AWS Security Token Service

Using Temporary Security Credentials

API Version 2011-06-15

Amazon Web Services

AWS Security Token Service: Using Temporary Security Credentials

Amazon Web Services

Welcome

Topics

Introduction

The AWS Security Token Service lets you grant a trusted user temporary, limited access to your Amazon Web Services (AWS) resources. Here are some are examples of when temporary access is useful:

- Federation. You can grant temporary access to people in a corporate network without having to define individual IAM identities for each corporate user. You can also let federated users log into the AWS Management Console without having to be defined as IAM users, which we refer to as single sign-on (SSO). AWS STS supports open standards like the SAML 2.0 (Security Assertion Markup Language 2.0), or you can manage your own solution for federating user identities.
- Federation for mobile apps. You can grant access to a user who logs in to a mobile application using Login with Amazon, Facebook, or Google. Users don't have to have IAM identities. (We refer to this as *web identity federation*.)
- Cross-account access. This lets IAM users in one account access resources in another account. (We refer to this as *cross-account API access*.)
- Security management for applications running on Amazon EC2 instances that need access to AWS resources. (We refer to this as *delegating API access* by using roles.)
- Security management to scope down permissions at run time. This is useful for IAM users who are using multi-factor authentication (MFA).

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 Support AWS Security Token Service (AWS STS) (p. 52).

Ways to Get Temporary Security Credentials

To request temporary security credentials using the AWS Security Token Service (AWS STS), you write code to call the API actions listed in the following table. You can make these calls using one of the AWS SDKs, which are available in a variety of programming languages, including Java, .NET, Python, Ruby, Android, and iOS. The SDKs take care of tasks such as cryptographically signing your service 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.

AWS Security Token Service API Actions

Action	Description
AssumeRole	Returns a set of temporary security credentials. You call this API by using the credentials of an existing IAM user. This API is useful for granting AWS access to users who do not have an IAM identity (that is, to federated users). It is also useful for allowing existing IAM users to access AWS resources that they don't already have access to, such as resources in another account. For more information, see Creating Temporary Security Credentials for Delegating API Access (p. 26).
AssumeRoleWithWebIdentity	Returns a set of temporary security credentials for federated users who are authenticated using a public identity provider like Login with Amazon, Facebook, or Google. This API is useful for creating mobile applications or client-based web applications that require access to AWS but where users do not have their own AWS or IAM identity. For more information, see Creating a Role to Allow AWS Access for the Mobile App (p. 14).
AssumeRoleWithSAML	Returns a set of temporary security credentials for federated users who are authenticated in your organization and who pass authentication and authorization information to AWS using SAML (Security Assertion Markup Language). This API is useful in organizations that have integrated their identity systems (such as Windows Active Directory) with software that can produce SAML assertions to provide information about user identity and permissions. For more information, see Creating Temporary Security Credentials for SAML Federation (p. 19).
GetFederationToken	Returns a set of temporary security credentials for federated users. This API differs from `AssumeRole` in that the default expiration period is substantially longer (up to 36 hours instead of up to 1 hour); this can help reduce the number of calls to AWS because you do not need to get new credentials as often. For more information, see Creating Temporary Security Credentials to Enable Access for Federated Users (p. 22).

Action	Description
GetSessionToken	Returns a set of temporary security credentials to an existing IAM user. This API is useful for providing enhanced security, such as to make AWS requests when MFA is enabled for the IAM user. For more information, see Creating Temporary Security Credentials to Enable Access for IAM Users (p. 29).

The AWS STS API actions return temporary security credentials that consist of an access key ID, a secret access key, and a session token. Users (or an application that the user is running) can then use these temporary security credentials to access your resources. The temporary security credentials are associated with an IAM access control policy that limits what the user can do when using these credentials. For more information, see Using Temporary Security Credentials (p. 38).

Important
Although temporary security credentials are short-lived, users who have temporary access can make lasting changes to your AWS resources. For example, if a user with temporary access launches an Amazon EC2 instance, the instance can continue to run and incur charges against your AWS account even after the user's temporary security credentials expire.

Advantages of Temporary Security Credentials

Using AWS Security Token Service to get temporary security credentials is useful for the following reasons:

- You do not have to distribute 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.
- The temporary security credentials have a limited lifetime, meaning that you do not have to rotate them or explicitly revoke them when they're no longer needed. After temporary security credentials have expired, they cannot be reused. You can specify how long the credentials are good for, up to a maximum limit.

Scenarios for Granting Temporary Access

Topics

You might choose to use temporary security credentials for several reasons. This section describes the most common scenarios.

Creating a Mobile App with Third-Party Sign-In

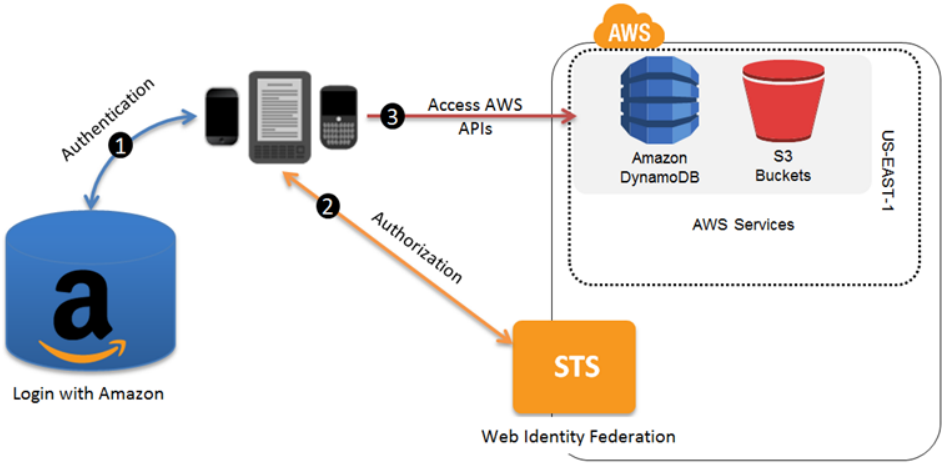
Adele the developer is building a game for a mobile device where user information such as scores and profiles is stored using Amazon S3 and Amazon DynamoDB. She knows that for security and maintenance reasons, long-term AWS security credentials should not be distributed with the game. She also knows that the game might have a large number of users. For all of these reasons, she does not create new user identities for each player. Instead, she builds the game so that users can sign in using an identity that they've already established with Amazon.com, Facebook, or Google. 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 company's AWS resources, Adele first registers for a developer ID with Login with Amazon, Facebook, and Google. She also configures the application with each of these providers. In the AWS account that owns the Amazon S3 bucket and Amazon DynamoDB table for the game, Adele creates IAM roles that precisely define permissions that the game needs.

In the app's code, Adele calls the sign-in interface for the identity provider that the user selects. The provider handles all the details of letting the user sign in, and the app gets an OAuth access token or OpenID Connect (OIDC) ID token from the provider. Using this information, Adele's app can call the AWS

Security Token Service (AWS STS) `AssumeRoleWithWebIdentity` action. This action returns temporary security credentials consisting of an AWS access key ID, a secret access key, and a session token. The user's instance of the app caches the temporary security credentials and uses them to access AWS services. The app is limited to the permissions defined in the role that it assumes. When the temporary credentials expire, the mobile app makes another call to AWS STS in order to get a new set of temporary security credentials.

The following figure shows a simplified flow for how this might work, using Login with Amazon as the identity provider. For Step 1, the app can also invoke Facebook or Google, but that's not shown here.



The following details enable this scenario:

- Adele the developer has registered the mobile app with different identity providers, who have assigned an app ID to the app.
- The mobile app includes logic to invoke the appropriate identity provider (depending on which sign-in option the user chooses) and to get back a token from the provider.
- The app can call `AssumeRoleWithWebIdentity` without using any AWS security credentials. The call includes the token from the provider received previously.
- AWS STS is able to verify that the token passed from Adele's app is valid and then returns temporary security credentials to the app. The mobile app's permissions to access AWS are established by the role that the app assumes.

You can learn more about web identity federation by working with the following sample applications:

- 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 AWS SDK for iOS and AWS SDK for Android toolkits include a sample application that demonstrates how to access an Amazon S3 bucket.

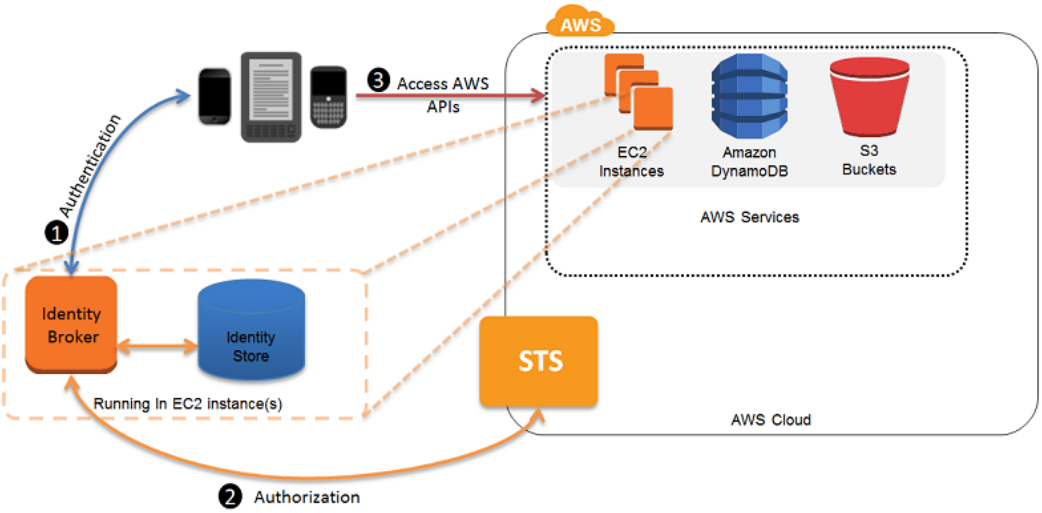
Creating a Mobile App with Custom Authentication

A company is building a mobile app that enables the app's registered users to access the company's AWS resources on the back end. Unlike the previous scenario, users don't sign in using one of the supported web identity federation providers (Login with Amazon, Facebook, or Google). Instead, the

company wants to use a custom solution for authenticating users and for managing the identity store. As in the previous scenario, the company doesn't want to distribute any AWS security credentials with the app.

Dave is the developer for this app. To enable the mobile app to access the company's AWS resources, Dave develops a custom identity broker application that runs on Amazon EC2. When the mobile app runs, it communicates with the custom identity broker. The broker application verifies that the users are authenticated and then calls an AWS STS action to get temporary security credentials. The application can call either `AssumeRole` or `GetFederationToken` to obtain the temporary credentials, depending on how Dave wants to manage the policies for users and when the temporary credentials should expire. (For more information about the differences between these APIs, see Ways to Get Temporary Security Credentials (p. 2) and Permissions in Temporary Security Credentials for Federated Users (p. 32).)

The AWS STS API returns temporary security credentials consisting of an AWS access key ID, a secret access key, and a session token. The custom identity broker application returns these temporary security credentials to the mobile app. 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.



The following details enable this scenario:

- The identity broker application has available a set of long-term AWS security credentials that are associated with an IAM user that it can use to call the `AssumeRole` or `GetFederationToken` action. The identity broker application runs in a server environment, where the credentials for the IAM user are not accessible to the apps running on mobile devices.
- The identity broker application (via the security credentials it uses to make the call) has permission to access the AWS STS API to create temporary security credentials.
- The identity broker application is able to verify that the mobile app users are authenticated.

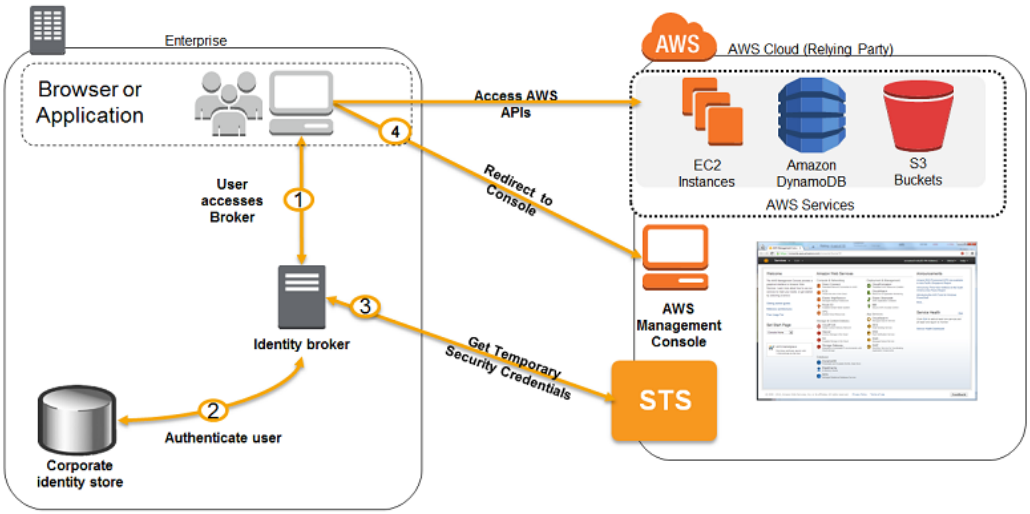
To see a sample application similar to the application described in this scenario, go to Authenticating Users of AWS Mobile Applications with a Token Vending Machine at *AWS Articles & Tutorials*. For information about creating temporary security credentials, see Creating Temporary Security Credentials (p. 11).

Using Your Organization's Authentication System to Grant Access to AWS Resources

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 APIs, see Ways to Get Temporary Security Credentials (p. 2) and Permissions in Temporary Security Credentials for Federated Users (p. 32).) 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 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.



In this scenario:

- The identity broker application has permissions to access the AWS 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).

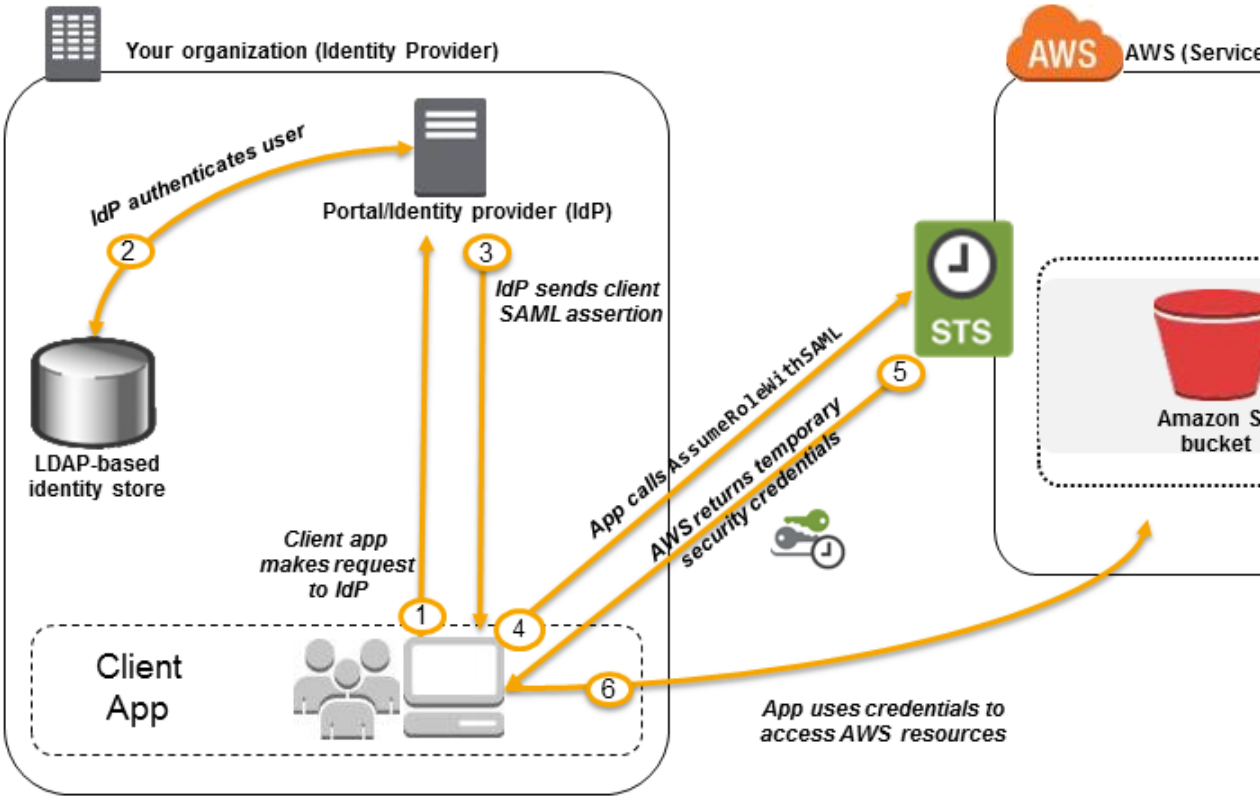
To see a sample application similar to the identity broker application described in this scenario, go to Identity Federation Sample Application for an Active Directory Use Case at *AWS Sample Code & Libraries*. For information about creating temporary security credentials, see Creating Temporary Security Credentials (p. 11). For more information about federated users getting access to the AWS Management Console, see Giving Federated Users Direct Access to the AWS Management Console (p. 41).

Using Your Organization's Authentication System and SAML to Grant Access to AWS Resources

Clarisse is an administrator who works at an organization that uses a SAML 2.0-compliant identity provider (IdP) to implement identity federation. In Clarisse's organization, an identity provider can authenticate users against an internal identity store. The IdP can then produce SAML assertions that indicate who the user is and that include information that can be used for authorization decisions by a service provider. Clarisse configures her organization's identity provider, and configures AWS as a service provider that can trust authentication responses from her organization.

Clarisse writes an application that runs on user's computers in her organization and that stores objects in Amazon S3 buckets. The application can get user information and ask the identity provider for an authentication response (assertion). It can then call the AWS STS `AssumeRoleWithSAML` API, passing the assertion and additional information.

The API returns temporary security credentials, which Clarisse's application can use to make calls to AWS directly. The following figure illustrates this scenario.



This scenario is similar to the previous one (using an organization's existing identity system). However, because Clarisse's organization uses SAML 2.0, the SAML identity provider in her organization can do much of the work associated with verifying a user's identity and generating information that can be passed to service providers, using an open standard for security information. AWS supports SAML 2.0 for establishing trust and for mapping assertions to IAM roles that determine a user's permissions.

For more information, see Creating Temporary Security Credentials for SAML Federation (p. 19).

Web-Based Single Sign-On (SSO)

Some users who are in an administrative group in your organization need to be able to go to the AWS Management Console and administer Amazon EC2 instances. You don't want to create new IAM users for each administrative user and make those users sign in again to the AWS Management Console.

Instead, you can create a SSO experience for your users. They can go to a portal in your organization, where they're already signed in. From the portal they can choose an option to go to the AWS Management Console. If they're authorized for AWS access, they're redirected to the AWS Management Console without having to sign in again. Behind the scenes, authentication information about the users is exchanged for temporary security credentials that are associated with an IAM role that determines what the users are allowed to do in AWS.

You can configure SSO in these ways:

- If your organization has an identity system that integrates SAML 2.0 (Security Assertion Markup Language 2.0), you can set things up in your organization and in IAM so that users can seamlessly be

redirected to the AWS Management Console. For details, see Giving Console Access Using SAML (p. 41).

- For other scenarios, you can write code that creates a URL that includes identity information and that you can distribute to users and that gives them secure and direct access to the AWS Management Console. For details, see Giving Console Access by Creating a URL (p. 46).

Delegating API Access

With IAM roles, you can delegate API access to AWS services (including third-party services) so that they can access your organization's AWS resources with temporary security credentials. Authorized IAM users or services can then use the temporary security credentials to access the resources that are defined in the role.

For more information, see Delegating API Access by Using Roles in *Using IAM*.

Cross-Account API Access

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

For more information, see Enabling Cross-Account API Access in *Using IAM*.

Creating Temporary Security Credentials

Topics

This topic describes how to use the AWS Security Token Service (AWS STS) API to create temporary security credentials. For information about using one of the supported SDKs to create temporary security credentials, see Ways to Get Temporary Security Credentials (p. 2).

The method you use to create the temporary security credentials depends on how you intend to use them:

- To get temporary security credentials after the user has authenticated using Login for Amazon, Facebook, or Google, an application calls `AssumeRoleWithWebIdentity`. The temporary security credentials are associated with an IAM role in which the trusted entity (the principal) is the identity provider (Amazon, Facebook, or Google), and where a condition tests the application ID that the provider assigns when the application is configured with that identity provider. Optionally, the application that calls `AssumeRoleWithWebIdentity` can pass an IAM policy that further restricts what the application using the temporary credentials is allowed to do in AWS.
- To get temporary security credentials in an organization that supports SAML 2.0 (Security Assertion Markup Language). In this scenario, your organization acts as a SAML-enabled identity provider and AWS acts as a service provider. An application in your organization calls the `AssumeRoleWithSAML` API using a SAML assertion, which is exchanged for temporary security credentials. The permissions granted for the temporary security credentials are defined in the IAM role that is assumed.
- To get temporary security credentials for federated users who are authenticated using a custom proxy application (for example, users can be authenticated against a corporate network identity system), the proxy application calls `GetFederationToken`. This call requires that the caller (the proxy application) use the security credentials of an existing IAM user. The call to `GetFederationToken` must include a policy that limits what the temporary security credentials permit the app to do; the final permissions are the intersection of the policy of the calling IAM user and the policy that's passed in the call.

- To get temporary security credentials for an IAM user in another AWS account, or for an AWS service (like Amazon EC2), the application or service calls `AssumeRole`. The role that is assumed by this API action must list the user or service as a principal. The assuming entity cannot be an AWS root account.
- To get temporary security credentials for their own use, IAM users call the `GetSessionToken`. Users do not need explicit permission to use `GetSessionToken`; it is available to all IAM users.
- To get temporary security credentials to support single sign-on (SSO) that allow users from your organization who are already signed into your network to access the AWS Management Console without having to have an IAM user identity and without having to sign in again in AWS. For details, see Giving Federated Users Direct Access to the AWS Management Console (p. 41).

For more general information about controlling user permissions, see Managing IAM Policies. The AWS STS API is described in detail in the AWS Security Token Service API Reference.

Important
Once you get temporary security credentials, you cannot revoke them. However, if you must disable temporary security credentials before they expire, you can modify or disable the permissions of the IAM user or role that the permissions are associated with. Changes to these permissions are applicable as soon as they have been propagated to all AWS regions, even if the temporary credentials have not expired. Because you cannot limit the permissions of a root user, we strongly recommend that you do not use your root account credentials to create temporary security credentials. (The `AssumeRole` action denies access to any request that is made using root credentials.) For more information, see Disabling Permissions Granted Through Temporary Security Credentials (p. 35).

Creating Temporary Security Credentials for Mobile Apps Using Identity Providers

Imagine that you have a mobile app that needs access to AWS resources. (Or it might be a web app that uses client script; the concepts presented here are the same.) The app might be a game that runs on a phone and stores player and score information in an Amazon S3 bucket or an Amazon DynamoDB table. Because the app needs to be able to distinguish individual users, users cannot be anonymous.

Most requests to AWS services must be signed, which requires an access key ID and secret access key. However, for apps that are downloaded to a user's device or computer, we recommend that you do not distribute long-term AWS security credentials such as those for an AWS account or for an IAM user.

Instead, you want to build the app such that it requests temporary security credentials using *web identity federation*. This lets you create an app that authenticates users—that is, lets users sign in—using these identity providers:

- Login with Amazon
- Facebook
- Google

Using any of these providers can simplify the development and management of your app. Instead of providing custom sign-in logic and having to manage user login information (either in a custom system or as IAM users), your app can rely on well-known and secure sign-in protocols that many users already have access to. Because you can trade a token from the identity provider for temporary security credentials, you don't have to distribute any credentials with the app, and you don't need to manage the process of rotating the credentials.

Topics

Process for Using Web Identity Federation for Mobile Apps

To use one of the supported identity providers and use web identity federation to get temporary security credentials, you follow the steps outlined here.

> **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.

1. Sign up as a developer with the identity provider. You also configure your app with the provider; when you do, the provider gives you an ID that's unique to your app. (Different providers use different terminology for this process. We're using the term *configure* for the process of identifying your app with the provider.) Each provider gives you an app ID that's unique to that provider, so if you configure the same app with multiple providers, your app will have multiple app IDs. You can configure multiple apps with each provider.

   The following external links provide information about using one of the identity providers:

   - Login with Amazon Developer Center
   - Registration on the Facebook site.
   - Using OAuth 2.0 to Access Google APIs on the Google site
2. In AWS, create one or more IAM roles. For each role, define who can assume the role (the trust policy or trust relationship) and what permissions the app's users will have (the access policy).

   Create one role for each identity provider for each app. For example, you might create a role that can be assumed by an app where the user signed in using Login with Amazon, a second role for the same app where the user has signed in using Facebook, and a third role for the app where users sign in using Google. For the trust relationship, specify the identity provider (like Amazon.com) as the federated principal (the trusted entity), and include a condition that matches the app's ID. Examples of the roles for different providers are shown later in this topic.
3. In your application, authenticate your users using Login with Amazon, Facebook, or Google. To do this, call the identity provider using an interface that they provide. For example, you might call an API and pass the user's credentials and possibly other information that the provider requires. The exact way in which you authenticate the user depends on the provider and on what platform your app is running. Typically, if the user is not already signed in, the identity provider takes care of displaying a sign-in page for that provider. After the identity provider authenticates the user, the provider returns a token to your app.
4. In your app, make an *unsigned* call to the `AssumeRoleWithWebIdentity` action to request temporary security credentials. In the request, you pass the identity provider's token and specify the ARN for the IAM role that you created for that identity provider. AWS verifies that the token is trusted and valid. If so, AWS STS returns temporary security credentials to your app that have the permissions derived from the role you named in the request. The response also includes metadata about the user from the identity provider, such as the unique user ID that the identity provider assigned to the user.
5. Using the temporary security credentials you get in the `AssumeRoleWithWebIdentity` response, your app makes signed requests to AWS APIs. The user ID information from the identity provider can

be used to distinguish users in the app—for example, you can put objects into Amazon S3 folders that include the user ID as prefixes. This allows you to create access control policies that lock that folder down so only the user with that ID can access it. For more information, see Identifying Providers, Apps, and Users with Web Identity Federation (p. 17) later in this topic.

6. Your app caches 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 identity provider and how they manage their tokens, you might have to refresh the provider's token before you make a new call to `AssumeRoleWithWebIdentity`, since the provider's tokens also usually expire after a fixed time. (If you're using the AWS SDK for iOS or the AWS SDK for Android, you can use the AmazonSTSCredentialsProvider action, which manages the AWS STS credentials, including refreshing them as required.)

Invoking the Identity Provider to Authenticate the User

In your app, when a user signs in, you invoke the authentication process for the identity providers you configured the app with. The specifics of how you do this vary both according to which identity provider you're using (Login with Amazon, Facebook, or Google) and what platform your app is running on. For example, an Android app can use a different way to authenticate than an iOS app or a JavaScript-based web app.

In general, the authentication process returns a token to the app that represents the authenticated user. You might also get back additional information about the user, depending on what the provider exposes and what information the user is willing to share. You can use this information in your app.

Creating a Role to Allow AWS Access for the Mobile App

In order to allow the mobile app to access resources, you must create one or more IAM roles that the app can assume. As with any role, a role for the mobile app contains two policies. One is the trust policy that specifies who can assume the role (the trusted entity, or principal). The other policy (the access policy) specifies the actual AWS actions and resources that the mobile app is allowed or denied access to, and is similar to user or resource policies.

The trust policy must grant an `Allow` effect for the `sts:AssumeRoleWithWebIdentity` action. In this role, you use two values that let you make sure that the role can be assumed only by your application:

- For the `Principal` element, you use the string `{"Federated":`*`providerUrl`*`}`. The following are acceptable ways to specify the principal:

  ```
  "Principal":{"Federated":"www.amazon.com"}
  ```

  ```
  "Principal":{"Federated":"graph.facebook.com"}
  ```

  ```
  "Principal":{"Federated":"accounts.google.com"}
  ```

- In the `Condition` element, you use a `StringEquals` condition to test that the app ID in the request matches the app ID that you got when you configured the app with the identity provider. This ensures the request is coming from your app. In the policy, you can test the app ID you have against the following policy variables:

  ```
  www.amazon.com:app_id
  ```

```
graph.facebook.com:app_id

accounts.google.com:aud
```

Notice that the values you use for the principal in the role are specific to an identity provider. A role can specify only one principal. Therefore, if the mobile app allows users to sign in using more than one identity provider, you must create a role for each of the identity providers.

> **Note**
> Because the policy for the trusted entity uses policy variables that represent the provider and the app ID, you must set the `Version` element to `2012-10-17`.

You can use the IAM console to create a role for web identity federation. The console lets you choose **Roles for Web Identity Provider Access** for the role type, and then walks you through the process of configuring the principal and of creating a condition that tests for the app ID. For more information, see Creating a Role in *Using IAM*.

The following example shows a trust policy for a role that the mobile app could assume if the user has signed in using Login with Amazon. In the example, *amzn1.application-oa2-123456* is assumed to be the app ID that Amazon assigned when you configured the app using Login with Amazon.

```
{
  "Version":"2012-10-17",
  "Id":"RoleForLoginWithAmazon",
  "Statement":[{
      "Principal":{"Federated":"www.amazon.com"},
      "Effect":"Allow",
      "Action":"sts:AssumeRoleWithWebIdentity",
      "Condition": {
          "StringEquals":
              {"www.amazon.com:app_id":"amzn1.application-oa2-123456"}
       }
   }]
}
```

The following example shows a policy for a role that the mobile app could assume if the user has signed in using Facebook. *111222333444555* is assumed to be the app ID assigned by Facebook.

```
{
  "Version":"2012-10-17",
  "Id":"RoleForFacebook",
  "Statement":[{
      "Principal":{"Federated":"graph.facebook.com"},
      "Effect":"Allow",
      "Action":"sts:AssumeRoleWithWebIdentity",
      "Condition": {
          "StringEquals":
              {"graph.facebook.com:app_id":"111222333444555"}
       }
   }]
}
```

The following example shows a policy for a role that the mobile app could assume if the user has signed in using Google. *111222333444555666777* is assumed to be the app ID assigned by Google.

```
{
   "Version":"2012-10-17",
   "Id":"RoleForGoogle",
   "Statement":[{
       "Principal":{"Federated":"accounts.google.com"},
       "Effect":"Allow",
       "Action":"sts:AssumeRoleWithWebIdentity",
       "Condition": {
           "StringEquals":
               {"accounts.google.com:aud":"111222333444555666777"}
        }
    }]
}
```

Getting Temporary Credentials

To get temporary credentials that your app can use to make calls to AWS APIs, you call the `AssumeRoleWithWebIdentity` action of the AWS Security Token Service. This is an unsigned call, meaning that the app does not have 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 ARN of the role that the app should assume, as described in the preceding section. As noted, if your app supports multiple ways for users to sign in, you will have defined multiple roles, one per identity provider. The call to `AssumeRoleWithWebIdentity` should include the ARN of the role that's specific to the provider through which the user signed in.
- The token that the app got from the identity provider after the app authenticated the user.
- The duration, which specifies how long the temporary security credentials are good for. The maximum (and the default) is 1 hour (3600 seconds). You need to pass this value only if you want the temporary credentials to expire before 1 hour. The minimum duration for the credentials is 15 minutes (900 seconds).
- A role session name, which is a string value that can be used to identify the session.
- Optionally, a policy (in JSON format). This policy is combined with the policy associated with the role. This lets you further restrict the access permissions that will be associated with the temporary credentials, beyond the restrictions already established by the role access policy. Note that this policy cannot be used to elevate privileges beyond what the assumed role is allowed to access.

  Note
  Because a call to `AssumeRoleWithWebIdentity` is not signed, you should only include this optional policy if the request is not being transmitted through an untrusted intermediary.

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 using long-term security credentials, except that you must include the session token, which lets AWS verify that the temporary security credentials are valid.

Your app should cache the credentials. As noted, by default the credentials expire after an hour. If you are not using the AmazonSTSCredentialsProvider action in the AWS SDK, it's up to your app to call `AssumeRoleWithWebIdentity` again to get a new set of temporary security credentials before the existing set expires.

Identifying Providers, Apps, and 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 identity provider. For example, your mobile app that's using web identity federation might keep information in Amazon S3 using a structure like this:

```
myBucket/app1/user1
myBucket/app1/user2
myBucket/app1/user3
...
myBucket/app2/user1
myBucket/app2/user2
myBucket/app2/user3
...
```

You might also want to additionally distinguish these paths by provider. In that case, the structure might look like the following (only two providers are listed to save space):

```
myBucket/Amazon/app1/user1
myBucket/Amazon/app1/user2
myBucket/Amazon/app1/user3
...
myBucket/Amazon/app2/user1
myBucket/Amazon/app2/user2
myBucket/Amazon/app2/user3

myBucket/Facebook/app1/user1
myBucket/Facebook/app1/user2
myBucket/Facebook/app1/user3
...
myBucket/Facebook/app2/user1
myBucket/Facebook/app2/user2
myBucket/Facebook/app2/user3
...
```

For these structures, `app1` and `app2` represent different apps, such as different games, and each of the app's users 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 `Amazon` and `Facebook`.

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 keys that reference the user ID that is returned from the provider:

- `www.amazon.com:user_id`
- `graph.facebook.com:id`
- `accounts.google.com:sub`

The following example shows an access policy that grants access to a bucket in Amazon S3 whose prefix matches this:

`myBucket/Amazon/mynumbersgame/`*`user1`*

The example assumes that the user has signed in using Login with Amazon, and that the user is using the app to which you've given the friendly name `mynumbersgame`. You would create similar policies for users who have signed in using Facebook and Google; those policies would use a different provider name as part of the path and would use different app IDs.

```
{
 "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}/*"

   ]
  }
 ]
}
```

Additional Resources for Web Identity Federation

The following resources can help you learn more about web identity federation:

- 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 in order to access an Amazon S3 bucket.

- The AWS SDK for iOS and the AWS SDK for Android contain sample apps. These apps include code that shows how to invoke the identity providers, and then how to use the information from these providers to get and use temporary security credentials.
- The article Web Identity Federation with Mobile Applications discusses web identity federation and shows an example of how to use web identity federation to get access to content in Amazon S3.

Creating Temporary Security Credentials for SAML Federation

AWS supports identity federation using the SAML 2.0 (Security Assertion Markup Language 2.0), an open standard used by many identity providers. This feature enables federated single sign-on (SSO), which lets users log into the AWS Management Console or make programmatic calls to AWS APIs. Using SAML can simplify the process of configuring federation with AWS, because you can use identity provider software instead of writing code.

AWS STS and IAM support these use cases:

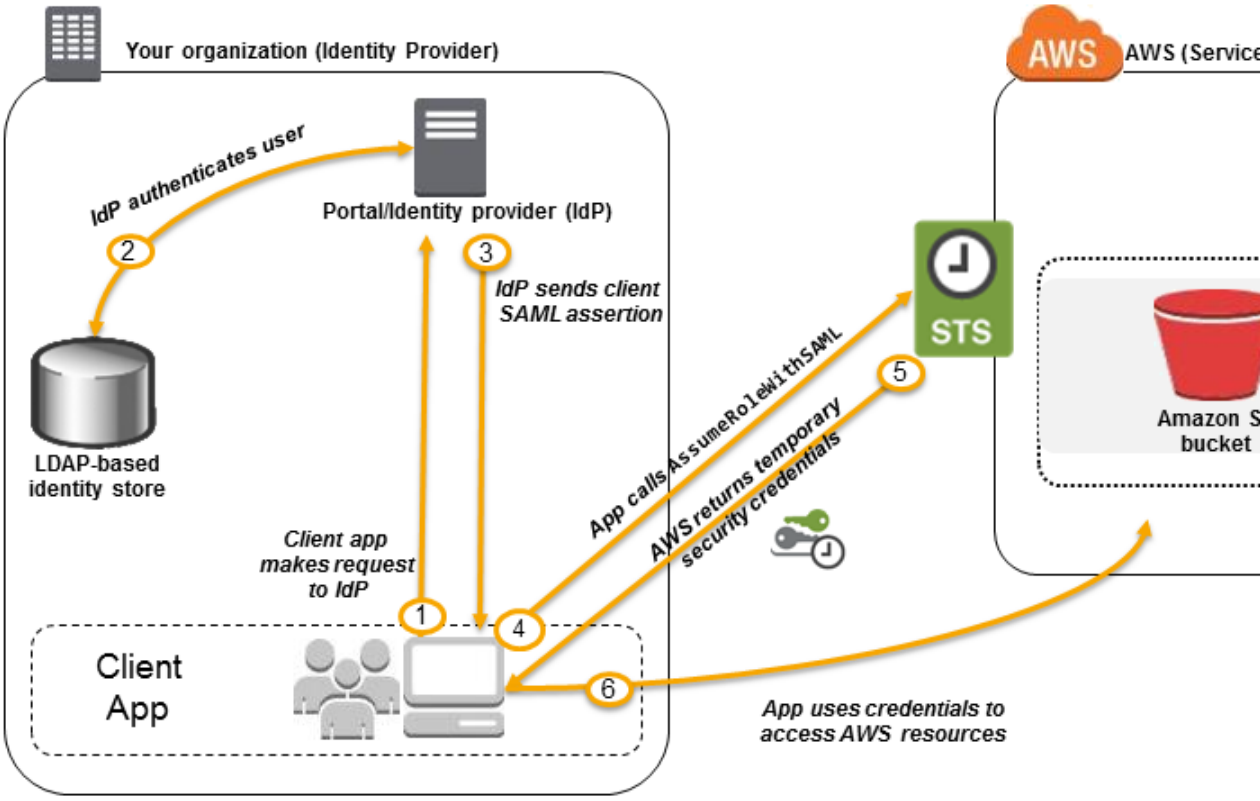
- Web-based single sign-on (WebSSO) to the AWS Management Console from your organization. Users can sign in to a portal in your organization, select an option to go to AWS, and be redirected to the console without having to provide additional sign-in information. For more information, see Giving Console Access Using SAML (p. 41).
- Federated access to allow a user or application in your organization to call AWS APIs using temporary security credentials. In effect, you can use a SAML assertion (as part of the authentication response) generated in your organization to get temporary security credentials. This scenario is similar to other federation scenarios supported by AWS STS and IAM, like those described in Creating Temporary Security Credentials to Enable Access for Federated Users (p. 22) and Creating Temporary Security Credentials for Mobile Apps Using Identity Providers (p. 12). However, SAML-based identity providers in your organization handle many of the details at run time for performing authentication and authorization checking.

Topics

Configuring SAML-Based Federation for API Access

Imagine that in your organization, you want to provide a way for users 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 Amazon S3 bucket. Users don't have direct access to AWS. Instead, the application gets the user's information from your organization's identity store (such as an LDAP directory) and gets a SAML assertion that includes authentication and authorization information about that user. The application can then use that assertion to make a call to the AWS STS `AssumeRoleWithSAML` API to get temporary security credentials and use those credentials to access a folder in the Amazon S3 bucket that's specific to the user.

The following diagram describes the flow.



Process for Using SAML-Based Federation

Inside your organization, you have an identity provider (IdP) that supports SAML 2.0, like Windows Active Directory Federation Services, Shibboleth, etc.

1. In your organization's IdP you register AWS as a service provider (SP) using the SAML metadata document that from the following URL:

   `https://signin.aws.amazon.com/static/saml-metadata.xml`
2. Using your organization's IdP, you generate an XML metadata document that includes 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 new SAML provider, which is an entity in IAM. As part of this process, you upload the SAML metadata document that was produced by the IdP in your organization.
4. In IAM, you create one or more IAM roles. In the role's trust policy, you set the SAML provider as a principal, which establishes a trust relationship between your organization and AWS. The role's access (permission) policy establishes what users from your organization will be allowed to do in AWS.
5. In your organization's IdP, you create set assertions and map the IAM role to users or groups in your organization who will be allowed to have the permissions specified in the role. 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 example of an S3 folder for users, it's possible that all users will map to the same role that provides Amazon S3 permissions.
6. In the application that you're creating, you call the AWS STS `AssumeRoleWithSAML` API, passing it the ARN of the SAML provider in IAM, the ARN of the role to assume, and a 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.

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.

Creating a Role to Allow AWS Access from Your Organization

The role or roles that you create in IAM define what federated users from your organization will be 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 to allow only certain users to sign in, based on SAML attributes. For example, you can specify that only users whose SAML affiliation is `staff` (as asserted by https://openidp.feide.no) will be allowed to sign in.

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Principal": {"AWS": "arn:aws:sts::account-number-without-hyphens:saml-
provider/ExampleOrgSSOProvider"},
    "Action": "sts:AssumeRoleWithSAML",
    "Condition": {
      "StringEquals": {
        "SAML:aud": "https://signin.aws.amazon.com/saml",
        "SAML:iss": "https://openidp.feide.no"
      },
      "ForAllValues:StringLike": {"SAML:eduPersonAffiliation": ["staff"]}
    }
  }]
}
```

For the access (permissions) policy in the role, you specify permissions as you would for any role. For example, if users from your organization will be allowed to administer Amazon EC2 instances, you explicitly allow Amazon EC2 actions in the permissions policy, such as those in the **Amazon EC2 Full Access** policy template.

Identifying Users for 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 who have authenticated using an identity provider. 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 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 then.

To write policies that restrict access so that users can access only their own folders, the information that you use to identify users has to be available in conditions keys for policies. The following keys are available for SAML-based federation for use in IAM policies. The values represented by these keys in turn represent how to create unique user identifiers for resources like Amazon S3 folders.

- `SAML:namequalifier`. This key contains a hash value that represents the combination of the `SAML:doc` and `SAML:iss` values. It is used as a namespace qualifier; the combination of `SAML:namequalifier` and `SAML:sub` uniquely identifies a user. 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(SAML:doc + SAML:iss))
  ```

- `SAML:sub` (string). This is the subject of the claim, which includes a value that uniquely identifies an individual user within an organization (for example, `_cbb88bf52c2510eabe00c1642d4643f41430fe25e3`).
- `SAML:sub_type` (string). This key can be "persistent" or "transient". A value of "Persistent" indicates that the value in `SAML:sub` is the same for a user between sessions. If the value is "Transient", the user has a different `SAML:sub_type` value for each session.

The following example shows an access 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 identity user-specific folders.

```
{
  "Version": "2012-10-17",
  "Statement": [{
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObject"
      ],
      "Resource": [
         "arn:aws:s3:::exampleorgBucket/backup/${SAML:namequalifi
er}/${SAML:sub}",
         "arn:aws:s3:::exampleorgBucket/backup/${SAML:namequalifi
er}/${SAML:sub}/*"
      ],
      "Condition": {
         "StringEquals": {
            "SAML:sub_type":"persistent"
         }
      }
   }]
}
```

For more information about mapping assertions from the IdP to policy keys, see Configure Assertions for the SAML Authentication Response (p. 44).

Creating Temporary Security Credentials to Enable Access for Federated Users

To grant temporary access to a non-AWS user whose identity you can authenticate (a federated user), you can use the AWS STS `AssumeRole` or `GetFederationToken` API actions. If you use a SAML

identity provider (IdP) in your organization, you can use the `AssumeRoleWithSAML` API action. These actions are useful if you have users who already have identities in an identity store like Microsoft Active Directory. (If you can authenticate the user using an identity provider like Login with Amazon, Facebook, or Google, you can use the `AssumeRoleWithWebIdentity` action. For more information, see Creating Temporary Security Credentials for Mobile Apps Using Identity Providers (p. 12).)7

You might use temporary security credentials to enable single sign-on (SSO) so that users can use the AWS Management Console without having to sign in as IAM users. You might also use temporary security credentials to create applications in your company that access AWS resources.

AssumeRole

When you create temporary security credentials for a federated user, you specify a role Amazon Resource Name (ARN). You can optionally specify the duration and a scoped-down policy for the temporary security credentials. The duration can be between 15 minutes to 1 hour. By default, the duration is 1 hour. The `AssumeRole` API action returns temporary security credentials consisting of the security token, access key, secret key, and expiration.

> **Note**
> You must use IAM user credentials to call `AssumeRole`. You can't use AWS account credentials to call `AssumeRole`; access is denied.

You use `AssumeRole` if you want to manage permissions in AWS. To view a sample application that uses `AssumeRole`, go to AWS Management Console federation proxy sample use case in the *AWS Sample Code & Libraries*.

The following example shows a sample request and response using `AssumeRole`. In this example, the request includes the name for the session named Bob. The *`Policy`* parameter includes a JSON document that specifies that the resulting credentials have permissions to access only Amazon S3.

Example Request

```
https://sts.amazonaws.com/
?Version=2011-06-15
&Action=AssumeRole
&RoleSessionName=Bob
&RoleArn=arn:aws:iam::123456789012:role/demo
&Policy=%7B%22Version%22%3A%222012-10-17%22%2C%22State
ment%22%3A%5B%7B%22Sid%22%3A%20%22Stmt1%22%2C%22Effect%22%3A%20%22Al
low%22%2C%22Action%22%3A%20%22s3%3A*%22%2C%22Resource%22%3A%20%22*%22%7D%5D%7D
&DurationSeconds=3600
&ExternalId=123ABC
&AUTHPARAMS
```

> **Note**
> The policy value shown in the example above is the URL-encoded version of the following policy:
> `{"Version":"2012-10-17","Statement":[{"Sid": "Stmt1","Effect": "Allow","Action": "s3:*","Resource": "*"}]}`

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 Response

```
<AssumeRoleResponse xmlns="https://sts.amazonaws.com/doc/2011-06-15/">
<AssumeRoleResult>
```

```
<Credentials>
  <SessionToken>
   AQoDYXdzEPT//////////wEXAMPLEtc764bNrC9SAPBSM22wDOk4x4HIZ8j4FZTwdQW
   LWsKWHGBuFqwAeMicRXmxfpSPfIeoIYRqTflfKD8YUuwthAx7mSEI/qkPpKPi/kMcGd
   QrmGdeehM4IC1NtBmUpp2wUE8phUZampKsburEDy0KPkyQDYwT7WZ0wq5VSXDvp75YU
   9HFvlRd8Tx6q6fE8YQcHNVXAkiY9q6d+xo0rKwT38xVqr7ZD0u0iPPkUL64lIZbqBAz
   +scqKmlzm8FDrypNC9Yjc8fPOLn9FX9KSYvKTr4rvx3iSIlTJabIQwj2ICCR/oLxBA==
  </SessionToken>
  <SecretAccessKey>
   wJalrXUtnFEMI/K7MDENG/bPxRfiCYzEXAMPLEKEY
  </SecretAccessKey>
  <Expiration>2011-07-15T23:28:33.359Z</Expiration>
  <AccessKeyId>AKIAIOSFODNN7EXAMPLE</AccessKeyId>
</Credentials>
<AssumedRoleUser>
  <Arn>arn:aws:sts::123456789012:assumed-role/demo/Bob</Arn>
  <AssumedRoleId>ARO123EXAMPLE123:Bob</AssumedRoleId>
</AssumedRoleUser>
<PackedPolicySize>6</PackedPolicySize>
</AssumeRoleResult>
<ResponseMetadata>
<RequestId>c6104cbe-af31-11e0-8154-cbc7ccf896c7</RequestId>
</ResponseMetadata>
</AssumeRoleResponse>
```

Note
`AssumeRole` stores the policy in a packed format. `AssumeRole` returns the size so you can adjust the calling parameters. For more information about the size constraints on the policy, go to AssumeRole in the *AWS Security Token Service API Reference*.

You can also grant permissions at the resource level. For example, if your AWS account number is 111122223333, and you have an Amazon S3 bucket that you want to allow Bob to access even though his temporary security credentials don't include a policy for the bucket, you would need to ensure that the bucket has a policy with an ARN that matches Bob's ARN: `arn:aws:sts::123456789012:assumed-role/demo/Bob`.

GetFederationToken

When you make a request to get temporary security credentials for a federated user, you make the request using the credentials of a specific user identity (an IAM user) and request a maximum duration for the temporary security credentials to remain valid. Credentials created by IAM users are valid for the specified duration, between 15 minutes and 36 hours; credentials created using account credentials have a maximum of one hour. The permissions available with the temporary security credentials are determined by an IAM policy that you pass when you call `GetFederationToken`.

The `GetFederationToken` call returns temporary security credentials consisting of the security token, access key, secret key, and expiration. You can use `GetFederationToken` if you want to manage permissions inside your organization (for example, using the proxy application to assign permissions). To view a sample application that uses `GetFederationToken`, go to Identity Federation Sample Application for an Active Directory Use Case in the *AWS Sample Code & Libraries*.

The following example shows a sample request and response using `GetFederationToken`. In this example, the request includes the name for a federated user named Jean. The *`Policy`* parameter includes a JSON document that specifies that the resulting credentials have permissions to access only Amazon S3. 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 Request

```
https://sts.amazonaws.com/
?Version=2011-06-15
&Action=GetFederationToken
&Name=Jean
&Policy=%7B%22Version%22%3A%222012-10-17%22%2C%22State
ment%22%3A%5B%7B%22Sid%22%3A%22Stmt1%22%2C%22Effect%22%3A%22Allow%22%2C%22Ac
tion%22%3A%22s3%3A*%22%2C%22Resource%22%3A%22*%22%7D%5D%7D
&DurationSeconds=3600
&AUTHPARAMS
```

Note
The policy value shown in the example above is the URL-encoded version of this policy: {"Version":"2012-10-17","Statement":[{"Sid":"Stmt1","Effect":"Allow","Action":"s3:*","Resource":"*"}]}.

Example Response

```
<GetFederationTokenResponse xmlns="https://sts.amazonaws.com/doc/2011-06-15/">
<GetFederationTokenResult>
<Credentials>
  <SessionToken>
   AQoDYXdzEPT//////////wEXAMPLEtc764bNrC9SAPBSM22wDOk4x4HIZ8j4FZTwdQW
   LWsKWHGBuFqwAeMicRXmxfpSPfIeoIYRqTflfKD8YUuwthAx7mSEI/qkPpKPi/kMcGd
   QrmGdeehM4IC1NtBmUpp2wUE8phUZampKsburEDy0KPkyQDYwT7WZ0wq5VSXDvp75YU
   9HFvlRd8Tx6q6fE8YQcHNVXAkiY9q6d+xo0rKwT38xVqr7ZD0u0iPPkUL64lIZbqBAz
   +scqKmlzm8FDrypNC9Yjc8fPOLn9FX9KSYvKTr4rvx3iSIlTJabIQwj2ICCEXAMPLE==
  </SessionToken>
  <SecretAccessKey>
  wJalrXUtnFEMI/K7MDENG/bPxRfiCYzEXAMPLEKEY
  </SecretAccessKey>
  <Expiration>2011-07-15T23:28:33.359Z</Expiration>
  <AccessKeyId>AKIAIOSFODNN7EXAMPLE;</AccessKeyId>
</Credentials>
<FederatedUser>
  <Arn>arn:aws:sts::123456789012:federated-user/Jean</Arn>
  <FederatedUserId>123456789012:Jean</FederatedUserId>
</FederatedUser>
<PackedPolicySize>6</PackedPolicySize>
</GetFederationTokenResult>
<ResponseMetadata>
<RequestId>c6104cbe-af31-11e0-8154-cbc7ccf896c7</RequestId>
</ResponseMetadata>
</GetFederationTokenResponse>
```

Note
`GetFederationToken` stores the policy in a packed format. The action returns the size so you can adjust the calling parameters. For more information about size constraints on the policy, go to GetFederationToken in the *AWS Security Token Service API Reference*.

If you prefer to grant permissions at the resource level (for example, you attach a 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, if your AWS account number is 111122223333, and you have an Amazon S3 bucket that you want to allow Susan to access even though her temporary security credentials don't include a policy

for the bucket, 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`.

Related Topics

- Making Query Requests
- Controlling Permissions for Temporary Security Credentials (p. 32)
- Disabling Permissions Granted Through Temporary Security Credentials (p. 35)
- Overview of Policies

Creating Temporary Security Credentials for Delegating API Access

You can delegate access to your AWS resources by using IAM roles. IAM roles allow you to establish trusted relationships with other AWS accounts (trusted entities). After a relationship has been established, an IAM user or an application from the trusted entity can use the AWS Security Token Service (AWS STS) `AssumeRole` action to obtain temporary security credentials that can be used to access AWS resources in your account.

The temporary security credentials contain an access key ID, a secret access key, and a security token. With the temporary security credentials, callers are granted the permissions that are defined in the role. However, callers can scope down the permissions derived from the assumed role by passing a policy in the `AssumeRole` call. (The passed policy can never escalate privileges beyond the permissions that are defined in the role.) This optional policy is useful if multiple callers might call the same role, but each caller requires different permissions. For example, different callers might require permissions to different Amazon S3 buckets, but creating a role for each bucket might be tedious. Instead, you can create one role that includes permissions for multiple buckets. The caller can then pass a policy that denies access to the buckets that a specific user doesn't need access to.

You can specify the duration of the temporary security credentials to be from 15 minutes to one hour. By default, the credentials are valid for one hour.

Requirements for assuming a role

To assume a role, the caller must meet the following requirements:

- The caller must have permission to call `AssumeRole` for the specific role.
- The role defines the caller's AWS account ID as a trusted entity.
- The caller must use IAM user credentials to assume a role.
- If the role has an external ID defined, the caller must pass that external ID when calling `AssumeRole`. For more information, see About the External ID (p. 26).

About the External ID

An *external ID* is an optional piece of information that you can test in an IAM role policy to provide additional control over who can assume the role. When a role policy includes an external ID, anyone who wants to assume the role must not only be specified as a principal in the role, but must also include the external ID.

The external ID is particularly useful when you delegate access to your AWS account to a third party—for example, when a company has multiple customers and manages AWS resources on behalf of those customers. Using the external ID as part of a request to assume a customer's role helps ensure that the

requester accesses the correct AWS account. (This association helps prevent a form of privilege escalation known as the "Confused Deputy" problem.)

A typical use is when a third-party company performs AWS tasks for customers, which works like this:

- An AWS customer, Bob, has an AWS account. Bob hires Example Corp, a third-party company, to administer his AWS resources. Example Corp also has an AWS account, and Example Corp manages AWS resources for other customers who have their own AWS accounts.
- Example Corp creates a unique identifier for Bob. Example Corp gives Bob his unique identifier and Example Corp's AWS account number. Bob needs this information to create an IAM role (next step).
- Bob signs into AWS and creates an IAM role that will give Example Corp access to his resources. Like any IAM role, the role has two policies, a permissions policy and a trust policy. The permissions policy for the role specifies what the role allows someone to do. For example, Bob might specify that the role allows someone to manage only his Amazon EC2 and Amazon RDS resources, but not his IAM users or groups.

  The role's trust policy specifies who can assume the role. In this scenario, the policy specifies the AWS account number of Example Corp as the principal (that is, as the entity that's allowed to assume the role). In addition, the trust policy includes a Condition element that tests the unique ID that Example Corp assigned to Bob when he hired the company. The trust policy might look like this:

```
{
"Version": "2012-10-17",
"Statement": [
{
  "Sid": "",
  "Effect": "Allow"
  "Principal": {
 "AWS": "arn:aws:iam::Example-Corp-account-number:root"
  },
  "Action": "sts:AssumeRole",
  "Condition": {
 "StringEquals": {
   "sts:ExternalId": "ID-issued-to-Bob-by-Example-Corp"
 }
  }
}
]
}
```

- Bob takes note of the ARN of the role and sends it to Example Corp. The role ARN might look like this:

```
arn:aws:iam::Bob-account-number:role/RoleForExampleCorp
```

- When Example Corp needs to administer Bob's 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.

  The request is authorized only if the role ARN and the external ID are correct, and if the request comes from someone using Example Corp's AWS account. If the request succeeds, it returns temporary security credentials that Example Corp can use to access the AWS resources that Bob's role allows.

Related Topics

The following information can show you what permissions are required to work with roles, how you can set role permissions, how to define trusted entities, and how to assume a role:

- Granting Applications that Run on an Amazon EC2 Access to AWS Resources. This information provides an overview of how applications that run on an instance can use role credentials to access AWS resources, a method that doesn't require anyone to share credentials on Amazon EC2 instances.
- Enabling Cross-Account Access. This information shows how IAM users can access AWS resources in another AWS account by using roles.

Granting an IAM Group Permission to Create Temporary Security Credentials

By default, IAM users do not have permission to create temporary security credentials for federated users and roles. However, IAM users can call `GetSessionToken` by default. To grant an IAM group permission to create temporary security credentials for federated users or roles, you should attach a policy to the IAM group that the IAM users belong to that grants one or both of the following privileges:

- For federated users, access to AWS STS `GetFederationToken`.
- For IAM roles, access to AWS STS `AssumeRole`.

Example A policy that grants permission to create temporary security credentials for a federated user

The following example shows a policy that grants permission to access `GetFederationToken`.

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

> **Important**
> When you give an IAM user permission to create temporary security credentials for federated users, you should be aware that this enables the IAM user to delegate his or her own permissions. For more information about delegating permissions across IAM users and AWS accounts, see Enabling Cross-Account Access. For more information about controlling permissions in temporary security credentials, see Controlling Permissions for Temporary Security Credentials (p. 32).

Example Example of granting a user limited permission to create temporary security credentials for federated users

When you let an IAM user call `GetFederationToken` to create temporary security credentials for federated users, it is a best practice to restrict as much as practical the permissions that the IAM user is allowed to 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*"]
}]
}
```

Example Example of a policy granting permission to assume a role

The following example shows a policy that grants permission to call `AssumeRole` for the `UpdateApp` role in AWS account `123123123123`.

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

Related Topics

- Managing IAM Policies
- Identifiers for IAM Entities
- Roles

Creating Temporary Security Credentials to Enable Access for IAM Users

IAM users can use the AWS Security Token Service `GetSessionToken` API action to create temporary security credentials for themselves. This enables access for IAM users or AWS accounts whose permissions are already defined. Because the credentials are temporary, they provide enhanced security when you have an IAM user who will be accessing your resources through a less secure environment, such as a mobile device or web browser.

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. For security reasons, a token for an AWS account's root identity is restricted to a duration of one hour.

`GetSessionToken` returns temporary security credentials consisting of a security token, an access key ID, and a secret access key. The following example shows a sample request and response using `GetSessionToken`. The response also includes the expiration time of the temporary security credentials.

Example Request

```
https://sts.amazonaws.com/
?Version=2011-06-15
&Action=GetSessionToken
&DurationSeconds=3600
&AUTHPARAMS
```

Example Response

```
<GetSessionTokenResponse xmlns="https://sts.amazonaws.com/doc/2011-06-15/">
<GetSessionTokenResult>
<Credentials>
  <SessionToken>
   AQoEXAMPLEH4aoAH0gNCAPyJxz4BlCFFxWNE1OPTgk5TthT+FvwqnKwRcOIfrRh3c/L
   To6UDdyJwOOvEVPvLXCrrrUtdnniCEXAMPLE/IvU1dYUg2RVAJBanLiHb4IgRmpRV3z
   rkuWJOgQs8IZZaIv2BXIa2R4OlgkBN9bkUDNCJiBeb/AXlzBBko7b15fjrBs2+cTQtp
   Z3CYWFXG8C5zqx37wnOE49mRl/+OtkIKGO7fAE
  </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>
```

Temporary Security Credentials for IAM Users with Multi-Factor Authentication (MFA)

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 so that the temporary security credentials can be used to access the MFA-protected API actions 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
```

Related Topics

- GetSessionToken in the *AWS Security Token Service API Reference*
- Using Multi-Factor Authentication (MFA) Devices with AWS in *Using IAM*
- Making Query Requests in *Using IAM*

Controlling Permissions for Temporary Security Credentials

Topics

AWS determines what permissions to associate with temporary security credentials at the time that the credentials are created. For example, the permissions for the temporary security credentials are bound to either the role that was assumed (`AssumeRole`, `AssumeRoleWithWebIdentity`, or `AssumeRoleWithSAML`) or to the IAM user that made the request (`GetFederationToken` or `GetSessionToken`). The temporary security credentials are not bound to a set of static permissions when the credentials are created. Instead, the effective permissions are evaluated when a request is made that uses the credentials, based on the current permissions of the associated IAM user or role that the temporary security credentials are bound to.

After temporary security credentials have been issued, they are valid through the expiration period and cannot be revoked. However, because the permissions for the temporary credentials are checked for each request, you can change the effective permissions for the temporary security credentials by editing (or deleting) the policy or policies that describe the permissions for the role or user. In effect, you can change the access rights for those credentials even after the credentials have been issued.

This section describes what you need to know about granting permissions in temporary security credentials, and how to update or disable permissions after temporary security credentials have been issued.

Permissions in Temporary Security Credentials for Federated Users

Calls to the `AssumeRole` action are made using the long-term security credentials of an IAM user. The call must specify the ARN of the role to assume. The IAM user whose credentials are used to make the call must as a minimum have `sts:AssumeRole` permissions, and must be listed as the principal in the role that is being assumed. By default, the role being assumed determines the permissions that are

granted to the temporary security credentials. The permissions of the IAM user that's used to make the `AssumeRole` API have no effect on the permissions granted to the temporary security credentials that are returned by the API. Optionally, the call can include a policy that further restricts the permissions of the temporary security credentials. The resulting credentials are based on the combination of the role's permissions and the passed permissions. (This means that the passed permissions can never escalate the permissions defined in the role.)

For web identity federation (p. 12), calls to `AssumeRoleWithWebIdentity` are made without any AWS credentials. As with `AssumeRole`, a parameter for the API call is the ARN of a role to assume. The role that is being assumed must list the web identity provider as a principal (for example, `"Principal":{"Federated":"www.amazon.com"}`). By default, as with `AssumeRole`, the permissions granted to the resulting temporary security credentials are determined by the role that is assumed. In addition, the call can include a policy that combines with the role's policy to determine the permissions granted to the temporary security credentials.

Similarly, for SAML-based federation (p. 19), calls to `AssumeRoleWithSAML` are made without any AWS credentials. The call includes a SAML assertion, the ARN of an SAML provider in IAM, and the ARN of the role to be assumed. (The trust policy of the role includes the SAML provider as the principal.) The permissions associated with the temporary security credentials are determined by the role that is assumed. The call can include a policy that combines with the role's access policy to further reduce the permissions associated with the credentials.

Calls to the `GetFederationToken` API action are made using the credentials of an IAM user. In addition, you pass a policy as a parameter to `GetFederationToken`. The permissions granted to the resulting temporary security credentials (that is, the permissions for the federated user) are the intersection of the permissions of the IAM user making the request with the permissions that are passed in the call.

> **Note**
> Passing a policy to the `GetFederationToken` action is optional. However, if you do not pass a policy, the resulting temporary security credentials that are returned have no effective permissions and requests made using those credentials are always denied. A typical approach for `GetFederationToken` is to create a policy for the calling IAM user that allows all actions on all resources that might possibly be invoked via the proxy application. Then when you call `GetFederationToken` to get temporary security credentials for a specific federated user, you pass an individualized policy that reduces the permissions to an appropriate level for that user.

In all cases, if the resource being accessed also has a policy attached to it (for example, an Amazon S3 bucket), that policy is evaluated along with the policies that are part of the call.

For more information about how permissions are evaluated, see Evaluation Logic in *Using IAM*.

Example of Permissions for a Federated User (GetFederationToken)

This section shows an example of how to use policies with the `GetFederationToken` API to control how temporary security credentials are created and how permissions are delegated. Suppose you want to grant read-only permissions to federated users so that they can access your Amazon S3 buckets. You have a proxy application that can issue temporary credentials, as described in Creating a Mobile App with Custom Authentication (p. 5). You create an IAM user named `Issuer` and set the IAM user's permissions using the following policy. The policy allows `Issuer` to call `GetFederationToken`, and it allows the user to get an item from `mybucket` in Amazon S3 as long as the item's name begins with `federated-user/`.

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

```
    {
      "Effect": "Allow",
      "Action": ["sts:GetFederationToken"],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": ["s3:GetObject"],
      "Resource": "arn:aws:s3:::mybucket/federated-user/*"
    }
  ]
}
```

To actually delegate permissions to a federated user, a call is made to `GetFederationToken` using the credentials of the user `Issuer`. The call includes a name that you assign to the federated user, the duration the token is valid, and a policy granting access to Amazon S3. Temporary security credentials that are returned by the call enable a federated user to read from the Amazon S3 bucket for as long as the temporary security credentials are valid.

Now suppose that you want to restrict the permissions for the federated user to a folder in the Amazon S3 bucket that matches the user's name. (For example, a federated user named Jill is able to read her own files but not the files of any other federated user.) When you call `GetFederationToken` using the credentials of IAM user `Issuer`, you can pass a policy like the following example.

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": "s3:GetObject",
    "Resource": "arn:aws:s3:::mybucket/federated-user/Jill/*"
  }]
}
```

This policy scopes down the permissions originally granted to user `Issuer` so that the federated user Jill has a subset of the issuer's permissions.

If conditions exist in either policy, the conditions must be satisfied by the request for authorization. For example, if `Issuer` can make Amazon S3 requests only subject to an `aws:SourceIp` condition, that condition also applies to calls made with temporary security credentials issued by `Issuer`.

AWS checks permissions each time a request is made. Imagine that you call `GetFederationToken` using the credentials of user `Issuer` to get temporary security credentials for federated user Jill, and that you pass the following policy in the call:

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

The call to `GetFederationToken` that includes this policy will succeed, even though these permissions allow more access than the policy that's attached to user `Issuer`. However, when federated user Jill makes tries to delete an Amazon S3 bucket, the request fails. Although the policy that was passed

`GetFederationToken` would allow this action, the policy attached to user `Issuer` does not, and Jill's effective permissions are the most restrictive set based on the intersection of the permissions for `Issuer` and the permissions passed in the call.

Permissions in Temporary Security Credentials for IAM Users

When the AWS Security Token Service (STS) `GetSessionToken` API action is called to create temporary security credentials, the credentials returned in the response match those of the IAM user entity that made the call. The user can access only the AWS resources that are granted in the policy or policies that apply to that user.

For more information about IAM user permissions and policies, see Overview of Permissions.

Disabling Permissions Granted Through Temporary Security Credentials

Temporary security credentials are valid until they expire, and they cannot be revoked. However, because policies are evaluated each time an AWS request is made using the temporary security credentials, you can modify access rights for temporary credentials after the credentials have been issued.

For IAM roles (AssumeRole and AssumeRoleWithWebIdentity)

- Delete the IAM role that has been assumed, or modify the role's permissions. For more information, see Modifying a Role and Deleting Roles or Instance Profiles in the *Using IAM* guide.

For IAM users and federated users (GetSessionToken and GetFederationToken)

- Create a resource policy that denies access to the user who created the temporary security credentials.
- Create an IAM user policy for the user who created the temporary security credentials that explicitly denies access to a specific resource by a federated user.

The methods for disabling IAM and federated users using `GetSessionToken` and `GetFederationToken` are described in the following sections.

> **Note**
> When you update existing policy permissions, or when you apply a new policy to a user or a resource, it may take a few minutes for policy updates to take effect.

For more information about how IAM evaluates permissions in the context of temporary security credentials, see Controlling Permissions for Temporary Security Credentials (p. 32). For general information about how IAM evaluates permissions, see Overview of Permissions in *Using IAM*.

Denying Access to the User Who Created the Temporary Security Credentials

To deny access to a user who has temporary security credentials that have not already expired, you can deny access to the IAM user whose credentials were used to generate the temporary credentials. In the case of federated users, denying access to the user who *created* the temporary credentials is effective

because the permissions granted to the federated user cannot exceed the permissions of the IAM user who created the temporary credentials. In the case of IAM users, the holder of the temporary credentials and the creator are the same identity.

In the following policy example, if the AWS account owner applied this resource policy to his Amazon Simple Queue Service (Amazon SQS) queue, the IAM user named John could not send messages from the queue, and neither could any federated users who have temporary security credentials created by John.

```
{
  "Version": "2012-10-17",
  "Id": "Queue1_Policy_UUID",
  "Statement": [{
    "Principal": "arn:aws:iam::111122223333:user/John"
    "Effect": "Deny",
    "Action": "sqs:SendMessage",
    "Resource": "arn:aws:sqs:us-east-1:111122223333:myqueue"
  }]
}
```

Important
An important reason never to use your root credentials to create temporary security credentials is the ability to deny access to the user who created those credentials. By using the credentials of an IAM user (and not your root account credentials) to request temporary security credentials, you can modify permissions for the issuing user without affecting your root account. For information about modifying user permissions, see Managing IAM Policies in *Using IAM*.

Denying Access to a Specific Resource for a Federated User

If you have issued temporary security credentials to a federated user but you want to revoke that user's temporary credentials before they expire (and no one else's), you can create a policy like the one in the following example. The policy includes an explicit `Deny` effect, which takes precedence over an `Allow` effect for the same actions and resources.

To put this policy into effect, you can attach it to the IAM user whose credentials were used to create the temporary security credentials. For this policy, it doesn't matter that IAM doesn't know who federated user `Jill` is. The policy works because it denies the permission of the IAM user who created the temporary security credentials to delegate access to the Amazon S3 `GetObject` action for Jill.

```
 {
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Deny",
    "Action": "s3:GetObject",
    "Resource": "arn:aws:s3:::mybucket/federated-user/Jill/*"
  }]
}
```

Related Topics

- Creating a Role to Allow AWS Access for the Mobile App (p. 14). This section discusses how to configure IAM roles when you use web identity federation and the `AssumeRoleWithWebIdentity` API.

- Overview of Policies in *Using IAM* .

Using Temporary Security Credentials

This document provides an overview of how you use temporary security credentials that you get from AWS STS.

You can use temporary security credentials to make programmatic requests for AWS resources using the AWS SDKs or using API calls, the same way that you can use long-term security credentials such as IAM user credentials. However, there are a few differences:

- When you make a call using temporary security credentials, you must include a session token that is returned along with those temporary credentials. This is used by AWS to validate the temporary security credentials.
- The temporary credentials expire after a specified interval. After the credentials expire, any calls that you make using those credentials will fail, so you must get a new set of credentials.

Using Temporary Security Credentials with the AWS SDKs

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 details about how to call `AssumeRole`, `GetFederationToken`, and other APIs and about how to get 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.

You can 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 APIs

If you're making direct API requests to AWS, you use the temporary access key ID and secret access key as you would use long-term credentials. For most services, you do the following:

- Use the temporary access key ID in place of the long-term access key ID that you would normally use for an AWS call (for example, as the `AWSAccessKeyId` parameter value in a call).
- Sign the request using the secret access key that is provided as part of the temporary security credentials.
- Include the IAM session token that is part of the temporary security credentials. You include the session token as an authorization header to the request—for example, as the `X-Amz-Security-Token` header. (The session token is not part of the information that's used to create the signature.)

The following example uses temporary security credentials to authenticate a `ListUsers` request to IAM. The request uses Signature Version 4 and includes authorization information in the headers.

Sample for services that use Signature Version 4 and add authorization information in the header

```
POST http://iam.amazonaws.com/ HTTP/1.1
Authorization: AWS4-HMAC-SHA256 Credential=Access Key ID provided by AWS Security
 Token Service/20110909/us-east-1/iam/aws4_request, SignedHeaders=host, Signa
ture=signature-calculated-using-the-temporary-access-key
host: iam.amazonaws.com
Content-type: application/x-www-form-urlencoded; charset=utf-8
X-Amz-Date: 20110909T233600Z
X-Amz-Security-Token: session-token

Action=ListUsers&Version=2010-05-08
```

The following example shows an Amazon SimpleDB request that uses Signature Version 2 and includes authorization information in the query string.

Sample for services that use Signature Version 2 and add authorization information in the query string

```
https://sdb.amazonaws.com/
?Action=GetAttributes
&AWSAccessKeyId=access-key-from-AWS Security Token Service
&DomainName=MyDomain
&ItemName=MyItem
&SignatureVersion=2
&SignatureMethod=HmacSHA256
&Timestamp=2010-01-25T15%3A03%3A07-07%3A00
&Version=2009-04-15
&Signature=signature-calculated-using-the-temporary-access-key
&SecurityToken=session-token
```

If you send requests using expired credentials, AWS denies the request.

More Information

For more information about using AWS STS with other AWS services, see the following links:

- **Amazon S3**. See Making Requests Using IAM User Temporary Credentials or Making Requests Using Federated User Temporary Credentials in the *Amazon Simple Storage Service Developer Guide*.
- **Amazon SNS**. See Using Temporary Security Credentials in the *Amazon Simple Notification Service Developer Guide*.
- **Amazon SQS**. See Using Temporary Security Credentials in the *Amazon Simple Queue Service Developer Guide*.
- **Amazon SimpleDB**. See Using Temporary Security Credentials in the *Amazon SimpleDB Developer Guide*.

Giving Federated Users Direct Access to the AWS Management Console

You can give your federated users single sign-on (SSO) access to the AWS Management Console through your identity and authorization system, without requiring users to sign into Amazon Web Services (AWS). The method you use to do this varies depending on how your organization is set up:

- If your organization has an identity system that integrates SAML 2.0 (Security Assertion Markup Language 2.0), you can set things up in your organization and in IAM so that users can seamlessly sign in to a portal inside your organization, select an option to go to the AWS Management Console, and be automatically taken to the console.
- For other scenarios, you can write code that creates a URL that includes identity information. You can distribute the URL to users to give them secure and direct access to the AWS Management Console.

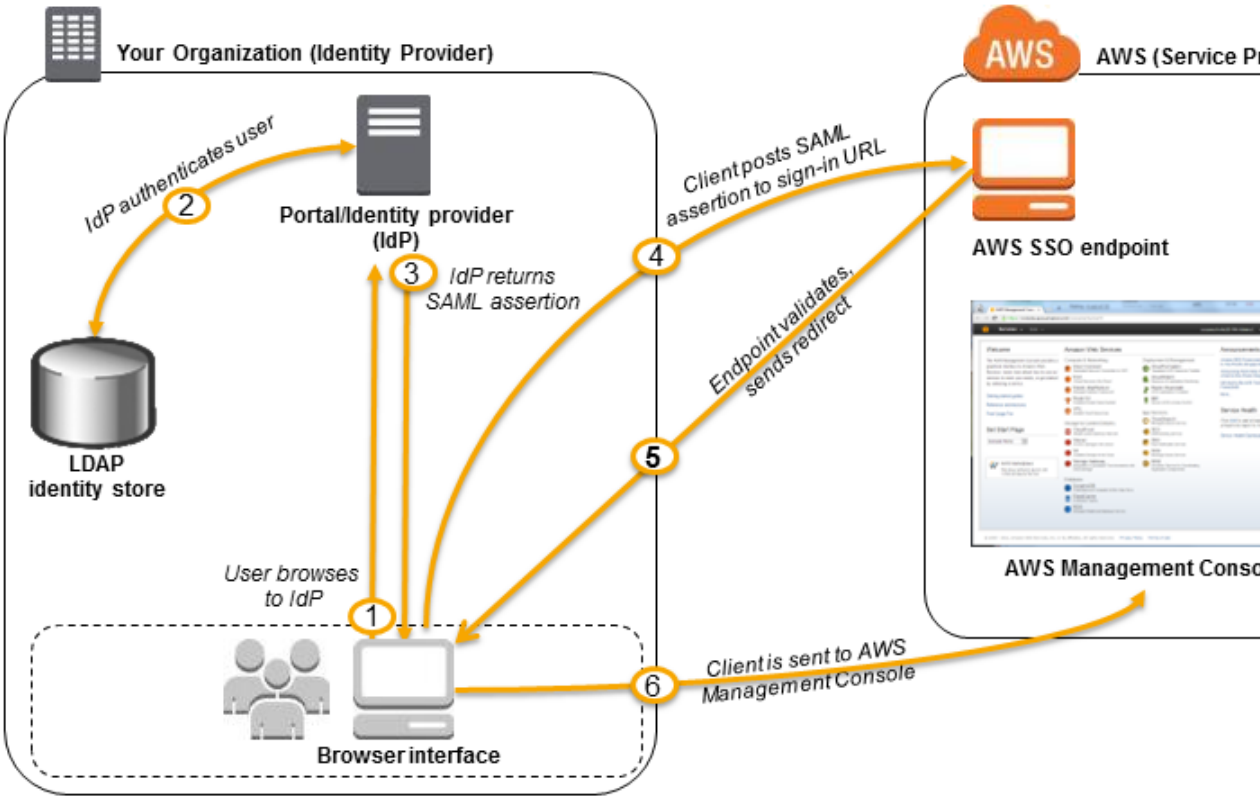
Topics

Giving Console Access Using SAML

If your organization supports SAML, you can let users who have been authenticated in your organization access the AWS Management Console without having to have IAM identities and without having to sign in again. AWS provides a single sign-on (SSO) endpoint (https://signin.aws.amazon.com/saml) that accepts SAML assertions that are used to grant your users federated access to the console.

> **Note**
> You can also configure your IdP and AWS to get temporary security credentials that can be used for programmatic access to AWS resources. For more information, see Creating Temporary Security Credentials for SAML Federation (p. 19).

The following diagram describes the flow for SAML-enabled single sign-on.



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 functions as a identity provider (IdP) that handles the exchange of trust between your organization and AWS.
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. The portal sends this response to the client (the user's browser).
4. The client posts the SAML assertion to an AWS single sign-on endpoint. The endpoint uses the AWS STS AssumeRoleWithSAML API to request temporary security credentials and creates a console sign-in URL.
5. AWS sends the sign-in URL back to the client with a redirect.
6. The client gets the console sign-in and is redirected to the AWS Management Console. (If the authentication response includes attributes that map to multiple IAM roles, the user is prompted to select the role to use for access to 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.

To use SAML-based federation for access to AWS resources, you perform steps inside your organization's network in order to configure it as an identity provider. You then configure AWS to act as a service provider.

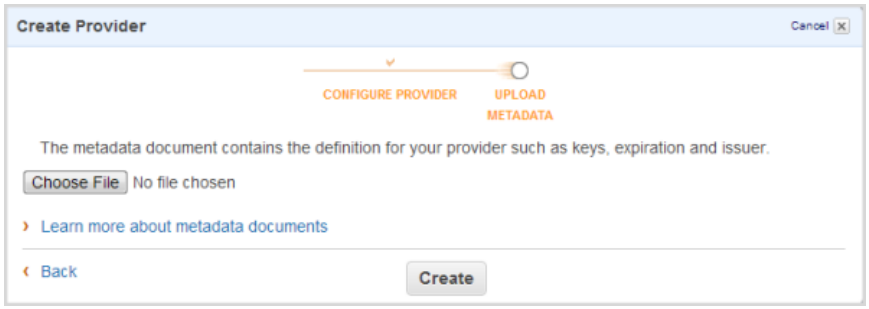
Topics

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 identity provider (IdP) like Windows Active Directory Federation Services, Shibboleth, etc. Using your IdP, you generate a metadata document that describes your organization as an identity provider 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.

Create a SAML Provider in IAM

Next, you go to the AWS Management Console. In the IAM console, you create a new SAML provider, which is an entity in IAM that holds information about your organization as an identity provider. As part of this process, you upload the metadata document that was produced by the SAML software in your organization.



For details, see Managing SAML Providers in the *Using IAM* guide.

When this task is done, you can create an IAM role that will be able to establish a trust relationship between your organization and IAM and that identities your organization as a principal (trusted entity) for purposes of federation.

Establish Permissions in AWS for Federated Users

The next step is to create an IAM role that defines what users from your organization will be allowed to do in AWS. You can use the IAM console to create this role. When you create the trust policy for the role, you specify the SAML provider that you created earlier in IAM, and you specify a SAML attribute that describes the user. For example, you can specify that only users whose SAML `eduPersonOrgDN` value is `ExampleOrg` will be 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 SSO. The trust role might look like this:

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

```
        "Federated": "arn:aws:iam::account-number-without-hyphens:saml-pro
vider/ExampleOrgSSOProvider"
      },
      "Action": "sts:AssumeRoleWithSAML",
      "Condition": {
        "StringEquals": {
          SAML:eduPersonOrgDN": "ExampleOrg",
          "SAML:aud": "https://signin.aws.amazon.com/saml"
        }
      }
    }
  ]
}
```

For the access (permissions) policy in the role, you specify permissions as you would for any role. For example, if users from your organization will be allowed to administer Amazon EC2 instances, you explicitly allow Amazon EC2 actions in the permissions policy, such as those in the **Amazon EC2 Full Access** policy template.

For details, see Creating a Role for SAML-Based Federation in the *Using IAM* guide.

Configure Assertions for the SAML Authentication Response

In your organization, after a user's identity has been verified, the IdP must send an authentication response to the AWS endpoint (https://signin.aws.amazon.com/saml). This response must be a POST request that includes a SAML token that adheres to SAML standards and that contains the following assertions. All of these assertions are required.

- `Subject` and `NameID`. The following excerpt shows an example. Your own values would substitute for the marked ones.

```
<Subject>
  <NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persist
ent">_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>
```

- An attribute with `Name` set to `https://aws.amazon.com/SAML/Attributes/Role`. This attribute contains one or more `AttributeValue` elements that list the IAM role and SAML provider that the user is mapped to in your IdP. The role and provider are specified as a comma-delimited pair of ARNs, in the same format that they are used for the `RoleArn` and `PrincipalArn` parameters that are passed to AssumeRoleWithSAML. The attribute must contain at least one role/provider pair, and can contain multiple pairs. If the attribute contains multiple pairs, when the user uses WebSSO to sign into the AWS Management Console, he or she is asked to select the role to assume.

```
<Attribute Name="https://aws.amazon.com/SAML/Attributes/Role">
  <AttributeValue>arn:aws:iam::account-number:role/role-name,arn:aws:iam::ac
count-number:saml-provider/provider-name</AttributeValue>
  <AttributeValue>arn:aws:iam::account-number:role/role-name,arn:aws:iam::ac
```

```
count-number:saml-provider/provider-name</AttributeValue>
  <AttributeValue>arn:aws:iam::account-number:role/role-name,arn:aws:iam::ac
count-number:saml-provider/provider-name</AttributeValue>
</Attribute>
```

- An attribute with `Name` set to `https://aws.amazon.com/SAML/Attributes/RoleSessionName`. The attribute value provides an identifier for the AWS temporary credentials that are issued for SSO. The value is typically a user name or email address for the principal, and is displayed in the AWS console. This value must be between 2 and 32 characters long, can contain only alphanumeric characters, underscores, and the following characters: `+=,.@:-`. It cannot contain spaces.

```
<Attribute Name="https://aws.amazon.com/SAML/Attributes/RoleSessionName">
  <AttributeValue>name</AttributeValue>
</Attribute>
```

Mapping SAML Attributes to AWS Policy Keys

The tables in this section list how commonly used SAML attributes are mapped to policy keys in AWS.

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 policies using `StringEquals` or `StringLike` conditions. For list values, you can test the values in policies using the `ForAnyValue` and `ForAllValues` policy set operators.

eduPerson and eduOrg Attributes

eduPerson or eduOrg Attribute	AWS Key	Type
urn:oid:1.3.6.1.4.1.5923.1.1.1.1	eduPersonAffiliation	List of string
urn:oid:1.3.6.1.4.1.5923.1.1.1.2	eduPersonNickname	List of string
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 string
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 string
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 string
urn:oid:1.3.6.1.4.1.5923.1.1.1.10	eduPersonTargetedID	List of string
urn:oid:1.3.6.1.4.1.5923.1.1.1.11	eduPersonAssurance	List of string
urn:oid:1.3.6.1.4.1.5923.1.2.1.2	eduOrgHomePageURI	List of string
urn:oid:1.3.6.1.4.1.5923.1.2.1.3	eduOrgIdentityAuthNPolicyURI	List of string
urn:oid:1.3.6.1.4.1.5923.1.2.1.4	eduOrgLegalName	List of string
urn:oid:1.3.6.1.4.1.5923.1.2.1.5	eduOrgSuperiorURI	List of string

eduPerson or eduOrg Attribute	AWS Key	Type
urn:oid:1.3.6.1.4.1.5923.1.2.1.6	eduOrgWhitePagesURI	List of string
urn:oid:2.5.4.3	cn	List of string

Giving Console Access by Creating a URL

You can let users who have signed in to your organization's network access the AWS Management Console by using code to create a URL that gives them secure and direct access to the console.

> **Note**
> If your organization uses SAML, you can set up access to the AWS Management Console without writing code. For details, see Giving Console Access Using SAML (p. 41).

To create the URL you need to complete the following tasks:

- Verify that the user is authenticated.
- Create temporary security credentials for the user.
- Construct the URL that passes the temporary security credentials to the AWS Management Console.
- Distribute the URL to the user.

The URL is valid for 15 minutes from the time it is created. The temporary security credentials associated with the URL are valid for the duration you specified when you created them, starting from the time they were created.

> **Important**
> Keep in mind that the URL grants access to your AWS resources through the AWS Management Console, to the extent that 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#. Each of these methods is described in the following sections.

Constructing the URL for the AWS Management Console (Query APIs)

This topic describes how to construct a URL that gives your federated users direct access to the AWS Management Console. This task uses the AWS Identity and Access Management (IAM) and AWS Security Token Service (AWS STS) HTTPS Query API. For more information about making Query requests, go to Making Query Requests in *Using IAM*.

> **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. Create temporary security credentials for the user. The credentials consist of an access key ID, a secret access key, and a security token. For more information about creating temporary credentials, see Creating Temporary Security Credentials (p. 11).

   Important
   When you create temporary security credentials, you must specify the permissions the credentials will grant to the user who holds them. For more information about controlling permissions in temporary security credentials, see Controlling Permissions for Temporary Security Credentials (p. 32).
3. After you obtain the temporary security credentials, you format them as a JSON string so that you can 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 create.

```
{"sessionId":"*** AWS Access Key ID ***",
"sessionKey":"*** AWS Secret Access Key ***",
"sessionToken":"*** AWS security token ***"}
```

4. Next, make a request to the AWS federation endpoint (https://signin.aws.amazon.com/federation) with the `Action` and `Session` parameters, as shown in the following example.

```
Action = getSigninToken
Session = *** the JSON string described in Step 3, form-urlencoded ***
```

   The following string is an example of what your request might look like.

```
https://signin.aws.amazon.com/federation?
Action=getSigninToken
&Session=%7B%22sessionId%22%3A%22ASIAEXAMPLEMDLUUAEYQ%22%2C%22sessionKey%22
%3A%22tpSl9thxr2PkEXAMPLETAnVLVGdwC5zXtGDr%2FqWi%22%2C%22sessionToken%22%3A
%22AQoDYXdzEXAMPLE4BrM96BJ7btBQRrAcCjQIbg55555555OBT7y8h2YJ7woJkRzsLpJBpklC
qPXxS2AjRorJAm%2BsBtv1YXlZF%2FfHljgORxOevE388GdGaKRfO9W4DxK4HU0fIpwL%2BQ7oX
2Fj%2BJa%2FAb5u0cL%2BzI1P5rJuDzH%2F0pWEiYfiWXXH20rWruXVXpIIO%2FPhMHlV3Jw%2B
gDc4ZJ0WItuLPsuyP7BVUXWLcAVyTFbxyLy36FBSXF1z8a%2FvJN7utcj0mJRGIiIZSV7FQuepa
WP5YARYMrOUMqBB3v308LKBU8Z0xYe2%2FqthrLXf1nX0njbU%2FJTrct%2BEdG9PRb3907qa5n
VbnnnxdVQJ3mPgQchAZpDI9LsDDbGsa67JHUyFYnyUUUkMRfe7G70gjvbz9gQ%EXAMPLE
```

   The response is a JSON document with an `SigninToken` value. It will look similar to the following example.

```
{"SigninToken":"*** the SigninToken string ***"}
```

5. Finally, you create the URL that your federated users will use to access the AWS Management Console. The URL is the federation URL endpoint (https://signin.aws.amazon.com/federation), plus the following parameters:

```
Action = login
Issuer = *** the form-urlencoded URL for your internal sign-in page ***
Destination = *** the desired AWS Management Console URL, also
 form-urlencoded ***
```

```
SigninToken = *** the value of SigninToken from the JSON document returned
 in Step 4 ***
```

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 associated with the URL are valid for the duration you specified when you created them.

```
https://signin.aws.amazon.com/federation?
Action=login
&Issuer=https%3A%2F%2Fexample.com
&Destination=https%3A%2F%2Fconsole.aws.amazon.com%2Fs
&SigninToken=VCQgs5qZZt3Q6fn8Tr5EXAMPLEmLnwB7JjUc-SHwnUUWabcRdnWsi4DBn-dvC
CZ85wrD0nmldUcZEXAMPLE-vXYH4Q__mleuF_W2BE5HYexbe9y4Of-kje53SsjNNecATfjIzpW1
WibbnH6YcYRiBoffZBGExbEXAMPLE5aiKX4THWjQKC6gg6alHu6JFrnOJoK3dtP6I9a6hi6yPgm
iOkPZMmNGmhsvVxetKzr8mx3pxhHbMEXAMPLETv1pij0rok3IyCR2YVcIjqwfWv32HU2Xlj471u
3fU6uOfUComeKiqTGX974xzJOZbdmX_t_lLrhEXAMPLEDDIisSnyHGw2xaZZqudm4mo2uTDk9Pv
9l5K0ZCqIgEXAMPLEcA6tgLPykEWGUyH6BdSC6166n4M4JkXIQgac7_7821YqixsNxZ6rsrpzwf
nQoS14O7R0eJCCJ684EXAMPLEZRdBNnuLbUYpz2Iw3vIN0tQgOujwnwydPscM9F7foaEK3jwMkg
Apeb1-6L_OB12MZhuFxx55555EXAMPLEhyETEd4ZulKPdXHkgl6T9ZkIlHz2Uy1RUTUhhUxNtSQ
nWc5xkbBoEcXqpoSIeK7yhje9Vzhd61AEXAMPLElbWeouACEMG6-Vd3dAgFYd6i5FYoyFrZLWvm
0LSG7RyYKeYN5VIzUk3YWQpyjP0RiT5KUrsUi-NEXAMPLExMOMdoODBEgKQsk-iu2ozh6r8bxwC
RNhujg
```

Constructing the URL for the AWS Management Console (Java)

This topic describes how to programmatically construct a URL that gives your federated users direct access to the AWS Management Console. The following code snippet uses the AWS SDK for Java. You replace the placeholder text with your own values.

```
import java.net.URLEncoder;
import java.net.URL;
import java.net.URLConnection;
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.AWSCredentials;
import com.amazonaws.auth.BasicAWSCredentials;
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;

AWSCredentials credentials = new BasicAWSCredentials(
  "*** Access Key ID ***",
  "*** Secret Key ***");
AWSSecurityTokenServiceClient stsClient =
  new AWSSecurityTokenServiceClient(credentials);

GetFederationTokenRequest getFederationTokenRequest =
  new GetFederationTokenRequest();
```

```
getFederationTokenRequest.setDurationSeconds(3600);
getFederationTokenRequest.setName("UserName");

// A sample policy for accessing Amazon SNS in the console.
String policy = "{\"Version\":\"2012-10-17\",\"Statement\":[{\"Ac
tion\":\"sns:*\"," +
  "\"Effect\":\"Allow\",\"Resource\":\"*\"}]}";

getFederationTokenRequest.setPolicy(policy);

GetFederationTokenResult federationTokenResult =
  stsClient.getFederationToken(getFederationTokenRequest);

Credentials federatedCredentials = federationTokenResult.getCredentials();

// The issuer parameter specifies your internal sign-in
// page, for example https://mysignin.internal.mycompany.com/.
// The console parameter specifies the URL to the destination console of the
// AWS Management Console. This example goes to Amazon SNS.
// The signin parameter is the URL to send the request to.
String issuerURL = "https://mysignin.internal.mycompany.com/";
String consoleURL = "https://console.aws.amazon.com/sns";
String signInURL = "https://signin.aws.amazon.com/federation";

// Create the sign-in token using temporary credentials,
// including the Access Key ID,  Secret Access Key, and security token.
String sessionJson = String.format(
  "{\"%1$s\":\"%2$s\",\"%3$s\":\"%4$s\",\"%5$s\":\"%6$s\"}",
  "sessionId", federatedCredentials.getAccessKeyId(),
  "sessionKey", federatedCredentials.getSecretAccessKey(),
  "sessionToken", federatedCredentials.getSessionToken());

String getSigninTokenURL = signInURL + "?Action=getSigninToken" +
  "&SessionType=json&Session=" + URLEncoder.encode(sessionJson,
  "UTF-8");
URL url = new URL(getSigninTokenURL);
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");
String destinationParameter = "&Destination=" +
  URLEncoder.encode(consoleURL,"UTF-8");
String loginURL = signInURL + "?Action=login" + signinTokenParameter +
  issuerParameter + destinationParameter;
```

Constructing the URL for the AWS Management Console (Ruby)

This topic describes how to programmatically construct a URL that gives your 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'S
require 'aws-sdk'

# Normally, the temporary credentials will come from your proxy
# application, but for this example we create them here
sts = AWS::STS.new(:access_key_id => "*** Your AWS Access Key ID ***",
  :secret_access_key => "*** Your AWS Secret Access Key ***")

# A sample policy for accessing Amazon SNS in the console.
policy = AWS::STS::Policy.new
policy.allow(:actions => "sns:*",:resources => :any)

session = sts.new_federated_session(
  "UserName",
  :policy => policy,
  :duration => 3600)

# 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.
issuer_url = "https://mysignin.internal.mycompany.com/"
console_url = "https://console.aws.amazon.com/sns"
signin_url = "https://signin.aws.amazon.com/federation"

# Create the sign-in token using temporary credentials,
# including the Access Key ID, Secret Access Key, and security token.
session_json = {
  :sessionId => session.credentials[:access_key_id],
  :sessionKey => session.credentials[:secret_access_key],
  :sessionToken => session.credentials[:session_token]
}.to_json

get_signin_token_url = signin_url + "?Action=getSigninToken" +
  "&SessionType=json&Session=" + CGI.escape(session_json)
returned_content = URI.parse(get_signin_token_url).read
signin_token = JSON.parse(returned_content)['SigninToken']
signin_token_param = "&SigninToken=" + CGI.escape(signin_token)

# The issuer parameter is optional, but recommended. Use it to direct users
# to your sign-in page when their session expires.
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
```

AWS Security Token Service Sample Applications

To see how you can use AWS STS to manage temporary security credentials, you can download the following sample applications that implement complete example scenarios.

- Identity Federation Sample Application for an Active Directory Use Case. Demonstrates how to issue temporary security credentials for accessing Amazon S3 files and buckets, using permissions that are tied to an Active Directory user. (.NET/C#)
- AWS Management Console Federation Proxy Sample Use Case. Demonstrates how to create a federation proxy that enables single sign-on (SSO) so that existing Active Directory users can sign into the AWS Management Console. (.NET/C#)
- Integrate Shibboleth with AWS Identity and Access Management. Shows how to use Shibboleth and SAML to provide users with single sign-on (SSO) access to the AWS Management Console.
- AWS SDK for iOS and AWS SDK for Android. These SDKs contain a sample application that demonstrates how to use web identity federation (p. 12), which lets you create a mobile app or client-based web app where users can sign in using Login with Amazon, Facebook, or Google. The samples include code that shows how to invoke the identity providers, and then how to use the information from these providers to get and use temporary security credentials.
- Web Identity Federation Playground. This website provides an interactive demonstration of web identity federation (p. 12).
- Authenticating Users of AWS Mobile Applications with a Token Vending Machine at *AWS Articles & Tutorials*. Demonstrates a server-based proxy application that serves temporary credentials to remote clients (such as mobile apps) so that the clients can sign web requests to AWS. This sample can be used with the sample client that is part of the AWS SDK for Android and the AWS SDK for iOS. (Java) For more information, see Credential Management for Mobile Applications, which is an article that provides additional details on how to secure AWS resources when using the token vending machine (TVM) with mobile applications.

AWS Services that Support AWS Security Token Service (AWS STS)

The following table describes the AWS products that support requests made using the temporary security credentials that are generated by AWS STS API actions.

For information about how to use temporary security credentials with the AWS SDKs or when making API calls, see Using Temporary Security Credentials (p. 38).

AWS Product	Supports Temporary Security Credentials?
Auto Scaling	Yes
AWS Account Billing	Yes
AWS CloudFormation	Yes
Amazon CloudFront	Yes
AWS CloudHSM	No
Amazon CloudSearch	Yes
Amazon CloudWatch	Yes
AWS Data Pipeline	Yes
AWS Direct Connect	Yes
Amazon DynamoDB	Yes
AWS Elastic Beanstalk	No
Amazon Elastic Compute Cloud (Amazon EC2)	Yes
Elastic Load Balancing	Yes
Amazon Elastic MapReduce (Amazon EMR)	No
Amazon Elastic Transcoder	Yes
Amazon ElastiCache	Yes

AWS Product	Supports Temporary Security Credentials?
Amazon Flexible Payments Service (Amazon FPS)	Yes
Amazon Fulfillment Web Service (Amazon FWS)	Yes
Amazon Glacier	Yes
AWS Identity and Access Management (IAM)	Yes; see below
AWS Import/Export	No
AWS Marketplace	Yes
Amazon Mechanical Turk	No
AWS OpsWorks	Yes
Amazon Redshift	Yes
Amazon Relational Database Service (Amazon RDS)	Yes
Amazon Route 53	Yes
Amazon Simple Storage Service (Amazon S3)	Yes
Amazon Simple Email Service (Amazon SES)	Yes
Amazon Simple Notification Service (Amazon SNS)	Yes
Amazon Simple Queue Service (Amazon SQS)	Yes
Amazon SimpleDB	Yes
AWS Storage Gateway	Yes
Amazon Simple Workflow Service	Yes
AWS Support	No
AWS Security Token Service	Yes; see below
AWS Storage Gateway	Yes
Amazon Virtual Private Cloud (Amazon VPC)	Yes

- **IAM**. Supports `AssumeRole`, `AssumeRoleWithWebIdentity`, and `AssumeRoleWithSAML`. If you use `GetFederationToken`, you can access IAM when using single sign-on to the AWS Management Console, but not from the API or CLI. For more information, see Giving Federated Users Direct Access to the AWS Management Console (p. 41).
- **AWS STS**. You can use the temporary security credentials that you get from the `AssumeRole`, `AssumeRoleWithWebIdentity`, or `AssumeRoleWithSAML` call to make subsequent calls to `AssumeRole`; however, you cannot use those credentials to call `GetFederationToken` or `GetSessionToken`. You cannot use the temporary security credentials from `GetFederationToken` or `GetSessionToken` to call any STS APIs.

More Information

For more information about using AWS STS with other AWS services, see the following links:

- **Amazon S3**. See Making Requests Using IAM User Temporary Credentials or Making Requests Using Federated User Temporary Credentials in the *Amazon Simple Storage Service Developer Guide*.
- **Amazon SNS**. See Using Temporary Security Credentials in the *Amazon Simple Notification Service Developer Guide*.
- **Amazon SQS**. See Using Temporary Security Credentials in the *Amazon Simple Queue Service Developer Guide*.
- **Amazon SimpleDB**. See Using Temporary Security Credentials in the *Amazon SimpleDB Developer Guide*.

Document History

The following table describes the documentation for this release of the AWS Security Token Service.

- **API version**: 2011-06-15
- **Latest documentation update**: May 21, 2013

Change	Description	Release Date
Identity federation using SAML	Added support for identity federation using the Security Assertion Markup Language (SAML) 2.0. With this feature, your organization acts as a SAML-enabled identity provider, and AWS acts as a service provider. Your organization can use SAML assertions to get temporary security credentials in order to access AWS resources and to support single sign-in (SSO) for the AWS Management Console. For more information, see Creating Temporary Security Credentials for SAML Federation (p. 19).	This release
Web identity federation	Added support for web identity federation. This feature lets you get temporary security credentials for users who have signed in using Login with Amazon, Facebook, or Google. For more information, see Creating Temporary Security Credentials for Mobile Apps Using Identity Providers (p. 12).	May 29, 2013
MFA-Protected API access	Introduced MFA-protected API access, a feature that enables you to add an extra layer of security over AWS APIs using AWS Multi-Factor Authentication (MFA), see Temporary Security Credentials for IAM Users with Multi-Factor Authentication (MFA) (p. 30).	July 10, 2012
Fixed API version in documentation	Corrected the API version displayed in *Using Temporary Security Credentials*. The API version of the AWS Security Token Service is not the same as the AWS Identity and Access Management IAM API version.	April 26, 2012
New Guide	This release introduces *Using Temporary Security Credentials*.	January 19, 2012