About SAML 2.0-based federation

PDF (iam-ug.pdf#id_roles_providers_saml) | **RSS (aws-iam-release-notes.rss)**

AWS supports identity federation with SAML 2.0 (Security Assertion Markup Language 2.0) (https://wiki.oasis-open.org/security) , an open standard that many identity providers (IdPs) use. This feature enables federated single sign-on (SSO), so users can log into the AWS Management Console or call the AWS API operations without you having to create an IAM user for everyone in your organization. By using SAML, you can simplify the process of configuring federation with AWS, because you can use the IdP's service instead of writing custom identity proxy code (https://docs.aws.amazon.com/STS/latest/UsingSTS/CreatingFedTokens.html) .

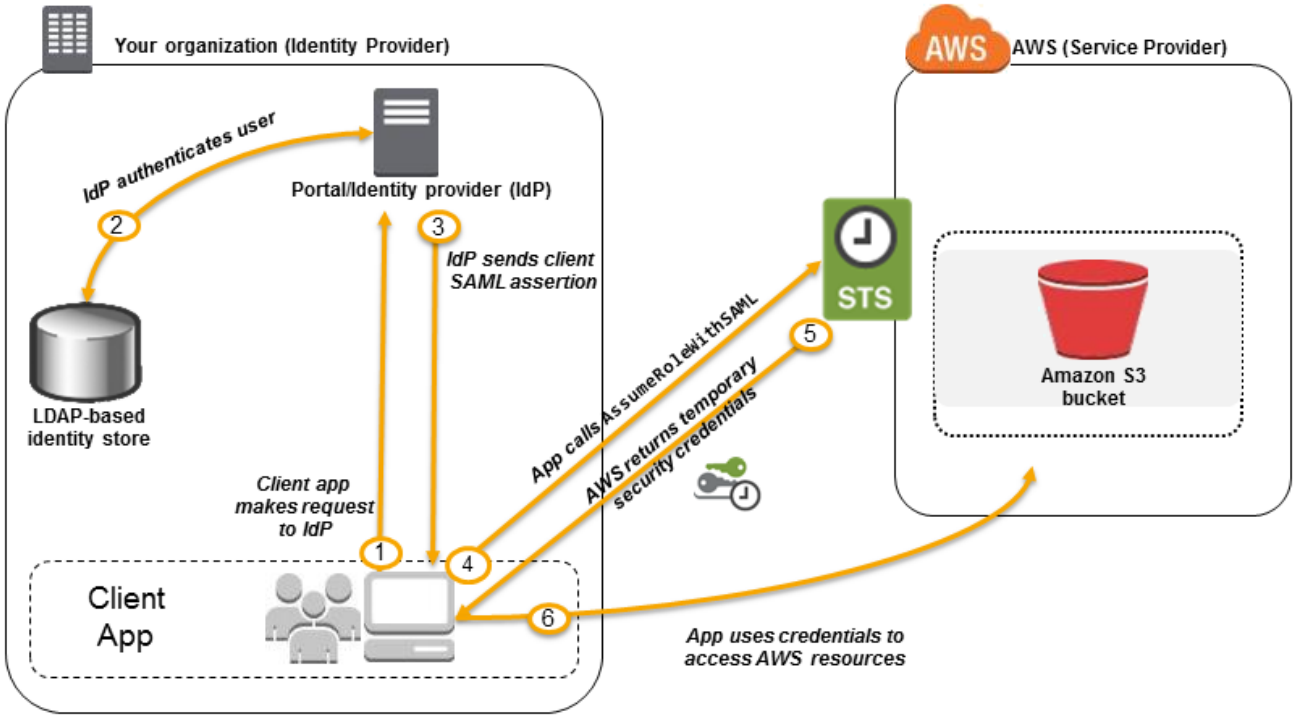
IAM federation supports these use cases:

- Federated access to allow a user or application in your organization to call AWS API operations (#CreatingSAML-configuring) . You use a SAML assertion (as part of the authentication response) that is generated in your organization to get temporary security credentials. This scenario is similar to other federation scenarios that IAM supports, like those described in Requesting temporary security credentials (./id_credentials_temp_request.html) and About web identity federation (./id_roles_providers_oidc.html) . However, SAML 2.0–based IdPs in your organization handle many of the details at run time for performing authentication and authorization checking. This is the scenario discussed in this topic.
- Web-based single sign-on (SSO) to the AWS Management Console from your organization (./id_roles_providers_enable-console-saml.html) . Users can sign in to a portal in your organization hosted by a SAML 2.0–compatible IdP, select an option to go to AWS, and be redirected to the console without having to provide additional sign-in information. You can use a third-party SAML IdP to establish SSO access to the console or you can create a custom IdP to enable console access for your external users. For more information about building a custom IdP, see Enabling custom identity broker access to the AWS console (./id_roles_providers_enable-console-custom-url.html) .

Using SAML-based federation for API access to AWS

Assume that you want to provide a way for employees to copy data from their computers to a backup folder. You build an application that users can run on their computers. On the back end,

the application reads and writes objects in an S3 bucket. Users don't have direct access to AWS. Instead, the following process is used:



1. A user in your organization uses a client app to request authentication from your organization's IdP.
2. The IdP authenticates the user against your organization's identity store.
3. The IdP constructs a SAML assertion with information about the user and sends the assertion to the client app.
4. The client app calls the AWS STS `AssumeRoleWithSAML` (https://docs.aws.amazon.com/STS/latest/APIReference/API_AssumeRoleWithSAML.html) API, passing the ARN of the SAML provider, the ARN of the role to assume, and the SAML assertion from IdP.
5. The API response to the client app includes temporary security credentials.
6. The client app uses the temporary security credentials to call Amazon S3 API operations.

Overview of configuring SAML 2.0-based federation

Before you can use SAML 2.0-based federation as described in the preceding scenario and diagram, you must configure your organization's IdP and your AWS account to trust each other. The general process for configuring this trust is described in the following steps. Inside your organization, you must have an IdP that supports SAML 2.0 (./id_roles_providers_saml_3rd-

party.html) , like Microsoft Active Directory Federation Service (AD FS, part of Windows Server), Shibboleth, or another compatible SAML 2.0 provider.

To configure your organization's IdP and AWS to trust each other

1. Register AWS as a service provider (SP) with the IdP of your organization. Use the SAML metadata document from `https://` *`region-code`* `.signin.aws.amazon.com/static/saml-metadata.xml`

   For a list of possible *`region-code`* values, see the **Region** column in AWS Sign-In endpoints (https://docs.aws.amazon.com/general/latest/gr/signin-service.html) .

   You can optionally use the SAML metadata document from `https://signin.aws.amazon.com/static/saml-metadata.xml` .
2. Using your organization's IdP, you generate an equivalent metadata XML file that can describe your IdP as an IAM identity provider in AWS. It must include the issuer name, a creation date, an expiration date, and keys that AWS can use to validate authentication responses (assertions) from your organization.
3. In the IAM console, you create a SAML identity provider entity. As part of this process, you upload the SAML metadata document that was produced by the IdP in your organization in Step 2 (#createxml) . For more information, see Creating IAM SAML identity providers (./id_roles_providers_create_saml.html) .
4. In IAM, you create one or more IAM roles. In the role's trust policy, you set the SAML provider as the principal, which establishes a trust relationship between your organization and AWS. The role's permission policy establishes what users from your organization are allowed to do in AWS. For more information, see Creating a role for a third-party Identity Provider (federation) (./id_roles_create_for-idp.html) .
5. In your organization's IdP, you define assertions that map users or groups in your organization to the IAM roles. Note that different users and groups in your organization might map to different IAM roles. The exact steps for performing the mapping depend on what IdP you're using. In the earlier scenario (#CreatingSAML-configuring) of an Amazon S3 folder for users, it's possible that all users will map to the same role that provides Amazon S3 permissions. For more information, see Configuring SAML assertions for the authentication response (./id_roles_providers_create_saml_assertions.html) .

   If your IdP enables SSO to the AWS console, then you can configure the maximum duration of the console sessions. For more information, see Enabling SAML 2.0 federated users to access the AWS Management Console (./id_roles_providers_enable-console-saml.html) .

> **Note**
>
> The AWS implementation of SAML 2.0 federation does not support encrypted SAML assertions between the IAM identity provider and AWS. However, the traffic between

the customer's systems and AWS is transmitted over an encrypted (TLS) channel.

6. In the application that you're creating, you call the AWS Security Token Service `AssumeRoleWithSAML` API, passing it the ARN of the SAML provider you created in Step 3 (#samlovrcreateentity) , the ARN of the role to assume that you created in Step 4 (#samlovrcreaterole) , and the SAML assertion about the current user that you get from your IdP. AWS makes sure that the request to assume the role comes from the IdP referenced in the SAML provider.

   For more information, see AssumeRoleWithSAML (https://docs.aws.amazon.com/STS/latest/APIReference/API_AssumeRoleWithSAML.html) in the *AWS Security Token Service API Reference*.
7. If the request is successful, the API returns a set of temporary security credentials, which your application can use to make signed requests to AWS. Your application has information about the current user and can access user-specific folders in Amazon S3, as described in the previous scenario.

Overview of the role to allow SAML-federated access to your AWS resources

The role or roles that you create in IAM define what federated users from your organization are allowed to do in AWS. When you create the trust policy for the role, you specify the SAML provider that you created earlier as the `Principal` . You can additionally scope the trust policy with a `Condition` to allow only users that match certain SAML attributes to access the role. For example, you can specify that only users whose SAML affiliation is `staff` (as asserted by https://openidp.feide.no) are allowed to access the role, as illustrated by the following sample policy:

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Principal": {"Federated": "arn:aws:iam::account-id:saml-
provider/ExampleOrgSSOProvider"},
    "Action": "sts:AssumeRoleWithSAML",
    "Condition": {
      "StringEquals": {
        "saml:aud": "https://signin.aws.amazon.com/saml",
        "saml:iss": "https://openidp.feide.no"
      },
```

```
      "ForAllValues:StringLike": {"saml:edupersonaffiliation":
["staff"]}
    }
  }]
}
```

For more information about the SAML keys that you can check in a policy, see Available keys for SAML-based AWS STS federation (./reference_policies_iam-condition-keys.html#condition-keys-saml) .

You can include regional endpoints for the `saml:aud` attribute at `https://` *`region-code`* `.signin.aws.amazon.com/static/saml-metadata.xml` . For a list of possible *`region-code`* values, see the **Region** column in AWS Sign-In endpoints (https://docs.aws.amazon.com/general/latest/gr/signin-service.html) .

For the permission policy in the role, you specify permissions as you would for any role. For example, if users from your organization are allowed to administer Amazon Elastic Compute Cloud instances, you must explicitly allow Amazon EC2 actions in the permissions policy, such as those in the **AmazonEC2FullAccess** managed policy.

Uniquely identifying users in SAML-based federation

When you create access policies in IAM, it's often useful to be able to specify permissions based on the identity of users. For example, for users who have been federated using SAML, an application might want to keep information in Amazon S3 using a structure like this:

```
myBucket/app1/user1
myBucket/app1/user2
myBucket/app1/user3
```

You can create the bucket (`myBucket`) and folder (*`app1`*) through the Amazon S3 console or the AWS CLI, since those are static values. However, the user-specific folders (*`user1`* , *`user2`* , *`user3`* , etc.) have to be created at run time using code, since the value that identifies the user isn't known until the first time the user signs in through the federation process.

To write policies that reference user-specific details as part of a resource name, the user identity has to be available in SAML keys that can be used in policy conditions. The following keys are available for SAML 2.0–based federation for use in IAM policies. You can use the values returned by the following keys to create unique user identifiers for resources like Amazon S3 folders.

- `saml:namequalifier` . A hash value based on the concatenation of the `Issuer` response value (`saml:iss`) and a string with the `AWS` account ID and the friendly name (the last part

of the ARN) of the SAML provider in IAM. The concatenation of the account ID and friendly name of the SAML provider is available to IAM policies as the key `saml:doc`. The account ID and provider name must be separated by a '/' as in "123456789012/provider_name". For more information, see the `saml:doc` key at Available keys for SAML-based AWS STS federation (./reference_policies_iam-condition-keys.html#condition-keys-saml) .

The combination of `NameQualifier` and `Subject` can be used to uniquely identify a federated user. The following pseudocode shows how this value is calculated. In this pseudocode `+` indicates concatenation, `SHA1` represents a function that produces a message digest using SHA-1, and `Base64` represents a function that produces Base-64 encoded version of the hash output.

```
Base64 ( SHA1 ( "https://example.com/saml" + "123456789012" +
"/MySAMLIdP" ) )
```

For more information about the policy keys that are available for SAML-based federation, see Available keys for SAML-based AWS STS federation (./reference_policies_iam-condition-keys.html#condition-keys-saml) .

- `saml:sub` (string). This is the subject of the claim, which includes a value that uniquely identifies an individual user within an organization (for example, `_cbb88bf52c2510eabe00c1642d4643f41430fe25e3`).
- `saml:sub_type` (string). This key can be `persistent`, `transient`, or the full `Format` URI from the `Subject` and `NameID` elements used in your SAML assertion. A value of `persistent` indicates that the value in `saml:sub` is the same for a user across all sessions. If the value is `transient`, the user has a different `saml:sub` value for each session. For information about the `NameID` element's `Format` attribute, see Configuring SAML assertions for the authentication response (./id_roles_providers_create_saml_assertions.html) .

The following example shows a permission policy that uses the preceding keys to grant permissions to a user-specific folder in Amazon S3. The policy assumes that the Amazon S3 objects are identified using a prefix that includes both `saml:namequalifier` and `saml:sub`. Notice that the `Condition` element includes a test to be sure that `saml:sub_type` is set to `persistent`. If it is set to `transient`, the `saml:sub` value for the user can be different for each session, and the combination of values should not be used to identify user-specific folders.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "s3:GetObject",
      "s3:PutObject",
```

```
      "s3:DeleteObject"
    ],
    "Resource": [

"arn:aws:s3:::exampleorgBucket/backup/${saml:namequalifier}/${saml:
sub}",

"arn:aws:s3:::exampleorgBucket/backup/${saml:namequalifier}/${saml:
sub}/*"
    ],
    "Condition": {"StringEquals": {"saml:sub_type": "persistent"}}
  }
}
```

For more information about mapping assertions from the IdP to policy keys, see Configuring SAML assertions for the authentication response (./id_roles_providers_create_saml_assertions.html) .