in the IAM console and then choose the **Policy versions** tab. For more information about job function policy updates, see [Updates to AWS managed policies for job functions \(p. 1335\)](#).

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 \(p. 35\)](#).

Billing job function

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 updates: AWS maintains and updates this policy. For a history of changes for this policy, view the policy in the IAM console and then choose the **Policy versions** tab. For more information about job function policy updates, see [Updates to AWS managed policies for job functions \(p. 1335\)](#).

Policy description: This policy grants full permissions for managing billing, costs, payment methods, budgets, and reports. For additional cost management policy examples, see [AWS Billing policy examples](#) in the *AWS Billing and Cost Management User Guide*

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 \(p. 35\)](#).

Database administrator job function

AWS managed policy name: [DatabaseAdministrator](#)

Use case: This user sets up, configures, and maintains databases in the AWS Cloud.

Policy updates: AWS maintains and updates this policy. For a history of changes for this policy, view the policy in the IAM console and then choose the **Policy versions** tab. For more information about job function policy updates, see [Updates to AWS managed policies for job functions \(p. 1335\)](#).

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 \(console\) \(p. 1335\)](#) 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	AWSLambda_FullAccess
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

Use case	Role name (* is a wildcard)	Service role type to select	Select this AWS managed policy
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	The AWS Data Pipeline documentation lists the required permissions for this use case. See IAM roles for AWS Data Pipeline
Allow your applications running on Amazon EC2 instances to access your AWS resources	DataPipelineDefaultRole	Create a role with a trust policy as defined in the AWS Data Pipeline Developer Guide	AmazonEC2RoleforDataPipelineRole

Data scientist job function

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 updates: AWS maintains and updates this policy. For a history of changes for this policy, view the policy in the IAM console and then choose the **Policy versions** tab. For more information about job function policy updates, see [Updates to AWS managed policies for job functions \(p. 1335\)](#).

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 \(console\) \(p. 1335\)](#) 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

Use case	Role name (* is a wildcard)	Service role type to select	AWS managed policy to select
Allow Amazon EMR access to access the Amazon EC2 service and resources for clusters	EMR_DefaultRole	Amazon EMR	AmazonEMRServicePolicy_v2
Allow Kinesis 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	The AWS Data Pipeline documentation lists the required permissions for this use case. See IAM roles for AWS Data Pipeline
Allow your applications running on Amazon EC2 instances to access your AWS resources	DataPipelineDefaultRole	CreateRole Create a role with a trust policy as defined in the AWS Data Pipeline Developer Guide	AmazonEC2RoleforDataPipelineRole

Developer power user job function

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 updates: AWS maintains and updates this policy. For a history of changes for this policy, view the policy in the IAM console and then choose the **Policy versions** tab. For more information about job function policy updates, see [Updates to AWS managed policies for job functions \(p. 1335\)](#).

Policy description: The first statement of this policy uses the [NotAction \(p. 1274\)](#) element to allow all actions for all AWS services and for all resources except AWS Identity and Access Management, AWS Organizations, and AWS Account Management. 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, Organizations, it allows the user to perform all IAM Identity Center actions if IAM Identity Center is enabled. It also grants Account Management permissions to view which AWS Regions are enabled or disabled for the account.

Network administrator job function

AWS managed policy name: [NetworkAdministrator](#)

Use case: This user is tasked with setting up and maintaining AWS network resources.

Policy updates: AWS maintains and updates this policy. For a history of changes for this policy, view the policy in the IAM console and then choose the **Policy versions** tab. For more information about job function policy updates, see [Updates to AWS managed policies for job functions \(p. 1335\)](#).

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 \(console\) \(p. 1335\)](#) 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 .

Read-only access

AWS managed policy name: [ReadOnlyAccess](#)

Use case: This user requires read-only access to every resource in an AWS account.

Policy updates: AWS maintains and updates this policy. For a history of changes for this policy, view the policy in the IAM console and then choose the **Policy versions** tab. For more information about job function policy updates, see [Updates to AWS managed policies for job functions \(p. 1335\)](#).

Policy description: This policy grants permissions to list, get, describe, and otherwise view resources and their attributes. It does not include mutating functions like create or delete. This policy does include read-only access to security-related AWS services, such as AWS Identity and Access Management and AWS Billing and Cost Management. View the policy for the full list of services and actions that this policy supports.

Security auditor job function

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 updates: AWS maintains and updates this policy. For a history of changes for this policy, view the policy in the IAM console and then choose the **Policy versions** tab. For more information about job function policy updates, see [Updates to AWS managed policies for job functions \(p. 1335\)](#).

Policy description: This policy grants permissions to view configuration data for many AWS services and to review their logs.

Support user job function

AWS managed policy name: [SupportUser](#)

Use case: This user contacts AWS Support, creates support cases, and views the status of existing cases.

Policy updates: AWS maintains and updates this policy. For a history of changes for this policy, view the policy in the IAM console and then choose the **Policy versions** tab. For more information about job function policy updates, see [Updates to AWS managed policies for job functions \(p. 1335\)](#).

Policy description: This policy grants permissions to create and update AWS Support cases.

System administrator job function

AWS managed policy name: [SystemAdministrator](#)

Use case: This user sets up and maintains resources for development operations.

Policy updates: AWS maintains and updates this policy. For a history of changes for this policy, view the policy in the IAM console and then choose the **Policy versions** tab. For more information about job function policy updates, see [Updates to AWS managed policies for job functions \(p. 1335\)](#).

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 \(console\) \(p. 1335\)](#) 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 job function

AWS managed policy name: [ViewOnlyAccess](#)

Use case: This user can view a list of AWS resources and basic metadata in the account across services. The user cannot read resource content or metadata that goes beyond the quota and list information for resources.

Policy updates: AWS maintains and updates this policy. For a history of changes for this policy, view the policy in the IAM console and then choose the **Policy versions** tab. For more information about job function policy updates, see [Updates to AWS managed policies for job functions \(p. 1335\)](#).

Policy description: This policy grants List*, Describe*, Get*, View*, and Lookup* access to resources for AWS services. To see what actions this policy includes for each service, see [ViewOnlyAccess](#).

Updates to AWS managed policies for job functions

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 services. 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.

For a job function policy, you can view the version history and the time and date of each update in the IAM console. To do this, use the links on this page to view the policy details. Then choose the **Policy versions** tab to view the versions. This page shows the last 25 versions of a policy. To view all of the versions for a policy, call the [get-policy-version](#) AWS CLI command or the [GetPolicyVersion](#) API operation.

Note

You can have up to five versions of a customer managed policy, but AWS retains the full version history of AWS managed policies.

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.
4. Choose the use case for your service. Use cases are defined by the service to include the trust policy that the service requires.
5. Choose **Next**.
6. If possible, select the policy to use for the permissions policy. Otherwise, choose **Create policy** to open a new browser tab and create a new policy from scratch. For more information, see [Creating IAM policies](#) in the *IAM User Guide*.
7. 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 let you do any of the following:

- Nothing, because the service defines the permissions for the role.
- Choose from a limited set of permissions.
- Choose from any permissions.
- Select no policies at this time. However, you can create the policies later, and then attach them to the role.

8. (Optional) Set a [permissions boundary](#). This is an advanced feature that is available for service roles, but not for service-linked roles.

Expand the **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 [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.

9. Choose **Next**.
10. For **Role name**, the degree of role name customization is defined by the service. If the service defines the role's name, you can't edit this option. In other cases, the service might define a prefix for the role and you can enter an optional suffix. For some services, you can 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, so don't create roles named both **PRODROLE** and **prodrole**. When a role name is used in a policy or as part of an ARN, the role name is case sensitive. When a role name appears to customers in the console, such as during the sign-in process, the role name is case insensitive. Because various entities might reference the role, you can't edit the name of the role after it is created.

11. (Optional) For **Description**, enter a description for the new role.
12. Choose **Edit** in the **Step 1: Select trusted entities** or **Step 2: Select permissions** sections to edit the use cases and permissions for the role.
13. (Optional) Add metadata to the role by attaching tags as key-value pairs. For more information about using tags in IAM, see [Tagging IAM resources](#) in the *IAM User Guide*.
14. 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 \(p. 1330\)](#). 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 they can manage the Amazon database services. That policy also allows Alice to pass a role called `rds-monitoring-role` to the Amazon RDS service that allows the service to monitor the Amazon RDS databases on their behalf.

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. Choose **Policies**, type **database** in the search box, and then press enter.
3. Select the radio button for the **DatabaseAdministrator** policy, choose **Actions**, and then choose **Attach**.
4. In the list of entities, 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`. Enter that for the **Role name**.
10. (Optional) For **Role description**, enter 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, they might do this when they create a DB instance, create a read replica, or modify a DB instance. They must enter the role name they created (`rds-monitoring-role`) in the **Monitoring Role** box when they set **Enable Enhanced Monitoring** to **Yes**.

Example 2: Configuring a user as a network administrator (console)

This example shows the steps required to configure Jorge, an IAM user, as a [Network Administrator \(p. 1332\)](#). It uses the information in the table in that section to allow Jorge to monitor IP traffic going to and from a VPC. It also allows Jorge to capture that information in the logs in CloudWatch Logs. You attach the [NetworkAdministrator](#) policy to Jorge's IAM user so that they can configure AWS network resources. That policy also allows Jorge to pass a role whose name begins with `flow-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 enter **network** in the search box, and then press enter.
3. Select the radio button next to **NetworkAdministrator** policy, choose **Actions**, and then choose **Attach**.
4. In the list of users, select the check box next to **Jorge** and then choose **Attach policy**. Jorge can now 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. In the **Policy editor** section, choose the **JSON** option 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": "*"  
        }  
    ]  
}
```

7. Resolve any security warnings, errors, or general warnings generated during [policy validation \(p. 588\)](#), and then choose **Next**.

Note

You can switch between the **Visual** and **JSON** editor options any time. However, if you make changes or choose **Next** in the **Visual** editor, IAM might restructure your policy to optimize it for the visual editor. For more information, see [Policy restructuring \(p. 1181\)](#).

8. On the **Review and create** page, type `vpc-flow-logs-policy-for-service-role` for the policy name. Review the **Permissions defined in this policy** 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.

9. In the navigation pane of the IAM console, choose **Roles**, and then choose **Create role**.
10. Choose the **AWS Service** role type, and then choose **Amazon EC2**.
11. Choose the **Amazon EC2** use case.
12. On the **Attach permissions policies** page, choose the policy you created earlier, **vpc-flow-logs-policy-for-service-role**, and then choose **Next: Review**.
13. The role name must be permitted by the NetworkAdministrator policy that Jorge now has. Any name that begins with flow-logs- is allowed. For this example, enter **flow-logs-for-jorge** for the **Role name**.
14. (Optional) For **Role description**, enter a description for the new role.
15. After you review the details, choose **Create role**.
16. Now you can configure the trust policy required for this scenario. On the **Roles** page, choose the **flow-logs-for-jorge** 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**.
17. Change the "Service" line to read as follows, replacing the entry for ec2.amazonaws.com:

```
"Service": "vpc-flow-logs.amazonaws.com"
```

18. Jorge 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-jorge** role. That role has the permissions to create the log and write data to it.

AWS global condition context keys

When a [principal \(p. 7\)](#) makes a [request \(p. 8\)](#) to AWS, AWS gathers the request information into a [request context \(p. 8\)](#). 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 \(p. 1278\)](#).

Note

If you use condition keys that are available only in some circumstances, you can use the [IfExists \(p. 1288\)](#) 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.

Important

To compare your condition against a request context with multiple key values, you must use the ForAllValues or ForAnyValue set operators. Use set operators only with multivalued

condition keys. Do not use set operators with single-valued condition keys. For more information, see [Multivalued context keys \(p. 1294\)](#).

aws:CalledVia

Works with [string operators \(p. 1282\)](#).

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.
- **Value type** – Multivalued

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 service principals with aws:CalledVia.

Service principal
athena.amazonaws.com
backup.amazonaws.com
cloud9.amazonaws.com
cloudformation.amazonaws.com
databrew.amazonaws.com
dataexchange.amazonaws.com
dynamodb.amazonaws.com
imagebuilder.amazonaws.com
kms.amazonaws.com
mgn.amazonaws.com
nimble.amazonaws.com
omics.amazonaws.com
ram.amazonaws.com
servicecatalog-appregistry.amazonaws.com
sqlworkbench.amazonaws.com

To allow or deny access when *any* service makes a request using the principal's credentials, use the [aws:ViaAWSService \(p. 1366\)](#) condition key. That condition key supports AWS services.

The `aws:CalledVia` key is a [multivalued key \(p. 1293\)](#). 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 \(p. 1294\)](#) 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 \(p. 1341\)](#) and [aws:CalledViaLast \(p. 1341\)](#) 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 \(p. 1341\)](#) and [aws:CalledViaLast \(p. 1341\)](#) 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 \(p. 1282\)](#).

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 \(p. 1339\)](#).

- **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.
- **Value type** – Single-valued

aws:CalledViaLast

Works with [string operators \(p. 1282\)](#).

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 \(p. 1339\)](#).

- **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.
- **Value type** – Single-valued

aws:CurrentTime

Works with [date operators \(p. 1285\)](#).

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 \(p. 532\)](#).

- **Availability** – This key is always included in the request context.
- **Value type** – Single-valued

aws:Ec2InstanceStateVpc

Works with [string operators \(p. 1282\)](#).

This key identifies the VPC to which Amazon EC2 IAM role credentials were delivered to. You can use this key in a policy with the [aws:SourceVPC \(p. 1364\)](#) global key to check if a call is made from a VPC (aws:SourceVPC) that matches the VPC where a credential was delivered to (aws:Ec2InstanceSourceVpc).

- **Availability** – This key is included in the request context whenever the requester is signing requests with an Amazon EC2 role credential. It can be used in IAM policies, service control policies, VPC endpoint policies, and resource policies.
- **Value type** – Single-valued

This key can be used with VPC identifier values, but is most useful when used as a variable combined with the aws:SourceVpc context key. The aws:SourceVpc context key is included in the request context only if the requester uses a VPC endpoint to make the request. Using aws:Ec2InstanceSourceVpc with aws:SourceVpc allows you to use aws:Ec2InstanceSourceVpc more broadly since it compares values that typically change together.

Note

This condition key is not available in EC2-Classic.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "RequireSameVPC",
      "Effect": "Deny",
      "Action": "*",
      "Resource": "*",
      "Condition": {
        "StringNotEquals": {
          "aws:SourceVpc": "${aws:Ec2InstanceSourceVpc}"
        },
        "Null": {
          "ec2:SourceInstanceARN": "false"
        },
        "BoolIfExists": {
          "aws:ViaAWSService": "false"
        }
      }
    }
  ]
}
```

In the example above, access is denied if the aws:SourceVpc value isn't equal to the aws:Ec2InstanceSourceVpc value. The policy statement is limited to only roles used as Amazon EC2 instance roles by testing for the existence of the ec2:SourceInstanceARN condition key.

The policy uses aws:ViaAWSService to allow AWS to authorize requests when requests are made on behalf of your Amazon EC2 instance roles. For example, when you make a request from an Amazon EC2 instance to an encrypted Amazon S3 bucket, Amazon S3 makes a call to AWS KMS on your behalf. Some of the keys are not present when the request is made to AWS KMS.

aws:Ec2InstanceSourcePrivateIPv4

Works with [IP address operators \(p. 1286\)](#).

This key identifies the private IPv4 address of the primary elastic network interface to which Amazon EC2 IAM role credentials were delivered. You must use this condition key with its companion key aws:Ec2InstanceSourceVpc to ensure that you have a globally unique combination of VPC ID and source private IP. Use this key with aws:Ec2InstanceSourceVpc to ensure that a request was made from the same private IP address that the credentials were delivered to.

- **Availability** – This key is included in the request context whenever the requester is signing requests with an Amazon EC2 role credential. It can be used in IAM policies, service control policies, VPC endpoint policies, and resource policies.
- **Value type** – Single-valued

Important

This key should not be used alone in an Allow statement. Private IP addresses are by definition not globally unique. You should use the `aws:Ec2InstanceSourceVpc` key every time you use the `aws:Ec2InstanceSourcePrivateIPv4` key to specify the VPC your Amazon EC2 instance credentials can be used from.

Note

This condition key is not available in EC2-Classic.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": "*",
            "Resource": "*",
            "Condition": {
                "StringNotEquals": {
                    "aws:Ec2InstanceSourceVpc": "${aws:SourceVpc}"
                },
                "Null": {
                    "ec2:SourceInstanceARN": "false"
                },
                "BoolIfExists": {
                    "aws:ViaAWSService": "false"
                }
            }
        },
        {
            "Effect": "Deny",
            "Action": "*",
            "Resource": "*",
            "Condition": {
                "StringNotEquals": {
                    "aws:Ec2InstanceSourcePrivateIPv4": "${aws:VpcSourceIp}"
                },
                "Null": {
                    "ec2:SourceInstanceARN": "false"
                },
                "BoolIfExists": {
                    "aws:ViaAWSService": "false"
                }
            }
        }
    ]
}
```

aws:EpochTime

Works with [date operators \(p. 1285\)](#) or [numeric operators \(p. 1284\)](#).

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.
- **Value type** – Single-valued

aws:FederatedProvider

Works with [string operators \(p. 1282\)](#).

Use this key to compare the principal's issuing identity provider (IdP) with the IdP that you specify in the policy. This means that an IAM role was assumed using the AssumeRoleWithWebIdentity or AssumeRoleWithSAML AWS STS operations. When the resulting role session's temporary credentials are used to make a request, the request context identifies the IdP that authenticated the original federated identity.

- **Availability** – This key is present when the principal is a role session principal and that session was issued using a third-party identity provider.
- **Value type** – Single-valued

For example, if the user was authenticated through Amazon Cognito, the request context includes the value cognito-identity.amazonaws.com. Similarly, if the user was authenticated through Login with Amazon, the request context includes the value www.amazon.com.

You can use any single-valued condition key as a [variable \(p. 1298\)](#). The following example resource-based policy uses the aws:FederatedProvider key as a policy variable in the ARN of a resource. This policy allows any principal who authenticated using an IdP to get objects out of an Amazon S3 bucket with a path that's specific to the issuing identity provider.

```
{  
    "Version": "2012-10-17",  
    "Statement": {  
        "Effect": "Allow",  
        "Principal": "*",
        "Action": "s3:GetObject",
        "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/${aws:FederatedProvider}/*"  
    }  
}
```

aws:MultiFactorAuthAge

Works with [numeric operators \(p. 1284\)](#).

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 \(p. 114\)](#).

- **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.
- **Value type** – Single-valued

aws:MultiFactorAuthPresent

Works with [Boolean operators \(p. 1285\)](#).

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.
- **Value type** – Single-valued

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 \(p. 1224\)](#).

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 \(p. 1288\)](#) 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 \(p. 1288\)](#) 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 \(p. 1288\)](#) 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 \(p. 1282\)](#).

Use this key to compare the account to which the requesting principal belongs with the account identifier that you specify in the policy. For anonymous requests, the request context returns anonymous.

- **Availability** – This key is included in the request context for all requests, including anonymous requests.
- **Value type** – Single-valued

In the following example, access is denied except to principals with the account number 123456789012.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyAccessFromPrincipalNotInSpecificAccount",
      "Action": "service:*",
```

```
    "Effect": "Deny",
    "Resource": [
        "arn:aws:service:region:accountID:resource"
    ],
    "Condition": {
        "StringNotEquals": {
            "aws:PrincipalArn": [
                "123456789012"
            ]
        }
    }
}
```

aws:PrincipalArn

Works with [ARN operators \(p. 1288\)](#) and [string operators \(p. 1282\)](#). AWS recommends that you use ARN operators instead of string operators when comparing ARNs.

Use this key to compare the [Amazon Resource Name \(p. 1213\)](#) (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.

- **Availability** – This key is included in the request context for all signed requests. Anonymous requests do not include this key. You can specify the following types of principals in this condition key:
 - IAM role
 - IAM user
 - AWS STS federated user session
 - AWS account root user
- **Value type** – Single-valued

The following list shows the request context value returned for different types of principals that you can specify in the aws:PrincipalArn condition key:

- **IAM role** – The request context contains the following value for condition key aws:PrincipalArn. Do not specify the assumed role session ARN as a value for this condition key. For more information about the assumed role session principal, see [Role session principals \(p. 1267\)](#).

```
arn:aws:iam::123456789012:role/role-name
```

- **IAM user** – The request context contains the following value for condition key aws:PrincipalArn.

```
arn:aws:iam::123456789012:user/user-name
```

- **AWS STS federated user sessions** – The request context contains the following value for condition key aws:PrincipalArn.

```
arn:aws:sts::123456789012:federated-user/user-name
```

- **AWS account root user** – The request context contains the following value for condition key aws:PrincipalArn. When you specify the root user ARN as the value for the aws:PrincipalArn condition key, it limits permissions only for the root user of the AWS account. This is different from specifying the root user ARN in the principal element of a resource-based policy, which delegates authority to the AWS account. For more information about specifying the root user ARN in the principal element of a resource-based policy, see [AWS account principals \(p. 1265\)](#).

```
arn:aws:iam::123456789012:root
```

You can specify the root user ARN as a value for condition key `aws:PrincipalArn` in AWS Organizations service control policies (SCPs). SCPs are a type of organization policy used to manage permissions in your organization and affect only member accounts in the organization. An SCP restricts permissions for IAM users and roles in member accounts, including the member account's root user. For more information about the effect of SCPs on permissions, see [SCP effects on permissions](#) in the *Organizations User Guide*.

aws:PrincipalIsAWSservice

Works with [Boolean operators \(p. 1285\)](#).

Use this key to check whether the call to your resource is being made directly by an AWS [service principal \(p. 1269\)](#). For example, AWS CloudTrail uses the service principal `cloudtrail.amazonaws.com` to write logs to your Amazon S3 bucket. The request context key is set to true when a service uses a service principal to perform a direct action on your resources. The context key is set to false if the service uses the credentials of an IAM principal to make a request on the principal's behalf. It is also set to false if the service uses a [service role](#) or [service-linked role](#) to make a call on the principal's behalf.

- **Availability** – This key is present in the request context for all signed API requests that use AWS credentials. Anonymous requests do not include this key.
- **Value type** – Single-valued

You can use this condition key to limit access to your trusted identities and expected network locations while safely granting access to AWS services.

In the following Amazon S3 bucket policy example, access to the bucket is restricted unless the request originates from `vpc-111bbb22` or is from a service principal, such as CloudTrail.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Expected-network+service-principal",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET1/AWSLogs/AccountNumber/*",
            "Condition": {
                "StringNotEqualsIfExists": {
                    "aws:SourceVpc": "vpc-111bbb22"
                },
                "BoolIfExists": {
                    "aws:PrincipalIsAWSService": "false"
                }
            }
        }
    ]
}
```

In the following video, learn more about how you might use the `aws:PrincipalIsAWSservice` condition key in a policy.

[Safely grant access to your authorized users, expected network locations, and AWS services together.](#)

aws:PrincipalOrgID

Works with [string operators \(p. 1282\)](#).

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. Anonymous requests do not include this key.
- **Value type** – Single-valued

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 \(p. 511\)](#). 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-xxxxxxxxxx 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-xxxxxxxxxx"}  
         }  
    ]  
}
```

Note

This global condition also applies to the management account of an AWS organization. This policy prevents all principals outside of the specified organization from accessing the Amazon S3 bucket. This includes any AWS services that interact with your internal resources, such as AWS CloudTrail sending log data to your Amazon S3 buckets. To learn how you can safely grant access for AWS services, see [aws:PrincipalAWSService \(p. 1348\)](#).

For more information about AWS Organizations, see [What Is AWS Organizations?](#) in the [AWS Organizations User Guide](#).

aws:PrincipalOrgPaths

Works with [string operators \(p. 1282\)](#).

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 \(p. 625\)](#).

- **Availability** – This key is included in the request context only if the principal is a member of an organization. Anonymous requests do not include this key.
- **Value type** – Multivalued

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-ab12-22222222 OU, but not in its child OUs.

```
"Condition" : { "ForAnyValue:StringEquals" : {
    "aws:PrincipalOrgPaths": ["o-a1b2c3d4e5/r-ab12/ou-ab12-11111111/ou-ab12-22222222/"]}}
```

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-ab12/ou-ab12-11111111/ou-ab12-22222222/*"]}}
```

The following condition returns true for principals in an account that is attached directly to any of the child OUs, but not directly to the parent OU. The previous condition is for the OU or any children. The following condition is for only the children (and any children of those children).

```
"Condition" : { "ForAnyValue:StringLike" : {
    "aws:PrincipalOrgPaths": ["o-a1b2c3d4e5/r-ab12/ou-ab12-11111111/ou-ab12-22222222/ou-*"]}}
```

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 can have multiple values in the request context. 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 more information about multivalued condition keys, see [Multivalued context keys \(p. 1294\)](#).

```
"Condition": {
    "ForAnyValue:StringLike": {
        "aws:PrincipalOrgPaths": [
            "o-a1b2c3d4e5/r-ab12/ou-ab12-33333333/*",
            "o-a1b2c3d4e5/r-ab12/ou-ab12-22222222/*"
        ]
    }
}
```

aws:PrincipalServiceName

Works with [string operators \(p. 1282\)](#).

Use this key to compare the [service principal \(p. 1269\)](#) name in the policy with the service principal that is making requests to your resources. You can use this key to check whether this call is made

by a specific service principal. When a service principal makes a direct request to your resource, the `aws:PrincipalServiceName` key contains the name of the service principal. For example, the AWS CloudTrail service principal name is `cloudtrail.amazonaws.com`.

- **Availability** – This key is present in the request when the call is made by an AWS service principal. This key is not present in any other situation, including the following:
 - If the service uses a [service role](#) or [service-linked role](#) to make a call on the principal's behalf.
 - If the service uses the credentials of an IAM principal to make a request on the principal's behalf.
 - If the call is made directly by an IAM principal.
 - If the call is made by an anonymous requester.
- **Value type** – Single-valued

You can use this condition key to limit access to your trusted identities and expected network locations while safely granting access to an AWS service.

In the following Amazon S3 bucket policy example, access to the bucket is restricted unless the request originates from `vpc-111bbb22` or is from a service principal, such as CloudTrail.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "expected-network+service-principal",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "s3:PutObject",
      "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET1/AWSLogs/AccountNumber/*",
      "Condition": {
        "StringNotEqualsIfExists": {
          "aws:SourceVpc": "vpc-111bbb22",
          "aws:PrincipalServiceName": "cloudtrail.amazonaws.com"
        }
      }
    }
  ]
}
```

aws:PrincipalServiceNamesList

Works with [string operators \(p. 1282\)](#).

This key provides a list of all [service principal \(p. 1269\)](#) names that belong to the service. This is an advanced condition key. You can use it to restrict the service from accessing your resource from a specific Region only. Some services may create Regional service principals to indicate a particular instance of the service within a specific Region. You can limit access to a resource to a particular instance of the service. When a service principal makes a direct request to your resource, the `aws:PrincipalServiceNamesList` contains an unordered list of all service principal names associated with the Regional instance of the service.

- **Availability** – This key is present in the request when the call is made by an AWS service principal. This key is not present in any other situation, including the following:
 - If the service uses a [service role](#) or [service-linked role](#) to make a call on the principal's behalf.
 - If the service uses the credentials of an IAM principal to make a request on the principal's behalf.
 - If the call is made directly by an IAM principal.
 - If the call is made by an anonymous requester.

- **Value type** – Multivalued

`aws:PrincipalServiceNamesList` is a multivalued condition key. Multivalued keys can have multiple values in the request context. You must use the `ForAnyValue` or `ForAllValues` set operators with [string condition operators \(p. 1282\)](#) for this key. For more information about multivalued condition keys, see [Multivalued context keys \(p. 1294\)](#).

aws:PrincipalTag/tag-key

Works with [string operators \(p. 1282\)](#).

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 \(p. 417\)](#). Anonymous requests do not include this key.
- **Value type** – Single-valued

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 resources \(p. 399\)](#). You can use `aws:PrincipalTag` to [control access \(p. 522\)](#) for AWS principals.

This example shows how you might create an identity-based policy that allows users with the `department=hr` 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 \(p. 581\)](#) or [edit a policy \(p. 609\)](#).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iam:*",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalTag/department

```

This example shows how you might create a resource-based policy with the `aws:PrincipalTag` key in the resource ARN. The policy allows the `s3:GetObject` action only if the bucket name ends with a team name from the `team` principal 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 \(p. 581\)](#) or [edit a policy \(p. 609\)](#).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET-${aws:PrincipalTag/team}/*"
    }
  ]
}
```

```
}
```

aws:PrincipalType

Works with [string operators \(p. 1282\)](#).

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 \(p. 1264\)](#). For specific examples of principal key values, see [Principal key values \(p. 1303\)](#).

- **Availability** – This key is included in the request context for all requests, including anonymous requests.
- **Value type** – Single-valued

aws:referer

Works with [string operators \(p. 1282\)](#).

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.
- **Value type** – Single-valued

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 \(p. 1282\)](#).

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.

- **Value type** – Single-valued

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 \(p. 1274\)](#).

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 extend across regions.

- 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 invoke s3:CreateBucket to create a bucket in another region, and use the s3:LocationConstraint condition key to control the applicable regions.

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 \(p. 1282\)](#).

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 \(p. 524\)](#).

- **Availability** – This key is included in the request context when tag key-value pairs are passed in the request. When multiple tags are passed in the request, there is one context key for each tag key-value pair.
- **Value type** – Single-valued

This context key is formatted "aws:RequestTag/*tag-key*": "*tag-value*" where *tag-key* and *tag-value* are a tag key and value pair. Tag keys and values are not case-sensitive. This means that if you specify "aws:RequestTag/TagKey1": "Value1" in the condition element of your policy, then the condition matches a request tag key named either TagKey1 or tagkey1, but not both.

This example shows that while the key is single-valued, you can still use multiple key-value pairs in a request if the keys are different.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "ec2:CreateTags",
      "Resource": "arn:aws:ec2:::instance/*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/environment": [
            "preprod",
            "production"
          ],
          "aws:RequestTag/team": [
            "engineering"
          ]
        }
      }
    }
  ]
}
```

aws:ResourceAccount

Works with [string operators \(p. 1282\)](#).

Use this key to compare the requested resource owner's [AWS account ID](#) with the resource account in the policy. You can then allow or deny access to that resource based on the account that owns the resource.

- **Availability** – This key is always included in the request context for most service actions. The following actions don't support this key:
 - Amazon Elastic Block Store – All actions
 - Amazon EC2

- ec2:AcceptTransitGatewayPeeringAttachment
- ec2:AcceptVpcEndpointConnections
- ec2:AcceptVpcPeeringConnection
- ec2:CopySnapshot
- ec2>CreateVolume
- ec2>CreateVpcEndpoint
- ec2>CreateVpcPeeringConnection
- ec2>DeleteTransitGatewayPeeringAttachment
- ec2>DeleteVpcPeeringConnection
- ec2:RejectTransitGatewayPeeringAttachment
- ec2:RejectVpcEndpointConnections
- ec2:RejectVpcPeeringConnection
- Amazon EventBridge
 - events:PutEvents – EventBridge PutEvents calls on an event bus in another account, if that event bus was configured as a cross-account EventBridge target before March 2, 2023. For more information, see [Grant permissions to allow events from other AWS accounts](#) in the *Amazon EventBridge User Guide*.
- Amazon Route 53
 - route53:AssociateVpcWithHostedZone
 - route53>CreateVPCAssociationAuthorization
 - route53>DeleteVPCAssociationAuthorization
 - route53:DisassociateVPCFromHostedZone
 - route53>ListHostedZonesByVPC
- Amazon WorkSpaces
 - workspaces:DescribeWorkspaceImages
- **Value type** – Single-valued

Note

For additional considerations for the above unsupported actions, see the [Data Perimeter Policy Examples](#) repository.

This key is equal to the AWS account ID for the account with the resources evaluated in the request.

For most resources in your account, the [ARN \(p. 1288\)](#) contains the owner account ID for that resource. For certain resources, such as Amazon S3 buckets, the resource ARN does not include the account ID. The following two examples show the difference between a resource with an account ID in the ARN, and an Amazon S3 ARN without an account ID:

- arn:aws:iam::123456789012:role/AWSExampleRole – IAM role created and owned within the account 123456789012.
- arn:aws:s3:::**DOC-EXAMPLE-BUCKET2** – Amazon S3 bucket created and owned within the account 111122223333, not displayed in the ARN.

Use the AWS console, or API, or CLI, to find all of your resources and corresponding ARNs.

You write a policy that denies permissions to resources based on the resource owner's account ID. For example, the following identity-based policy denies access to the *specified resource* if the resource does not belong to the *specified account*.

To use this policy, replace the *italicized placeholder text* with your account information.

Important

This policy does not allow any actions. Instead, it uses the Deny effect which explicitly denies access to all of the resources listed in the statement that do not belong to the listed account. Use this policy in combination with other policies that allow access to specific resources.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "DenyInteractionWithResourcesNotInSpecificAccount",  
            "Action": "service:*",  
            "Effect": "Deny",  
            "Resource": [  
                "arn:aws:service:region:account:*"  
            ],  
            "Condition": {  
                "StringNotEquals": {  
                    "aws:ResourceAccount": [  
                        "account"  
                    ]  
                }  
            }  
        }  
    ]  
}
```

This policy denies access to all resources for a specific AWS service unless the specified AWS account owns the resource.

Note

Some AWS services require access to AWS owned resources that are hosted in another AWS account. Using `aws:ResourceAccount` in your identity-based policies might impact your identity's ability to access these resources.

Certain AWS services, such as AWS Data Exchange, rely on access to resources outside of your AWS accounts for normal operations. If you use the element `aws:ResourceAccount` in your policies, include additional statements to create exemptions for those services. The example policy [AWS: Deny access to Amazon S3 resources outside your account except AWS Data Exchange \(p. 545\)](#) demonstrates how to deny access based on the resource account while defining exceptions for service-owned resources.

Use this policy example as a template for creating your own custom policies. Refer to your service [documentation](#) for more information.

aws:ResourceOrgID

Works with [string operators \(p. 1282\)](#).

Use this key to compare the identifier of the organization in AWS Organizations to which the requested resource belongs with the identifier specified in the policy.

- **Availability** – This key is included in the request context only if the account that owns the resource is a member of an organization. This global condition key does not support the following actions:
 - Amazon Elastic Block Store – All actions
 - Amazon EC2
 - `ec2:AcceptTransitGatewayPeeringAttachment`
 - `ec2:AcceptVpcEndpointConnections`
 - `ec2:AcceptVpcPeeringConnection`
 - `ec2:CopySnapshot`
 - `ec2>CreateVolume`

- ec2:CreateVpcEndpoint
- ec2:CreateVpcPeeringConnection
- ec2:DeleteTransitGatewayPeeringAttachment
- ec2:DeleteVpcPeeringConnection
- ec2:RejectTransitGatewayPeeringAttachment
- ec2:RejectVpcEndpointConnections
- ec2:RejectVpcPeeringConnection
- Amazon EventBridge
 - events:PutEvents – EventBridge PutEvents calls on an event bus in another account, if that event bus was configured as a cross-account EventBridge target before March 2, 2023. For more information, see [Grant permissions to allow events from other AWS accounts](#) in the *Amazon EventBridge User Guide*.
- Amazon Route 53
 - route53:AssociateVpcWithHostedZone
 - route53>CreateVPCAssociationAuthorization
 - route53>DeleteVPCAssociationAuthorization
 - route53:DisassociateVPCFromHostedZone
 - route53>ListHostedZonesByVPC
- Amazon WorkSpaces
 - workspaces:DescribeWorkspaceImages
- **Value type** – Single-valued

Note

For additional considerations for the above unsupported actions, see the [Data Perimeter Policy Examples](#) repository.

This global key returns the resource organization ID for a given request. It allows you to create rules that apply to all resources in an organization that are specified in the Resource element of an [identity-based policy \(p. 511\)](#). You can specify the [organization ID](#) in the condition element. When you add and remove accounts, policies that include the aws:ResourceOrgID key automatically include the correct accounts and you don't have to manually update it.

For example, the following policy prevents the principal from adding objects to the policy-genius-dev resource unless the Amazon S3 resource belongs to the same organization as the principal making the request.

Important

This policy does not allow any actions. Instead, it uses the Deny effect which explicitly denies access to all of the resources listed in the statement that do not belong to the listed account. Use this policy in combination with other policies that allow access to specific resources.

```
{  
    "Version": "2012-10-17",  
    "Statement": {  
        "Sid": "DenyPutObjectToS3ResourcesOutsideMyOrganization",  
        "Effect": "Deny",  
        "Action": "s3:PutObject",  
        "Resource": "arn:partition:s3:::policy-genius-dev/*",  
        "Condition": {  
            "StringNotEquals": {  
                "aws:ResourceOrgID": "${aws:PrincipalOrgID}"  
            }  
        }  
    }  
}
```

}

Note

Some AWS services require access to AWS owned resources that are hosted in another AWS account. Using `aws:ResourceOrgID` in your identity-based policies might impact your identity's ability to access these resources.

Certain AWS services, such as AWS Data Exchange, rely on access to resources outside of your AWS accounts for normal operations. If you use the `aws:ResourceOrgID` key in your policies, include additional statements to create exemptions for those services. The example policy [AWS: Deny access to Amazon S3 resources outside your account except AWS Data Exchange \(p. 545\)](#) demonstrates how to deny access based on the resource account while defining exceptions for service-owned resources. You can create a similar policy to restrict access to resources within your organization using the `aws:ResourceOrgID` key, while accounting for service-owned resources.

Use this policy example as a template for creating your own custom policies. Refer to your service [documentation](#) for more information.

In the following video, learn more about how you might use the `aws:ResourceOrgID` condition key in a policy.

[Ensure identities and networks can only be used to access trusted resources.](#)

aws:ResourceOrgPaths

Works with [string operators \(p. 1282\)](#).

Use this key to compare the AWS Organizations path for the accessed resource to the path in the policy. In a policy, this condition key ensures that the resource belongs to 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 \(p. 625\)](#)

- **Availability** – This key is included in the request context only if the account that owns the resource is a member of an organization. This global condition key does not support the following actions:
 - Amazon Elastic Block Store – All actions
 - Amazon EC2
 - `ec2:AcceptTransitGatewayPeeringAttachment`
 - `ec2:AcceptVpcEndpointConnections`
 - `ec2:AcceptVpcPeeringConnection`
 - `ec2:CopySnapshot`
 - `ec2>CreateVolume`
 - `ec2>CreateVpcEndpoint`
 - `ec2>CreateVpcPeeringConnection`
 - `ec2>DeleteTransitGatewayPeeringAttachment`
 - `ec2>DeleteVpcPeeringConnection`
 - `ec2:RejectTransitGatewayPeeringAttachment`
 - `ec2:RejectVpcEndpointConnections`
 - `ec2:RejectVpcPeeringConnection`
 - Amazon EventBridge
 - `events:PutEvents` – EventBridge PutEvents calls on an event bus in another account, if that event bus was configured as a cross-account EventBridge target before March 2, 2023. For more information, see [Grant permissions to allow events from other AWS accounts](#) in the *Amazon EventBridge User Guide*.

- Amazon Route 53
 - route53:AssociateVpcWithHostedZone
 - route53>CreateVPCAssociationAuthorization
 - route53>DeleteVPCAssociationAuthorization
 - route53:DisassociateVPCFromHostedZone
 - route53>ListHostedZonesByVPC
- Amazon WorkSpaces
 - workspaces:DescribeWorkspaceImages
- **Value type – Multivalued**

Note

For additional considerations for the above unsupported actions, see the [Data Perimeter Policy Examples](#) repository.

`aws:ResourceOrgPaths` is a multivalued condition key. Multivalued keys can have multiple values in the request context. You must use the `ForAnyValue` or `ForAllValues` set operators with [string condition operators \(p. 1282\)](#) for this key. For more information about multivalued condition keys, see [Multivalued context keys \(p. 1294\)](#).

For example, the following condition returns True for resources that belong to the organization `o-a1b2c3d4e5`. When you include a wildcard, you must use the [StringLike \(p. 1281\)](#) condition operator.

```
"Condition": {  
    "ForAnyValue:StringLike": {  
        "aws:ResourceOrgPaths": ["o-a1b2c3d4e5/*"]  
    }  
}
```

The following condition returns True for resources with the OU ID `ou-ab12-11111111`. It will match resources owned by accounts attached to the OU `ou-ab12-11111111` or any of the child OUs.

```
"Condition": { "ForAnyValue:StringLike" : {  
    "aws:ResourceOrgPaths": ["o-a1b2c3d4e5/r-ab12/ou-ab12-11111111/*"]  
}}
```

The following condition returns True for resources owned by accounts attached directly to the OU ID `ou-ab12-22222222`, but not the child OUs. The following example uses the [StringEquals \(p. 1281\)](#) condition operator to specify the exact match requirement for the OU ID and not a wildcard match.

```
"Condition": { "ForAnyValue:StringEquals" : {  
    "aws:ResourceOrgPaths": ["o-a1b2c3d4e5/r-ab12/ou-ab12-11111111/ou-ab12-22222222/"]  
}}
```

Note

Some AWS services require access to AWS owned resources that are hosted in another AWS account. Using `aws:ResourceOrgPaths` in your identity-based policies might impact your identity's ability to access these resources.

Certain AWS services, such as AWS Data Exchange, rely on access to resources outside of your AWS accounts for normal operations. If you use the `aws:ResourceOrgPaths` key in your policies, include additional statements to create exemptions for those services. The example policy [AWS: Deny access to Amazon S3 resources outside your account except AWS Data Exchange \(p. 545\)](#) demonstrates how to deny access based on the resource account while defining exceptions for service-owned resources. You

can create a similar policy to restrict access to resources within an organizational unit (OU) using the `aws:ResourceOrgPaths` key, while accounting for service-owned resources.

Use this policy example as a template for creating your own custom policies. Refer to your service [documentation](#) for more information.

aws:ResourceTag/*tag-key*

Works with [string operators \(p. 1282\)](#).

Use this key to compare the tag key-value pair that you specify in the policy with the key-value pair 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 \(p. 523\)](#).

- **Availability** – This key is included in the request context when the requested resource already has attached tags or in requests that create a resource with an attached tag. This key is returned only for resources that [support authorization based on tags \(p. 1224\)](#). There is one context key for each tag key-value pair.
- **Value type** – Single-valued

This context key is formatted "`aws:ResourceTag/tag-key" ":" tag-value" where tag-key and tag-value are a tag key and value pair. Tag keys and values are not case-sensitive. This means that if you specify "aws: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.`

For examples of using the `aws:ResourceTag` key to control access to IAM resources, see [Controlling access to AWS resources \(p. 523\)](#).

For examples of using the `aws:ResourceTag` key to control access to other AWS resources, see [Controlling access to AWS resources using tags \(p. 523\)](#).

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 \(p. 51\)](#).

aws:SecureTransport

Works with [Boolean operators \(p. 1285\)](#).

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.
- **Value type** – Single-valued

aws:SourceAccount

Works with [string operators \(p. 1282\)](#).

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 account ID of the source to the called service. This account ID includes the source account ID.
- **Value type** – Single-valued

You can use this condition key to prevent an AWS service from being used as a [confused deputy \(p. 193\)](#) during transactions between services. Set the value of this condition key to the account of the resource in the request. For example, when an Amazon S3 bucket update triggers an Amazon SNS topic post, the Amazon S3 service invokes the sns:Publish API operation. In the policy that allows the sns:Publish operation, set the value of the condition key to the account ID of the Amazon S3 bucket. For information about how and when these condition keys are recommended, see the documentation for the AWS services you are using.

aws:SourceArn

Works with [ARN operators \(p. 1288\)](#) and [string operators \(p. 1282\)](#). AWS recommends that you use ARN operators instead of string operators when comparing ARNs.

Use this key to compare the [Amazon Resource Name \(ARN\) \(p. 1213\)](#) of the resource making a service-to-service request with the ARN that you specify in the policy. The source's ARN includes the account ID, so it is not necessary to use aws:SourceAccount with aws:SourceArn.

This key does not work with the ARN of the principal making the request. Instead, use [aws:PrincipalArn \(p. 1347\)](#).

- **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.
- **Value type** – Single-valued

You can use this condition key to prevent an AWS service from being used as a [confused deputy \(p. 193\)](#) during transactions between services. Set the value of this condition key to the ARN of the resource in the request. For example, when an Amazon S3 bucket update triggers an Amazon SNS topic post, the Amazon S3 service invokes the sns:Publish API operation. In the policy that allows the sns:Publish operation, set the value of the condition key to the ARN of the Amazon S3 bucket. For information about how and when these condition keys are recommended, see the documentation for the AWS services you are using.

aws:SourceIdentity

Works with [string operators \(p. 1282\)](#).

Use this key to compare the source identity that was set by the principal with the source identity that you specify in the policy.

- **Availability** – This key is included in the request context after a source identity has been set when a role is assumed using any AWS STS assume-role CLI command, or AWS STS AssumeRole API operation.
- **Value type** – Single-valued

You can use this key in a policy to allow actions in AWS by principals that have set a source identity when assuming a role. Activity for the role's specified source identity appears in [AWS CloudTrail \(p. 473\)](#). This makes it easier for administrators to determine who or what performed actions with a role in AWS.

Unlike [sts:RoleSessionName \(p. 1381\)](#), after the source identity is set, the value cannot be changed. It is present in the request context for all actions taken by the role. The value persists into subsequent role sessions when you use the session credentials to assume another role. Assuming one role from another is called [role chaining \(p. 185\)](#).

The [sts:SourceIdentity \(p. 1382\)](#) key is present in the request when the principal initially sets a source identity while assuming a role using any AWS STS assume-role CLI command, or AWS STS

AssumeRole API operation. The `aws:SourceIdentity` key is present in the request for any actions that are taken with a role session that has a source identity set.

The following role trust policy for `CriticalRole` in account 111122223333 contains a condition for `aws:SourceIdentity` that prevents a principal without a source identity that is set to Saanvi or Diego from assuming the role.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AssumeRoleIfSourceIdentity",  
            "Effect": "Allow",  
            "Principal": {"AWS": "arn:aws:iam::123456789012:role/CriticalRole"},  
            "Action": [  
                "sts:AssumeRole",  
                "sts:SetSourceIdentity"  
            ],  
            "Condition": {  
                "StringLike": {  
                    "aws:SourceIdentity": ["Saanvi", "Diego"]  
                }  
            }  
        }  
    ]  
}
```

To learn more about using source identity information, see [Monitor and control actions taken with assumed roles \(p. 444\)](#).

aws:SourceIp

Works with [IP address operators \(p. 1286\)](#).

Use this key to compare the requester's IP address with the IP address that you specify in the policy. The `aws:SourceIp` condition key can only be used for public IP address ranges.

- **Availability** – This key is included in the request context, except when the requester uses a VPC endpoint to make the request.
- **Value type** – Single-valued

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 \(p. 1366\)](#) key to ensure that the source IP restriction applies only to requests made directly by a principal.

Note

`aws:SourceIp` supports both IPv4 and IPv6 address or range of IP addresses. For details, see [Upgrade IAM policies to IPv6](#).

For example, you can attach the following policy to an IAM user. This policy allows the user to put an object into the `DOC-EXAMPLE-BUCKET3` 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:::DOC-EXAMPLE-BUCKET3/*",
        "Condition": {
            "Bool": {"aws:ViaAWSService": "false"},
            "IpAddress": {"aws:SourceIp": "203.0.113.0"}
        }
    },
    {
        "Sid": "ServicePutObject",
        "Effect": "Allow",
        "Action": "s3:PutObject",
        "Resource": "arn:aws:s3:::DOC-EXAMPLE-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 \(p. 1363\)](#). For more information about using VPC endpoints, see [Identity and access management for VPC endpoints](#) and [VPC endpoint services](#) in the *AWS PrivateLink Guide*.

aws:SourceVpc

Works with [string operators \(p. 1282\)](#).

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 User Guide*.

- **Availability** – This key is included in the request context only if the requester uses a VPC endpoint to make the request.
- **Value type** – Single-valued

aws:SourceVpce

Works with [string operators \(p. 1282\)](#).

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 User Guide*.

- **Availability** – This key is included in the request context only if the requester uses a VPC endpoint to make the request.
- **Value type** – Single-valued

aws:TagKeys

Works with [string operators \(p. 1282\)](#).

Use this key to compare the tag keys in a request with the keys that you specify in the policy. We recommend that 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" \(p. 525\)](#).

- **Availability** – This key is included in the request context if the operation supports passing tags in the request.
- **Value type** – Multivalued

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 \(p. 1293\)](#) request. In this case, you must use the `ForAllValues` or `ForAnyValue` set operators. For more information, see [Multivalued context keys \(p. 1294\)](#).

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 \(p. 1285\)](#).

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. The key is not present in AWS CLI, AWS API, or AWS SDK requests that are made using access keys.
- **Value type** – Single-valued

To learn which services support using temporary credentials, see [AWS services that work with IAM \(p. 1224\)](#).

aws:UserAgent

Works with [string operators \(p. 1282\)](#).

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.
- **Value type** – Single-valued

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 \(p. 1282\)](#).

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 \(p. 1264\)](#). For specific examples of principal key values, see [Principal key values \(p. 1303\)](#).

- **Availability** – This key is included in the request context for all requests, including anonymous requests.
- **Value type** – Single-valued

aws:username

Works with [string operators \(p. 1282\)](#).

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 \(p. 1264\)](#). For specific examples of principal key values, see [Principal key values \(p. 1303\)](#).

- **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 IAM Identity Center credentials do not include this key in the context.
- **Value type** – Single-valued

aws:ViaAWSService

Works with [Boolean operators \(p. 1285\)](#).

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.
- **Value type** – Single-valued

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 \(p. 544\)](#).

aws:VpcSourceIp

Works with [IP address operators \(p. 1286\)](#).

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.
- **Value type** – Single-valued

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 \(p. 1376\)](#). These keys are available when a user who was federated using SAML performs AWS operations in other services. Other examples include [saml:namequalifier](#) 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 \(p. 1278\)](#).

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 User Guide*.

Topics

- [Available keys for IAM \(p. 1367\)](#)
- [Available keys for AWS web identity federation \(p. 1372\)](#)
- [Cross-service AWS web identity federation context keys \(p. 1375\)](#)
- [Available keys for SAML-based AWS STS federation \(p. 1376\)](#)
- [Cross-service SAML-based AWS STS federation context keys \(p. 1379\)](#)
- [Available keys for AWS STS \(p. 1379\)](#)

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 \(p. 1288\)](#).

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 \(p. 279\)](#) 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. In addition, 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"},
        "ArnLike": [
            "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 \(p. 1371\)](#) also support this condition key.

iam:AWSServiceName

Works with [string operators \(p. 1282\)](#).

Specifies the AWS service to which this role is attached.

In this example, you allow an entity to create a service-linked role if the service name is *access-analyzer.amazonaws.com*.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "access-analyzer.amazonaws.com"
                }
            }
        }
    ]
}
```

iam:FIDO-certification

Works with [string operators \(p. 1282\)](#).

Checks the MFA device FIDO certification level at the time of registration of a FIDO security key. The device certification is retrieved from the [FIDO Alliance Metadata Service \(MDS\)](#). If the certification status or level of your FIDO security key changes, it will not be updated unless the device is unregistered and registered again to fetch the updated certification information.

Possible values of L1, L1plus, L2, L2plus, L3, L3plus

In this example, you register a security key and retrieve the FIDO Level 1 plus certification for your device.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:EnableMFADevice",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:RegisterSecurityKey" : "Create"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "iam:EnableMFADevice",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:RegisterSecurityKey" : "Activate",
                    "iam:FIDO-certification": "L1plus"
                }
            }
        }
    ]
}
```

iam:FIDO-FIPS-140-2-certification

Works with [string operators \(p. 1282\)](#).

Checks the MFA device FIPS-140-2 validation certification level at the time of registration of a FIDO security key. The device certification is retrieved from the [FIDO Alliance Metadata Service \(MDS\)](#). If the certification status or level of your FIDO security key changes, it will not be updated unless the device is unregistered and registered again to fetch the updated certification information.

Possible values of L1, L2, L3, L4

In this example, you register a security key and retrieve the FIPS-140-2 Level 2 certification for your device.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:EnableMFADevice",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:RegisterSecurityKey" : "Create"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "iam:EnableMFADevice",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:FIPS-140-2-certification" : "L2"
                }
            }
        }
    ]
}
```

```
        "Condition": {
            "StringEquals": {
                "iam:RegisterSecurityKey" : "Activate",
                "iam:FIDO-FIPS-140-2-certification": "L2"
            }
        }
    ]
}
```

iam:FIDO-FIPS-140-3-certification

Works with [string operators \(p. 1282\)](#).

Checks the MFA device FIPS-140-3 validation certification level at the time of registration of a FIDO security key. The device certification is retrieved from the [FIDO Alliance Metadata Service \(MDS\)](#). If the certification status or level of your FIDO security key changes, it will not be updated unless the device is unregistered and registered again to fetch the updated certification information.

Possible values of L1, L2, L3, L4

In this example, you register a security key and retrieve the FIPS-140-3 Level 3 certification for your device.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:EnableMFADevice",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:RegisterSecurityKey" : "Create"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "iam:EnableMFADevice",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:RegisterSecurityKey" : "Activate",
                    "iam:FIDO-FIPS-140-3-certification": "L3"
                }
            }
        }
    ]
}
```

iam:RegisterSecurityKey

Works with [string operators \(p. 1282\)](#).

Checks the current state of MFA device enablement.

Possible values of Create or Activate.

In this example, you register a security key and retrieve the FIPS-140-3 Level 1 certification for your device.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:EnableMFADevice",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:RegisterSecurityKey" : "Create"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "iam:EnableMFADevice",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:RegisterSecurityKey" : "Activate",
                    "iam:FIDO-FIPS-140-3-certification": "L1"
                }
            }
        }
    ]
}
```

iam:OrganizationsPolicyId

Works with [string operators \(p. 1282\)](#).

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 \(p. 570\)](#).

iam:PassedToService

Works with [string operators \(p. 1282\)](#).

Specifies the service principal of the service to which a role can be passed. This condition key applies to only the [PassRole \(p. 279\)](#) 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 \(p. 185\)](#) 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 \(p. 564\)](#).

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.

Sometimes you pass a role to a service that then passes the role to a different service. iam:PassedToService includes only the final service that assumes the role, not the intermediate service that passes the role.

Note

Some services do not support this condition key.

iam:PermissionsBoundary

Works with [ARN operators \(p. 1288\)](#).

Checks that the specified policy is attached as permissions boundary on the IAM principal resource. For more information, see [Permissions boundaries for IAM entities \(p. 501\)](#)

iam:PolicyARN

Works with [ARN operators \(p. 1288\)](#).

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 \(p. 517\)](#).

iam:ResourceTag/*key-name*

Works with [string operators \(p. 1282\)](#).

Checks that the tag attached to the identity resource (user or role) matches the specified key name and value.

Note

IAM and AWS STS support both the `iam:ResourceTag` IAM condition key and the `aws:ResourceTag` global condition key.

You can add custom attributes to IAM resources in the form of a key-value pair. For more information about tags for IAM resources, see [the section called “Tagging IAM resources” \(p. 399\)](#). You can use `ResourceTag` to [control access \(p. 523\)](#) to AWS resources, including IAM 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 an identity-based 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 \(p. 581\)](#) or [edit a policy \(p. 609\)](#).

```
{  
    "Version": "2012-10-17",  
    "Statement": [{  
        "Effect": "Allow",  
        "Action": "iam:DeleteUser",  
        "Resource": "*",  
        "Condition": {"StringEquals": {"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 OpenID Connect compliant OpenID Provider (OP) to an IAM OpenID Connect (OIDC) identity provider in your AWS account. Examples of such providers include Login with Amazon, Amazon Cognito, Google, or Facebook. Identity tokens (`id_tokens`) from your own OpenID OP may be used, as well as `id_tokens` issued to service accounts of Amazon Elastic Kubernetes Service clusters. 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. Define condition keys using the name of the OIDC provider followed by the

claim (:aud, :azp, :amr, sub). For roles used by Amazon Cognito, keys are defined using cognito-identity.amazonaws.com followed by the claim.

amr

Works with [string operators \(p. 1282\)](#).

Example: cognito-identity.amazonaws.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 \(p. 1294\)](#). 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 \(p. 1282\)](#).

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:

- graph.facebook.com:app_id
- accounts.google.com:aud
- cognito-identity.amazonaws.com:aud

The graph.facebook.com:app_id field supplies the audience context that matches the aud field used by other identity providers.

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 \(p. 1374\)](#) 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 \(p. 1282\)](#).

Examples:

- 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 \(p. 1282\)](#).

Example: accounts.google.com:aud

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 \(p. 1282\)](#).

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.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        "Condition": {  
            "StringEquals": {  
                "oidc.eks.us-east-1.amazonaws.com/id/111122223333:aud":  
                "sts.amazonaws.com",  
                "oidc.eks.us-east-1.amazonaws.com/id/111122223333:sub":  
                "system:serviceaccount:default:assumer"  
            }  
        }  
    ]  
}
```

More information about web identity federation

For more information about web identity federation, see the following:

- [Amazon Cognito User Guide](#)
- [About web identity federation \(p. 199\)](#)

Cross-service AWS web identity federation context keys

Some web identity federation condition keys can be used in role trust policies to define what users are allowed to access in other AWS services. These are the following condition keys that can be used in role trust policies when federated principals assume another role, and in resource policies from other AWS services to authorize resource access by federated principals. If you are using Amazon Cognito for web identity federation, then these keys are available when the user is authenticated.

Select a condition key to see the description.

- [amr](#)
- [aud](#)
- [id](#)
- [sub](#)

Note

No other web identitiy based federation condition keys are available for use after the external identity provider (IdP) authentication and authorization for the initial AssumeRoleWithWebIdentity operation.

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 \(p. 1282\)](#).

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 \(p. 1282\)](#).

This is a commonName attribute.

`saml:cn[]`

Works with [string operators \(p. 1282\)](#).

This is an eduOrg attribute.

`saml:doc`

Works with [string operators \(p. 1282\)](#).

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 \(p. 218\)](#).

`saml:edupersonaffiliation[]`

Works with [string operators \(p. 1282\)](#).

This is an eduPerson attribute.

`saml:edupersonassurance[]`

Works with [string operators \(p. 1282\)](#).

This is an eduPerson attribute.

`saml:edupersonentitlement[]`

Works with [string operators \(p. 1282\)](#).

This is an eduPerson attribute.

`saml:edupersonnickname[]`

Works with [string operators \(p. 1282\)](#).

This is an eduPerson attribute.

`saml:edupersonorgdn`

Works with [string operators \(p. 1282\)](#).

This is an eduPerson attribute.

saml:edupersonorgunitdn[]

Works with [string operators \(p. 1282\)](#).

This is an eduPerson attribute.

saml:edupersonprimaryaffiliation

Works with [string operators \(p. 1282\)](#).

This is an eduPerson attribute.

saml:edupersonprimaryorgunitdn

Works with [string operators \(p. 1282\)](#).

This is an eduPerson attribute.

saml:edupersonprincipalname

Works with [string operators \(p. 1282\)](#).

This is an eduPerson attribute.

saml:edupersonscopedaffiliation[]

Works with [string operators \(p. 1282\)](#).

This is an eduPerson attribute.

saml:edupersontargetedid[]

Works with [string operators \(p. 1282\)](#).

This is an eduPerson attribute.

saml:eduorghomepageuri[]

Works with [string operators \(p. 1282\)](#).

This is an eduOrg attribute.

saml:eduorgidentityauthnpolicyuri[]

Works with [string operators \(p. 1282\)](#).

This is an eduOrg attribute.

saml:eduorglegalname[]

Works with [string operators \(p. 1282\)](#).

This is an eduOrg attribute.

saml:eduorgsuperioruri[]

Works with [string operators \(p. 1282\)](#).

This is an eduOrg attribute.

saml:eduorgwhitepagesuri[]

Works with [string operators \(p. 1282\)](#).

This is an eduOrg attribute.

saml:givenName[]

Works with [string operators \(p. 1282\)](#).

This is a givenName attribute.

saml:iss

Works with [string operators \(p. 1282\)](#).

The issuer, which is represented by a URN.

saml:mail[]

Works with [string operators \(p. 1282\)](#).

This is a mail attribute.

saml:name[]

Works with [string operators \(p. 1282\)](#).

This is a name attribute.

saml:namequalifier

Works with [string operators \(p. 1282\)](#).

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 \(p. 208\)](#).

saml:organizationStatus[]

Works with [string operators \(p. 1282\)](#).

This is an organizationStatus attribute.

saml:primaryGroupSID[]

Works with [string operators \(p. 1282\)](#).

This is a primaryGroupSID attribute.

saml:sub

Works with [string operators \(p. 1282\)](#).

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 \(p. 1282\)](#).

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 \(p. 225\)](#).

saml:surname[]

Works with [string operators \(p. 1282\)](#).

This is a surnameuid attribute.

saml:uid[]

Works with [string operators \(p. 1282\)](#).

This is a uid attribute.

saml:x500UniqueIdentifier[]

Works with [string operators \(p. 1282\)](#).

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 \(p. 1294\)](#) (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" ]  
    }  
}
```

Cross-service SAML-based AWS STS federation context keys

Some SAML-based federation condition keys can be used in subsequent requests to authorize AWS operations in other services and AssumeRole calls. These are the following condition keys that can be used in role trust policies when federated principals assume another role, and in resource policies from other AWS services to authorize resource access by federated principals. For more information about using these keys, see [About SAML 2.0-based federation](#).

Select a condition key to see the description.

- [saml:namequalifier](#)
- [saml:sub](#)
- [saml:sub_type](#)

Note

No other SAML-based federation condition keys are available for use after the initial external identity provider (IdP) authentication response.

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.

saml:sub

Works with [string operators \(p. 1282\)](#).

This is the subject of the claim, which includes a value that uniquely identifies an individual user within an organization (for example, _cbb88bf52c2510eabe00c1642d4643f41430fe25e3).

sts:AWSServiceName

Works with [string operators \(p. 1282\)](#).

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 \(p. 467\)](#).

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 \(p. 1284\)](#).

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 \(p. 467\)](#).

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 \(p. 1282\)](#).

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 \(p. 191\)](#).

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 \(p. 1282\)](#).

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, any assume-role CLI command, or any AWS STS 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 \(p. 473\)](#) with the session name that matches their user name. This makes it easier for administrators to differentiate between role sessions when a role is used by different principals.

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 \(p. 1298\)](#) 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.

Important

You can use any single-valued condition key as a [variable \(p. 1298\)](#). You can't use a multivalued condition key as a variable.

```
{
    "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": "AROACQRSTUVWRAOEXAMPLE:matjac",
    "arn": "arn:aws:sts::111122223333:assumed-role/MateoRole/matjac"
}
```

If you allow [cross-account access using roles \(p. 188\)](#), 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:SourceIdentity

Works with [string operators \(p. 1282\)](#).

Use this key to compare the source identity 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 provides a source identity while assuming a role using any AWS STS assume-role CLI command, or AWS STS AssumeRole API operation.

You can use this key in a role trust policy to require that your users set a specific source identity when they assume a role. For example, you can require your workforce or federated identities to specify a value for source identity. You can configure your identity provider (IdP) to use one of the attributes that are associated with your users, like a user name or email as the source identity. The IdP then passes the source identity as an attribute in the assertions or claims that it sends to AWS. The value of the source identity attribute identifies the user or application who is assuming the role.

After the user assumes the role, activity appears in [AWS CloudTrail logs \(p. 473\)](#) with the source identity value that was set. This makes it easier for administrators to determine who or what performed actions with a role in AWS. You must grant permissions for the sts: SetSourceIdentity action to allow an identity to set a source identity.

Unlike [sts:RoleSessionName \(p. 1381\)](#), after the source identity is set, the value cannot be changed. It is present in the request context for all actions taken with the role by the source identity. The value persists into subsequent role sessions when you use the session credentials to assume another role. Assuming one role from another is called [role chaining \(p. 185\)](#).

You can use the [aws:SourceIdentity \(p. 1362\)](#) global condition key to further control access to AWS resources based on the value of source identity in subsequent requests.

The following role trust policy allows the IAM user AdminUser to assume a role in account 111122223333. It also grants permission to the AdminUser to set a source identity, as long as the source identity set is DiegoRamirez.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowAdminUserAssumeRole",  
            "Effect": "Allow",  
            "Principal": {"AWS": "arn:aws:iam::111122223333:user/AdminUser"},  
            "Action": [  
                "sts:AssumeRole",  
                "sts:SetSourceIdentity"  
            ],  
            "Condition": {  
                "StringEquals": {"sts:SourceIdentity": "DiegoRamirez"}  
            }  
        }  
    ]  
}
```

To learn more about using source identity information, see [Monitor and control actions taken with assumed roles \(p. 444\)](#).

sts:TransitiveTagKeys

Works with [string operators \(p. 1282\)](#).

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 \(p. 1279\)](#) includes the [aws:PrincipalTag \(p. 1352\)](#) context key. This key includes a list of [session tags \(p. 417\)](#), [transitive session tags \(p. 424\)](#), 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 \(p. 185\)](#).

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

- [Identities \(p. 1384\)](#)
- [Credentials \(passwords, access keys, and MFA devices\) \(p. 1384\)](#)
- [Permissions and policies \(p. 1384\)](#)
- [Federation and delegation \(p. 1385\)](#)
- [IAM and other AWS products \(p. 1385\)](#)
- [General security practices \(p. 1386\)](#)
- [General resources \(p. 1386\)](#)

Identities

Consult these resources for creating, managing, and using identities.

- [Manage identities in IAM Identity Center](#) – Procedural information about creating users and group in IAM Identity Center.
- [IAM Identities \(users, user groups, and roles\) \(p. 70\)](#) – An in-depth discussion of users, groups, and roles.

Credentials (passwords, access keys, and MFA devices)

Review the following guides to manage passwords, access keys, and MFA devices for your AWS account and for IAM users.

- [Managing user passwords in AWS \(p. 93\)](#) – Describes options for managing passwords for IAM users in your account.
- [Managing access keys for IAM users \(p. 103\)](#) – Describes how access keys work and how you can use them to make programmatic calls to AWS. There are other more secure alternatives to access keys that we recommend you consider first. For more information, see [Considerations and alternatives for long-term access keys](#) in the [AWS General Reference guide](#).
- [Using multi-factor authentication \(MFA\) in AWS \(p. 114\)](#) – 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.)

For general information about the types of credentials you use to access Amazon Web Services, see [AWS Security Credentials](#) in the [AWS General Reference guide](#).

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 \(p. 485\)](#) – 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 \(p. 1260\)](#) – Provides descriptions and examples of each policy language element.
- [Validating IAM policies \(p. 588\)](#) – Find resources for JSON policy validation.
- [Example IAM identity-based policies \(p. 529\)](#) – 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 \(p. 40\)](#) – Guides you through granting cross-account access to an IAM user in another AWS account.
- [Common scenarios for temporary credentials \(p. 426\)](#) – 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 \(p. 1224\)](#).

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 \(p. 381\)](#) – 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:

- [Best Practices for Security, Identity, & Compliance](#) – Find resources for how to manage security across AWS accounts and products, including suggestions for security architecture, use of IAM, encryption and data security, and more.
- [Identity and Access Management](#) – The AWS Well-Architected Framework helps you understand key concepts, design principles, and architectural best practices for designing and running workloads in the cloud.
- [Security best practices in IAM \(p. 1032\)](#) – 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.
- [AWS re:Post for AWS Identity and Access Management](#) – Visit AWS re:Post to discuss technical questions related to IAM with the AWS community.
- [Classes & Workshops](#) – Links to role-based and specialty courses, in addition to self-paced labs to help sharpen your AWS skills and gain practical experience.
- [AWS Developer Center](#) – Explore tutorials, download tools, and learn about AWS developer events.
- [AWS Developer Tools](#) – Links to developer tools, SDKs, IDE toolkits, and command line tools for developing and managing AWS applications.
- [Getting Started Resource Center](#) – Learn how to set up your AWS account, join the AWS community, and launch your first application.

- [Hands-On Tutorials](#) – Follow step-by-step tutorials to launch your first application on AWS.
- [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 \(p. 1388\)](#)
- [HTTPS required \(p. 1389\)](#)
- [Signing IAM API requests \(p. 1389\)](#)

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 \(p. 461\)](#).

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 \(p. 1032\)](#). Presents a list of suggestions for using IAM service to help secure your AWS resources.
- [Temporary security credentials in IAM \(p. 426\)](#). Describes how to create and use temporary security credentials.

Document history for IAM

The following table describes major documentation updates for IAM.

Change	Description	Date
<u>Support for multiple multi-factor authentication (MFA) devices for root users and IAM users</u>	Now you can add up to eight MFA devices per user, including FIDO security keys, software time-based one-time password (TOTP) with virtual authenticator applications, or hardware TOTP tokens.	November 16, 2022
<u>IAM Access Analyzer support for new resource types</u>	IAM Access Analyzer added support for the following resource types: <ul style="list-style-type: none"> Amazon EBS volume snapshots Amazon ECR repositories Amazon EFS file systems Amazon RDS DB snapshots Amazon RDS DB cluster snapshots Amazon SNS topics 	October 25, 2022
<u>U2F deprecation and WebAuthn/FIDO update</u>	Removed mentions of U2F as an MFA option and added information about WebAuthn, FIDO2, and FIDO security keys.	May 31, 2022
<u>Updates to resilience in IAM</u>	Added information about maintaining access to IAM credentials when an event disrupts communication between AWS Regions.	May 16, 2022
<u>New global condition keys for resources</u>	You can now control access to resources based on the account, Organizational Unit (OU), or organization in AWS Organizations that contains your resources. You can use the <code>aws:ResourceAccount</code> , <code>aws:ResourceOrgID</code> , and <code>aws:ResourceOrgPaths</code> global condition keys in an IAM policy.	April 27, 2022
<u>Code examples for IAM using AWS SDKs</u>	Added code examples that show how to use IAM with an AWS software development kit (SDK). The examples are divided into	April 7, 2022

code excerpts that show you how to call individual service functions and examples that show you how to accomplish a specific task by calling multiple functions within the same service.

<u>Updates to policy evaluation logic flow chart</u>	Updates to the policy evaluation logic flow chart and related text in the <u>Determining whether a request is allowed or denied within an account</u> section.	November 17, 2021
<u>Updates to security best practices</u>	Added information about creating administrative users instead of using root user credentials, removed the best practice of using user groups to assign permissions to IAM users, and clarified when to use managed policies instead of inline policies.	October 5, 2021
<u>Updates to policy evaluation logic topic for resource-based policies</u>	Added information about the impact of resource-based policies and different principal types in the same account.	October 5, 2021
<u>Updates to single-valued and multivalued condition keys</u>	The differences between single-valued and multivalued condition keys are now explained in more detail. The value type was added to each <u>AWS global condition context key</u> .	September 30, 2021
<u>IAM Access Analyzer supports Amazon S3 Multi-Region Access Points</u>	IAM Access Analyzer identifies Amazon S3 buckets that allow public and cross-account access, including those that use Amazon S3 <u>Multi-Region Access Points</u> .	September 2, 2021
<u>AWS managed policy updates - Update to an existing policy</u>	IAM Access Analyzer updated an existing AWS managed policy.	September 2, 2021
<u>More services supported for action-level policy generation</u>	IAM Access Analyzer can generate IAM policies with action-level access activity information for additional AWS services.	August 24, 2021

<u>Generate IAM policies for cross-account trails</u>	You can now use IAM Access Analyzer to generate fine-grained policies based on your access activity using a AWS CloudTrail trail in a different account, for example, a centralized AWS Organizations trail.	August 18, 2021
<u>Additional IAM Access Analyzer policy checks</u>	IAM Access Analyzer extended policy validation by adding new policy checks that validate conditions included in IAM policies. These checks analyze the condition block in your policy statement and report security warnings, errors, and suggestions along with actionable recommendations. IAM Access Analyzer added the following policy checks: <ul style="list-style-type: none">• <u>Error – Invalid service principal format</u>• <u>Error – Missing tag key in condition</u>• <u>Security Warning – Deny NotAction with unsupported tag condition key for service</u>• <u>Security Warning – Deny with unsupported tag condition key for service</u>• <u>Security Warning – Missing paired condition keys</u>• <u>Suggestion – Allow NotAction with unsupported tag condition key for service</u>• <u>Suggestion – Allow with unsupported tag condition key for service</u>	June 29, 2021
<u>Action last accessed support for more services</u>	You can now view action last accessed information in the IAM console about the last time an IAM principal used an action for the following services: Amazon EC2, IAM, Lambda, and Amazon S3 management actions. You can also use the AWS CLI or AWS API to retrieve a data report. You can use this information to identify unnecessary permissions so that you can refine your IAM policies to better adhere to the principle of least privilege.	April 19, 2021

<u>Monitor and control actions taken with assumed roles</u>	Administrators can configure IAM roles to require that identities pass a source identity, which is logged in AWS CloudTrail. Reviewing source identity information helps administrators determine who or what performed actions with assumed role sessions.	April 13, 2021
<u>Generate IAM policies based on access activity</u>	You can now use IAM Access Analyzer to generate fine-grained policies based on your access activity found in your AWS CloudTrail.	April 7, 2021
<u>IAM Access Analyzer policy checks</u>	IAM Access Analyzer now provides over 100 policy checks with actionable recommendations during policy authoring.	March 16, 2021
<u>Expanded policy validation options</u>	Expanded policy validation available in the IAM console, AWS API, and AWS CLI using policy checks in IAM Access Analyzer to help you author secure and functional JSON policies.	March 15, 2021
<u>Tagging IAM resources</u>	You can now tag additional IAM resources using a tag key-value pair.	February 11, 2021
<u>Default password policy for IAM users</u>	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
<u>The actions, resources, and condition keys pages for AWS services have moved</u>	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

<u>IAM users longer role session duration</u>	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
<u>Use Service Quotas to request quick increases for IAM entities</u>	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
<u>Last accessed information in IAM now includes Amazon S3 management actions</u>	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 unnecessary permissions so that you can refine your IAM policies to better adhere to the principle of least privilege.	June 3, 2020
<u>Security chapter addition</u>	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
<u>sts:RoleSessionName</u>	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

<u>AWS sign-in page update</u>	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 information. This documentation includes updated screen captures to help you understand the AWS sign-in pages.	March 4, 2020
<u>aws:ViaAWSService and aws:CalledVia condition keys</u>	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
<u>Policy simulator adds support for permissions boundaries</u>	You can now test the effect of permissions boundaries on IAM entities with the IAM policy simulator.	January 23, 2020
<u>Cross-account policy evaluation</u>	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

<u>Session tags</u>	You can now include tags when you assume a role or federate a user in AWS STS. When you perform the AssumeRole or GetFederationToken operation, you can pass the session tags as attributes. When you perform the AssumeRoleWithSAML or AssumeRoleWithWebIdentity operations, you can pass attributes from your corporate identities to AWS.	November 22, 2019
<u>Control access for groups of AWS accounts in AWS Organizations</u>	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 aws:PrincipalOrgPaths condition key in your policies.	November 20, 2019
<u>Role last used</u>	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
<u>Update to the global condition context keys page</u>	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 aws:userId key and not the aws:userName key. The aws:userName 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 [aws_sts_get-access-key-info](#) AWS CLI command or the [GetAccessKeyInfo](#) 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

<u>AWS STS Region compatibility of session tokens for the global endpoint</u>	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
<u>Allow enabling and disabling AWS regions</u>	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
<u>IAM user my security credentials page</u>	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
<u>Access advisor API</u>	You can now use the AWS CLI and AWS API to view service last accessed information.	December 7, 2018
<u>Tagging IAM users and roles</u>	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
<u>U2F security keys</u>	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
<u>Support for Amazon VPC endpoints</u>	You can now establish a private connection between your VPC and AWS STS in the US West (Oregon) Region.	July 31, 2018

<u>Permissions boundaries</u>	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
<u>aws:PrincipalOrgID</u>	New condition key provides an easier way to control access to AWS resources by specifying the AWS organization of IAM principals.	May 17, 2018
<u>aws:RequestedRegion</u>	New condition key provides an easier way to use IAM policies to control access to AWS Regions.	April 25, 2018
<u>Increased session duration for IAM roles</u>	An IAM role can now have a session duration of 12 hours.	March 28, 2018
<u>Updated role-creation workflow</u>	New workflow improves the process of creating trust relationships and attaching permissions to roles.	September 8, 2017
<u>AWS account sign-in process</u>	Updated AWS sign-in experience allows both the root user and IAM users to use the Sign In to the Console link on the AWS Management Console's home page.	August 25, 2017
<u>Example IAM policies</u>	Documentation update features more than 30 example policies.	August 2, 2017
<u>IAM best practices</u>	Information added to the Users section of the IAM console makes it easier to follow IAM best practices.	July 5, 2017
<u>Auto Scaling resources</u>	Resource-level permissions can control access to and permissions for Auto Scaling resources.	May 16, 2017
<u>Amazon RDS for MySQL and Amazon Aurora databases</u>	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
<u>Service-linked roles</u>	Service-linked roles provide an easier and more secure way to delegate permissions to AWS services.	April 19, 2017
<u>Policy summaries</u>	New policy summaries make it easier to understand permissions in IAM policies.	March 23, 2017