Request temporary security credentials - AWS Identity and Access Management

Archived: 2026-04-05 23:44:39 UTC

To request temporary security credentials, you can use AWS Security Token Service (AWS STS) operations in the AWS API. These include operations to create and provide trusted users with temporary security credentials that can control access to your AWS resources. For more information about AWS STS, see Temporary security credentials in IAM. To learn about the different methods that you can use to request temporary security credentials by assuming a role, see Methods to assume a role.

To call the API operations, you can use one of the AWS SDKs. The SDKs are available for a variety of programming languages and environments, including Java, .NET, Python, Ruby, Android, and iOS. The SDKs take care of tasks such as cryptographically signing your requests, retrying requests if necessary, and handling error responses. You can also use the AWS STS Query API, which is described in the AWS Security Token Service API Reference. Finally, two command line tools support the AWS STS commands: the AWS Command Line Interface, and the AWS Tools for Windows PowerShell.

The AWS STS API operations create a new session with temporary security credentials that include an access key pair and a session token. The access key pair consists of an access key ID and a secret key. Users (or an application that the user runs) can use these credentials to access your resources. You can create a role session and pass session policies and session tags programmatically using AWS STS API operations. The resulting session permissions are the intersection of the role's identity-based policies and the session policies. For more information about session policies, see Session policies. For more information about session tags, see Pass session tags in AWS STS.

Note

The size of the session token that AWS STS API operations return is not fixed. We strongly recommend that you make no assumptions about the maximum size. The typical token size is less than 4096 bytes, but that can vary.

Using AWS STS with AWS Regions

You can send AWS STS API calls either to a global endpoint or to one of the Regional endpoints. If you choose an endpoint closer to you, you can reduce latency and improve the performance of your API calls. You also can choose to direct your calls to an alternative Regional endpoint if you can no longer communicate with the original endpoint. If you are using one of the various AWS SDKs, then use that SDK method to specify a Region before you make the API call. If you manually construct HTTP API requests, then you must direct the request to the correct endpoint yourself. For more information, see the AWS STS section of *Regions and Endpoints* and Manage AWS STS in an AWS Region.

The following are the API operations that you can use to acquire temporary credentials for use in your AWS environment and applications.

Requesting credentials for cross-account delegation and federation through a custom identity broker

The `AssumeRole` API operation is useful for allowing existing IAM users to access AWS resources that they don't already have access to. For example, the user might need access to resources in another AWS account. It is also useful as a means to temporarily gain privileged access—for example, to provide multi-factor authentication (MFA). You must call this API using active credentials. To learn who can call this operation, see Compare AWS STS credentials. For more information, see Create a role to give permissions to an IAM user and Secure API access with MFA.

To request temporary security credentials for cross-account delegation and federation through a custom identity broker

1. Authenticate with your AWS security credentials. This call must be made using valid AWS security credentials.
2. Call the operation `AssumeRole`.

The following example shows a sample request and response using `AssumeRole`. This example request assumes the `demo` role for the specified duration with the included session policy, session tags, external ID, and source identity. The resulting session is named `John-session`.

Example request

```
 https://sts.amazonaws.com/
?Version=2011-06-15
&Action=AssumeRole
&RoleSessionName=John-session
&RoleArn=arn:aws::iam::123456789012:role/demo
&Policy=%7B%22Version%22%3A%222012-10-17%22%2C%22Statement%22%3A%5B%7B%22Sid%22%3A%20%22Stmt1%22%2C%22Effect%22%
&DurationSeconds=1800
&Tags.member.1.Key=Project
&Tags.member.1.Value=Pegasus
&Tags.member.2.Key=Cost-Center
&Tags.member.2.Value=12345
&ExternalId=123ABC
&SourceIdentity=DevUser123
&AUTHPARAMS
```

The policy value shown in the preceding example is the URL-encoded version of the following policy:

```
{"Version":"2012-10-17","Statement":[{"Sid":"Stmt1","Effect":"Allow","Action":"s3:*","Resource":"*"}]}
```

The `AUTHPARAMS` parameter in the example is a placeholder for your *signature*. A signature is the authentication information that you must include with AWS HTTP API requests. We recommend using the AWS SDKs to create API requests, and one benefit of doing so is that the SDKs handle request signing for you. If you must create and sign API requests manually, see Signing AWS Requests By Using Signature Version 4 in the *Amazon Web Services General Reference* to learn how to sign a request.

In addition to the temporary security credentials, the response includes the Amazon Resource Name (ARN) for the federated user and the expiration time of the credentials.

Example response

```
 <AssumeRoleResponse xmlns="https://sts.amazonaws.com/doc/2011-06-15/">
<AssumeRoleResult>
<SourceIdentity>DevUser123</SourceIdentity>
<Credentials>
  <SessionToken>
   AQoDYXdzEPT//////////wEXAMPLEtc764bNrC9SAPBSM22wDOk4x4HIZ8j4FZTwdQW
   LWsKWHGBuFqwAeMicRXmxfpSPfIeoIYRqTflfKD8YUuwthAx7mSEI/qkPpKPi/kMcGd
   QrmGdeehM4IC1NtBmUpp2wUE8phUZampKsburEDy0KPkyQDYwT7WZ0wq5VSXDvp75YU
   9HFvlRd8Tx6q6fE8YQcHNVXAkiY9q6d+xo0rKwT38xVqr7ZD0u0iPPkUL64lIZbqBAz
   +scqKmlzm8FDrypNC9Yjc8fPOLn9FX9KSYvKTr4rvx3iSIlTJabIQwj2ICCR/oLxBA==
  </SessionToken>
  <SecretAccessKey>
   wJalrXUtnFEMI/K7MDENG/bPxRfiCYzEXAMPLEKEY
  </SecretAccessKey>
  <Expiration>2019-07-15T23:28:33.359Z</Expiration>
  <AccessKeyId>AKIAIOSFODNN7EXAMPLE</AccessKeyId>
</Credentials>
<AssumedRoleUser>
  <Arn>arn:aws:sts::123456789012:assumed-role/demo/John</Arn>
  <AssumedRoleId>ARO123EXAMPLE123:John</AssumedRoleId>
</AssumedRoleUser>
<PackedPolicySize>8</PackedPolicySize>
</AssumeRoleResult>
<ResponseMetadata>
<RequestId>c6104cbe-af31-11e0-8154-cbc7ccf896c7</RequestId>
</ResponseMetadata>
</AssumeRoleResponse>
```

Note

An AWS conversion compresses the passed session policies and session tags into a packed binary format that has a separate limit. Your request can fail for this limit even if your plaintext meets the other requirements. The `PackedPolicySize` response element indicates by percentage how close the policies and tags for your request are to the upper size limit.

Requesting credentials through an OIDC provider

The `AssumeRoleWithWebIdentity` API operation returns a set of temporary AWS security credentials in exchange for a JSON Web Token (JWT). This includes public identity providers, such as Login with Amazon, Facebook, Google, and providers that issue JWTs that are compatible with OpenID Connect (OIDC) discovery, such as GitHub actions or Azure Devops. For more information, see OIDC federation.

Note

`AssumeRoleWithWebIdentity` requests are not signed with, and do not require AWS credentials.

Requesting credentials through an OIDC provider

1. Call the operation `AssumeRoleWithWebIdentity`.

   When you call `AssumeRoleWithWebIdentity`, AWS validates the token presented by verifying the digital signature using public keys made available through your IdP's JSON web keyset (JWKS). If the token is valid, and all conditions set forth in the IAM role trust policy are met, 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.

2. Your application may then use the temporary security credentials that were returned in the response to make AWS API calls. This is the same process as making an AWS API call with long-term security credentials. The difference is that you must include the session token, which lets AWS verify that the temporary security credentials are valid.

Your application should cache the credentials returned by AWS STS and refresh them as needed. If your application is built using an AWS SDK, the SDK has credential providers that can handle calling `AssumeRoleWithWebIdentity` and refreshing AWS credentials before they expire. For more information, see AWS SDKs and Tools standardized credential providers in the *AWS SDKs and Tools Reference Guide*.

Requesting credentials through a SAML 2.0 identity provider

The `AssumeRoleWithSAML` API operation returns a set of temporary security credentials for SAML federated principals who are authenticated by your organization's existing identity system. The users must also use SAML

2.0 (Security Assertion Markup Language) to pass authentication and authorization information to AWS. This API operation is useful in organizations that have integrated their identity systems (such as Windows Active Directory or OpenLDAP) with software that can produce SAML assertions. Such an integration provides information about user identity and permissions (such as Active Directory Federation Services or Shibboleth). For more information, see SAML 2.0 federation.

1. Call the operation `AssumeRoleWithSAML`.

   This is an unsigned call, meaning you do not need to authenticate AWS security credentials prior to making the request.

   Note

   A call to `AssumeRoleWithSAML` is not signed (encrypted). Therefore, you should only include optional session policies if the request is transmitted through a trusted intermediary. In this case, someone could alter the policy to remove the restrictions.

2. When you call `AssumeRoleWithSAML`, AWS verifies the authenticity of the SAML assertion. Assuming that the identity provider validates the assertion, AWS returns the following information to you:

   - A set of temporary security credentials. These consist of an access key ID, a secret access key, and a session token.
   - The role ID and the ARN of the assumed role.
   - An `Audience` value that contains the value of the `Recipient` attribute of the `SubjectConfirmationData` element of the SAML assertion.
   - An `Issuer` value that contains the value of the `Issuer` element of the SAML assertion.
   - A `NameQualifier` element that contains a hash value built from the `Issuer` value, the AWS account ID, and the friendly name of the SAML provider. When combined with the `Subject` element, they can uniquely identify the SAML federated principal.
   - A `Subject` element that contains the value of the `NameID` element in the `Subject` element of the SAML assertion.
   - A `SubjectType` element that indicates the format of the `Subject` element. The value can be `persistent`, `transient`, or the full `Format` URI from the `Subject` and `NameID` elements used in your SAML assertion. For information about the `NameID` element's `Format` attribute, see Configure SAML assertions for the authentication response.

3. Use the temporary security credentials returned in the response to make AWS API calls. This is the same process as making an AWS API call with long-term security credentials. The difference is that you must include the session token, which lets AWS verify that the temporary security credentials are valid.

Your app should cache the credentials. By default the credentials expire after an hour. If you are not using the AmazonSTSCredentialsProvider action in the AWS SDK, it's up to you and your app to call `AssumeRoleWithSAML` again. Call this operation to get a new set of temporary security credentials before the old ones expire.

Requesting credentials through a custom identity broker

The `GetFederationToken` API operation returns a set of temporary security credentials for AWS STS federated user principals. This API differs from `AssumeRole` in that the default expiration period is substantially longer (12 hours instead of one hour). Additionally, you can use the `DurationSeconds` parameter to specify a duration for the temporary security credentials to remain valid. The resulting credentials are valid for the specified duration, between 900 seconds (15 minutes) to 129,600 seconds (36 hours). The longer expiration period can help reduce the number of calls to AWS because you do not need to get new credentials as often.

1. Authenticate with the AWS security credentials of your specific IAM user. This call must be made using valid AWS security credentials.
2. Call the operation `GetFederationToken` .

The `GetFederationToken` call returns temporary security credentials that consist of the session token, access key, secret key, and expiration. You can use `GetFederationToken` if you want to manage permissions inside your organization (for example, using the proxy application to assign permissions).

The following example shows a sample request and response that uses `GetFederationToken` . This example request federates the calling user for the specified duration with the session policy ARN and session tags. The resulting session is named `Jane-session` .

Example request

```
 https://sts.amazonaws.com/
?Version=2011-06-15
&Action=GetFederationToken
&Name=Jane-session
&PolicyArns.member.1.arn==arn%3Aaws%3Aiam%3A%3A123456789012%3Apolicy%2FRole1policy
&DurationSeconds=1800
&Tags.member.1.Key=Project
&Tags.member.1.Value=Pegasus
&Tags.member.2.Key=Cost-Center
&Tags.member.2.Value=12345
&AUTHPARAMS
```

The policy ARN shown in the preceding example includes the following URL-encoded ARN:

`arn:aws:iam::123456789012:policy/Role1policy`

Also, note that the `&AUTHPARAMS` parameter in the example is meant as a placeholder for the authentication information. This is the *signature*, which you must include with AWS HTTP API requests. We recommend using the AWS SDKs to create API requests, and one benefit of doing so is that the SDKs handle request signing for you. If you must create and sign API requests manually, go to Signing AWS Requests By Using Signature Version 4 in the *Amazon Web Services General Reference* to learn how to sign a request.

In addition to the temporary security credentials, the response includes the Amazon Resource Name (ARN) for the federated user and the expiration time of the credentials.

Example 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>2019-04-15T23:28:33.359Z</Expiration>
  <AccessKeyId>AKIAIOSFODNN7EXAMPLE;</AccessKeyId>
</Credentials>
<FederatedUser>
  <Arn>arn:aws:sts::123456789012:federated-user/Jean</Arn>
  <FederatedUserId>123456789012:Jean</FederatedUserId>
</FederatedUser>
<PackedPolicySize>4</PackedPolicySize>
</GetFederationTokenResult>
<ResponseMetadata>
<RequestId>c6104cbe-af31-11e0-8154-cbc7ccf896c7</RequestId>
</ResponseMetadata>
</GetFederationTokenResponse>
```

Note

An AWS conversion compresses the passed session policies and session tags into a packed binary format that has a separate limit. Your request can fail for this limit even if your plaintext meets the other requirements. The `PackedPolicySize` response element indicates by percentage how close the policies and tags for your request are to the upper size limit.

AWS recommends that you grant permissions at the resource level (for example, you attach a resource-based policy to an Amazon S3 bucket), you can omit the `Policy` parameter. However, if you do not include a policy for the AWS STS federated user principal, the temporary security credentials will not grant any permissions. In this case, you *must* use resource policies to grant the federated user access to your AWS resources.

For example, assume your AWS account number is 111122223333, and you have an Amazon S3 bucket that you want to allow Susan to access. Susan's temporary security credentials don't include a policy for the bucket. In that case, you would need to ensure that the bucket has a policy with an ARN that matches Susan's ARN, such as `arn:aws:sts::111122223333:federated-user/Susan`.

Requesting credentials for users in untrusted environments

The `GetSessionToken` API operation returns a set of temporary security credentials to an existing IAM user. This is useful for providing enhanced security, such as allowing AWS requests only when MFA is enabled for the IAM user. Because the credentials are temporary, they provide enhanced security when you have an IAM user who accesses your resources through a less secure environment. Examples of less secure environments include a mobile device or web browser.

1. Authenticate with the AWS security credentials of your specific IAM user. This call must be made using valid AWS security credentials.
2. Call the operation `GetSessionToken`.
3. `GetSessionToken` returns temporary security credentials consisting of a session token, an access key ID, and a secret access key.

By default, temporary security credentials for an IAM user are valid for a maximum of 12 hours. But you can request a duration as short as 15 minutes or as long as 36 hours using the `DurationSeconds` parameter. For security reasons, a token for an AWS account root user is restricted to a duration of one hour.

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=1800
&AUTHPARAMS
```

The `AUTHPARAMS` parameter in the example is a placeholder for your *signature*. A signature is the authentication information that you must include with AWS HTTP API requests. We recommend using the AWS SDKs to create API requests, and one benefit of doing so is that the SDKs handle request signing for you. If you must create and

sign API requests manually, go to Signing AWS Requests By Using Signature Version 4 in the *Amazon Web Services General Reference* to learn how to sign a request.

Example 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>
```

Optionally, the `GetSessionToken` request can include `SerialNumber` and `TokenCode` values for AWS multi-factor authentication (MFA) verification. If the provided values are valid, AWS STS provides temporary security credentials that include the state of MFA authentication. The temporary security credentials can then be used to access the MFA-protected API operations or AWS websites for as long as the MFA authentication is valid.

The following example shows a `GetSessionToken` request that includes an MFA verification code and device serial number.

```
 https://sts.amazonaws.com/
?Version=2011-06-15
&Action=GetSessionToken
&DurationSeconds=7200
&SerialNumber=YourMFADeviceSerialNumber
&TokenCode=123456
&AUTHPARAMS
```

Note

The call to AWS STS can be to the global endpoint or to any of the Regional endpoints that you activate your AWS account. For more information, see the AWS STS section of *Regions and Endpoints*.

The `AUTHPARAMS` parameter in the example is a placeholder for your *signature*. A signature is the authentication information that you must include with AWS HTTP API requests. We recommend using the AWS SDKs to create API requests, and one benefit of doing so is that the SDKs handle request signing for you. If you must create and sign API requests manually, see Signing AWS Requests By Using Signature Version 4 in the *Amazon Web Services General Reference* to learn how to sign a request.
