Enabling custom identity broker access to the AWS console

PDF (iam-ug.pdf#id_roles_providers_enable-console-custom-url) | RSS (aws-iam-release-notes.rss)

Filter View All

You can write and run code to create a URL that lets users who sign in to your organization's network securely access the AWS Management Console. The URL includes a sign-in token that you get from AWS and that authenticates the user to AWS.

> **Note**
>
> If your organization uses an identity provider (IdP) that is compatible with SAML, you can set up access to the console without writing code. This works with providers like Microsoft's Active Directory Federation Services or open-source Shibboleth. For details, see Enabling SAML 2.0 federated users to access the AWS Management Console (./id_roles_providers_enable-console-saml.html) .

To enable your organization's users to access the AWS Management Console, you can create a custom *identity broker* that performs the following steps:

1. Verify that the user is authenticated by your local identity system.
2. Call the AWS Security Token Service (AWS STS) AssumeRole (https://docs.aws.amazon.com/STS/latest/APIReference/API_AssumeRole.html) (recommended) or GetFederationToken (https://docs.aws.amazon.com/STS/latest/APIReference/API_GetFederationToken.html) API operations to obtain temporary security credentials for the user. To learn about the different methods that you can use to assume a role, see Using IAM roles (./id_roles_use.html) . To learn how to pass optional session tags when you obtain your security credentials, see Passing session tags in AWS STS (./id_session-tags.html) .
   - If you use one of the `AssumeRole*` API operations to get the temporary security credentials for a role, you can include the `DurationSeconds` parameter in your call. This parameter specifies the duration of your role session, from 900 seconds (15 minutes) up to the maximum session duration setting for the role. When you use `DurationSeconds` in an `AssumeRole*` operation, you must call it as an IAM user with

long-term credentials. Otherwise, the call to the federation endpoint in step 3 fails. To learn how to view or change the maximum value for a role, see View the maximum session duration setting for a role (./id_roles_use.html#id_roles_use_view-role-max-session) .

- If you use the `GetFederationToken` API operation to get the credentials, you can include the `DurationSeconds` parameter in your call. This parameter specifies the duration of your role session. The value can range from 900 seconds (15 minutes) to 129,600 seconds (36 hours). You can make this API call only by using the long-term AWS security credentials of an IAM user. You can also make these calls using AWS account root user credentials, but we do not recommend it. If you make this call as the root user, the default session lasts for one hour. Or you can specify a session from 900 seconds (15 minutes) up to 3,600 seconds (one hour).

3. Call the AWS federation endpoint and supply the temporary security credentials to request a sign-in token.
4. Construct a URL for the console that includes the token:
   - If you use one of the `AssumeRole*` API operations in your URL, you can include the `SessionDuration` HTTP parameter. This parameter specifies the duration of the console session, from 900 seconds (15 minutes) to 43200 seconds (12 hours).
   - If you use the `GetFederationToken` API operation in your URL, you can include the `DurationSeconds` parameter. This parameter specifies the duration of the federated console session. The value can range from 900 seconds (15 minutes) to 129,600 seconds (36 hours).

   > **Note**
   >
   > Do not use the `SessionDuration` HTTP parameter if you got the temporary credentials with `GetFederationToken`. Doing so will cause the operation to fail.

5. Give the URL to the user or invoke the URL on the user's behalf.

The URL that the federation endpoint provides is valid for 15 minutes after it is created. This differs from the duration (in seconds) of the temporary security credential session that is associated with the URL. Those credentials are valid for the duration you specified when you created them, starting from the time they were created.

> **Important**
>
> The URL grants access to your AWS resources through the AWS Management Console if you have enabled permissions in the associated temporary security credentials. For this reason, you should treat the URL as a secret. We recommend returning the URL through

a secure redirect, for example, by using a 302 HTTP response status code over an SSL connection. For more information about the 302 HTTP response status code, go to RFC 2616, section 10.3.3 (https://tools.ietf.org/html/rfc2616#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 (https://aws.amazon.com/code/4001165270590826) in the *AWS Sample Code & Libraries*.

To complete these tasks, you can use the HTTPS Query API for AWS Identity and Access Management (IAM) (https://docs.aws.amazon.com/IAM/latest/APIReference/) and the AWS Security Token Service (AWS STS) (https://docs.aws.amazon.com/STS/latest/APIReference/) . Or, you can use programming languages, such as Java, Ruby, or C#, along with the appropriate AWS SDK (https://aws.amazon.com/tools/) . Each of these methods is described in the following sections.

Topics

Example code using IAM query API operations

You can construct a URL that gives your federated users direct access to the AWS Management Console. This task uses the IAM and AWS STS HTTPS Query API. For more information about making query requests, see Making Query Requests (https://docs.aws.amazon.com/IAM/latest/UserGuide/IAM_UsingQueryAPI.html) .

> **Note**
>
> The following procedure contains examples of text strings. To enhance readability, line breaks have been added to some of the longer examples. When you create these strings for your own use, you should omit any line breaks.

To give a federated user access to your resources from the AWS Management Console

1. Authenticate the user in your identity and authorization system.
2. Obtain temporary security credentials for the user. The temporary credentials consist of an access key ID, a secret access key, and a session token. For more information about creating

temporary credentials, see Temporary security credentials in IAM (./id_credentials_temp.html) .

To get temporary credentials, you call either the AWS STS AssumeRole (https://docs.aws.amazon.com/STS/latest/APIReference/API_AssumeRole.html) API (recommended) or the GetFederationToken (https://docs.aws.amazon.com/STS/latest/APIReference/API_GetFederationToken.html) API. For more information about the differences between these API operations, see Understanding the API Options for Securely Delegating Access to Your AWS Account (http://aws.amazon.com/blogs/security/understanding-the-api-options-for-securely-delegating-access-to-your-aws-account) in the AWS Security Blog.

> **Important**
>
> When you use the GetFederationToken (https://docs.aws.amazon.com/STS/latest/APIReference/API_GetFederationToken.html) API to create temporary security credentials, you must specify the permissions that the credentials grant to the user who assumes the role. For any of the API operations that begin with `AssumeRole*`, you use an IAM role to assign permissions. For the other API operations, the mechanism varies with the API. For more details, see Controlling permissions for temporary security credentials (./id_credentials_temp_control-access.html) . Additionally, if you use the `AssumeRole*` API operations, you must call them as an IAM user with long-term credentials. Otherwise, the call to the federation endpoint in step 3 fails.

3. After you obtain the temporary security credentials, build them into a JSON session string to exchange them for a sign-in token. The following example shows how to encode the credentials. You replace the placeholder text with the appropriate values from the credentials that you receive in the previous step.

```
{"sessionId":"*** temporary access key ID ***",
"sessionKey":"*** temporary secret access key ***",
"sessionToken":"*** session token ***"}
```

4. URL encode (https://en.wikipedia.org/wiki/Percent-encoding) the session string from the previous step. Because the information that you are encoding is sensitive, we recommend that you avoid using a web service for this encoding. Instead, use a locally installed function or feature in your development toolkit to securely encode this information. You can use the `urllib.quote_plus` function in Python, the `URLEncoder.encode` function in Java, or the `CGI.escape` function in Ruby. See the examples later in this topic.

5.

> **Note**
>
> AWS supports POST requests here.

Send your request to the AWS federation endpoint:

`https://` *`region-code`* `.signin.aws.amazon.com/federation`

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

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

The request must include the `Action` and `Session` parameters, and (optionally) if you used an `AssumeRole*` (https://docs.aws.amazon.com/STS/latest/APIReference/API_AssumeRole.html) API operation, a `SessionDuration` HTTP parameter as shown in the following example.

```
Action = getSigninToken
SessionDuration = time in seconds
Session = *** the URL encoded JSON string created in steps 3 &
4 ***
```

> **Note**
>
> The following instructions in this step only work using GET requests.

The `SessionDuration` HTTP parameter specifies the duration of the console session. This is separate from the duration of the temporary credentials that you specify using the `DurationSeconds` parameter. You can specify a `SessionDuration` maximum value of 43,200 (12 hours). If the `SessionDuration` parameter is missing, then the session defaults to the duration of the credentials that you retrieved from AWS STS in step 2 (which defaults to one hour). See the documentation for the `AssumeRole` API (https://docs.aws.amazon.com/STS/latest/APIReference/API_AssumeRole.html) for details about how to specify a duration using the `DurationSeconds` parameter. The ability to create a console session that is longer than one hour is intrinsic to the `getSigninToken` operation of the federation endpoint.

> **Note**
>
> Do not use the `SessionDuration` HTTP parameter if you got the temporary credentials with `GetFederationToken`. Doing so will cause the operation to fail.

When you enable console sessions with an extended duration, you increase the risk of credential exposure. To help you mitigate this risk, you can immediately disable the active console sessions for any role by choosing **Revoke Sessions** on the **Role Summary** IAM console page. For more information, see Revoking IAM role temporary security credentials (./id_roles_use_revoke-sessions.html) .

The following is an example of what your request might look like. The lines are wrapped here for readability, but you should submit it as a one-line string.

```
https://signin.aws.amazon.com/federation
?Action=getSigninToken
&SessionDuration=1800
&Session=%7B%22sessionId%22%3A+%22ASIAJUMHIZPTOKTBMK5A%22%2C+%2
2sessionKey%22
%3A+%22LSD7LWI%2FL%2FN%2BgYpan5QFz0XUpc8s7HYjRsgcsrsm%22%2C+%22
sessionToken%2
2%3A+%22FQoDYXdzEBQaDLbj3VWv2u50NN%2F3yyLSASwYtWhPnGPMNmzZFfZsL
0Qd3vtYHw5A5dW
AjOsrkdPkghomIe3mJip5%2F0djDBbo7SmO%2FENDEiCdpsQKodTpleKA8xQq0C
wFg6a69xdEBQT8
FipATnLbKoyS4b%2FebhnsTUjZZQWp0wXXqFF7gSm%2FMe2tXe0jzsdP0O12obe
z9lijPSdF1k2b5
PfGhiuyAR9aD5%2BubM0pY86fKex1qsytjvyTbZ9nXe6DvxVDcnCOhOGETJ7XFk
SFdH0v%2FYR25C
UAhJ3nXIkIbG7Ucv9cOEpCf%2Fg23ijRgILIBQ%3D%3D%22%7D
```

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

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

6. **Note**

   AWS supports POST requests here.

Finally, create the URL that your federated users can use to access the AWS Management Console. The URL is the same federation URL endpoint that you used in Step 5 (#STSConsoleLink_manual_step5) , plus the following parameters:

```
?Action = login
&Issuer = *** the form-urlencoded URL for your internal sign-in
page ***
&Destination = *** the form-urlencoded URL to the desired AWS
console page ***
&SigninToken = *** the value of SigninToken received in the
previous step ***
```

> **Note**
>
> The following instructions in this step only work using GET API.

The following example shows what the final URL might look like. The URL is valid for 15 minutes from the time it is created. The temporary security credentials and console session embedded within the URL are valid for the duration you specify in the `SessionDuration` HTTP parameter when you initially request them.

```
https://signin.aws.amazon.com/federation
?Action=login
&Issuer=https%3A%2F%2Fexample.com
&Destination=https%3A%2F%2Fconsole.aws.amazon.com%2F
&SigninToken=VCQgs5qZZt3Q6fn8Tr5EXAMPLEmLnwB7JjUc-
SHwnUUWabcRdnWsi4DBn-dvC
CZ85wrD0nmldUcZEXAMPLE-vXYH4Q__mleuF_W2BE5HYexbe9y4Of-
kje53SsjNNecATfjIzpW1
WibbnH6YcYRiBoffZBGExbEXAMPLE5aiKX4THWjQKC6gg6alHu6JFrnOJoK3dtP
6I9a6hi6yPgm
iOkPZMmNGmhsvVxetKzr8mx3pxhHbMEXAMPLETv1pij0rok3IyCR2YVcIjqwfWv
32HU2Xlj471u
3fU6uOfUComeKiqTGX974xzJOZbdmX_t_lLrhEXAMPLEDDIisSnyHGw2xaZZqud
m4mo2uTDk9Pv
9l5K0ZCqIgEXAMPLEcA6tgLPykEWGUyH6BdSC6166n4M4JkXIQgac7_7821Yqix
sNxZ6rsrpzwf
nQoS14O7R0eJCCJ684EXAMPLEZRdBNnuLbUYpz2Iw3vIN0tQgOujwnwydPscM9F
7foaEK3jwMkg
Apeb1-
6L_OB12MZhuFxx55555EXAMPLEhyETEd4ZulKPdXHkgl6T9ZkIlHz2Uy1RUTUhh
UxNtSQ
nWc5xkbBoEcXqpoSIeK7yhje9Vzhd61AEXAMPLElbWeouACEMG6-
```

```
Vd3dAgFYd6i5FYoyFrZLWvm
0LSG7RyYKeYN5VIzUk3YWQpyjP0RiT5KUrsUi-NEXAMPLExMOMdoODBEgKQsk-
iu2ozh6r8bxwC
RNhujg
```

Example code using Python

The following examples show how to use Python to programmatically construct a URL that gives federated users direct access to the AWS Management Console. There are two examples:

- Federate via GET requests to AWS
- Federate via POST requests to AWS

Both examples use the the AWS SDK for Python (Boto3) (https://aws.amazon.com/tools/) and AssumeRole (https://docs.aws.amazon.com/STS/latest/APIReference/API_AssumeRole.html) API to obtain temporary security credentials.

Use GET Requests

```
import urllib, json, sys
import requests # 'pip install requests'
import boto3 # AWS SDK for Python (Boto3) 'pip install boto3'

# Step 1: Authenticate user in your own identity system.

# Step 2: Using the access keys for an IAM user in your AWS
account,
# call "AssumeRole" to get temporary access keys for the federated
user

# Note: Calls to AWS STS AssumeRole must be signed using the access
key ID
# and secret access key of an IAM user or using existing temporary
credentials.
# The credentials can be in Amazon EC2 instance metadata, in
environment variables,
# or in a configuration file, and will be discovered automatically
by the
```

```
# client('sts') function. For more information, see the Python SDK
docs:
# http://boto3.readthedocs.io/en/latest/reference/services/sts.html
#
http://boto3.readthedocs.io/en/latest/reference/services/sts.html#S
TS.Client.assume_role
sts_connection = boto3.client('sts')

assumed_role_object = sts_connection.assume_role(
    RoleArn="arn:aws:iam::account-id:role/ROLE-NAME",
    RoleSessionName="AssumeRoleSession",
)

# Step 3: Format resulting temporary credentials into JSON
url_credentials = {}
url_credentials['sessionId'] =
assumed_role_object.get('Credentials').get('AccessKeyId')
url_credentials['sessionKey'] =
assumed_role_object.get('Credentials').get('SecretAccessKey')
url_credentials['sessionToken'] =
assumed_role_object.get('Credentials').get('SessionToken')
json_string_with_temp_credentials = json.dumps(url_credentials)

# Step 4. Make request to AWS federation endpoint to get sign-in
token. Construct the parameter string with
# the sign-in action request, a 12-hour session duration, and the
JSON document with temporary credentials
# as parameters.
request_parameters = "?Action=getSigninToken"
request_parameters += "&SessionDuration=43200"
if sys.version_info[0] < 3:
    def quote_plus_function(s):
        return urllib.quote_plus(s)
else:
    def quote_plus_function(s):
        return urllib.parse.quote_plus(s)
request_parameters += "&Session=" +
quote_plus_function(json_string_with_temp_credentials)
request_url = "https://signin.aws.amazon.com/federation" +
request_parameters
```

```
r = requests.get(request_url)
# Returns a JSON document with a single element named SigninToken.
signin_token = json.loads(r.text)

# Step 5: Create URL where users can use the sign-in token to sign
in to
# the console. This URL must be used within 15 minutes after the
# sign-in token was issued.
request_parameters = "?Action=login"
request_parameters += "&Issuer=Example.org"
request_parameters += "&Destination=" +
quote_plus_function("https://console.aws.amazon.com/")
request_parameters += "&SigninToken=" + signin_token["SigninToken"]
request_url = "https://signin.aws.amazon.com/federation" +
request_parameters

# Send final URL to stdout
print (request_url)
```

Use POST Requests

```
import urllib, json, sys
import requests # 'pip install requests'
import boto3 # AWS SDK for Python (Boto3) 'pip install boto3'
import os
from selenium import webdriver # 'pip install selenium', 'brew
install chromedriver'

# Step 1: Authenticate user in your own identity system.

# Step 2: Using the access keys for an IAM user in your AAWS
account,
# call "AssumeRole" to get temporary access keys for the federated
user

# Note: Calls to AWS STS AssumeRole must be signed using the access
key ID
# and secret access key of an IAM user or using existing temporary
credentials.
```

```
# The credentials can be in Amazon EC2 instance metadata, in
environment variables,
# or in a configuration file, and will be discovered automatically
by the
# client('sts') function. For more information, see the Python SDK
docs:
# http://boto3.readthedocs.io/en/latest/reference/services/sts.html
#
http://boto3.readthedocs.io/en/latest/reference/services/sts.html#S
TS.Client.assume_role
if sys.version_info[0] < 3:
    def quote_plus_function(s):
        return urllib.quote_plus(s)
else:
    def quote_plus_function(s):
        return urllib.parse.quote_plus(s)

sts_connection = boto3.client('sts')

assumed_role_object = sts_connection.assume_role(
    RoleArn="arn:aws:iam::account-id:role/ROLE-NAME",
    RoleSessionName="AssumeRoleDemoSession",
)

# Step 3: Format resulting temporary credentials into JSON
url_credentials = {}
url_credentials['sessionId'] =
assumed_role_object.get('Credentials').get('AccessKeyId')
url_credentials['sessionKey'] =
assumed_role_object.get('Credentials').get('SecretAccessKey')
url_credentials['sessionToken'] =
assumed_role_object.get('Credentials').get('SessionToken')
json_string_with_temp_credentials = json.dumps(url_credentials)

# Step 4. Make request to AWS federation endpoint to get sign-in
token. Construct the parameter string with
# the sign-in action request, a 12-hour session duration, and the
JSON document with temporary credentials
# as parameters.
request_parameters = {}
```

```
request_parameters['Action'] = 'getSigninToken'
request_parameters['SessionDuration'] = '43200'
request_parameters['Session'] = json_string_with_temp_credentials

request_url = "https://signin.aws.amazon.com/federation"
r = requests.post( request_url, data=request_parameters)

# Returns a JSON document with a single element named SigninToken.
signin_token = json.loads(r.text)

# Step 5: Create a POST request where users can use the sign-in
token to sign in to
# the console. The POST request must be made within 15 minutes
after the
# sign-in token was issued.
request_parameters = {}
request_parameters['Action'] = 'login'
request_parameters['Issuer']='Example.org'
request_parameters['Destination'] =
'https://console.aws.amazon.com/'
request_parameters['SigninToken'] =signin_token['SigninToken']

jsrequest = '''
var form = document.createElement('form');
form.method = 'POST';
form.action = '{request_url}';
request_parameters = {request_parameters}
for (var param in request_parameters) {{
    if (request_parameters.hasOwnProperty(param)) {{
        const hiddenField = document.createElement('input');
        hiddenField.type = 'hidden';
        hiddenField.name = param;
        hiddenField.value = request_parameters[param];
        form.appendChild(hiddenField);
    }}
}}
document.body.appendChild(form);
form.submit();
'''.format(request_url=request_url,
request_parameters=request_parameters)
```

```
driver = webdriver.Chrome()
driver.execute_script(jsrequest);
```

Example code using Java

The following example shows how to use Java to programmatically construct a URL that gives federated users direct access to the AWS Management Console. The following code snippet uses the AWS SDK for Java (http://aws.amazon.com/documentation/sdkforjava/) .

```
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.GetFederationTokenReques
t;
import
com.amazonaws.services.securitytoken.model.GetFederationTokenResult
;


/* Calls to AWS STS API operations must be signed using the access
key ID
   and secret access key of an IAM user or using existing temporary
   credentials. The credentials should not be embedded in code. For
   this example, the code looks for the credentials in a
   standard configuration file.
*/
AWSCredentials credentials =
```

```
  new PropertiesCredentials(

AwsConsoleApp.class.getResourceAsStream("AwsCredentials.properties"
));

AWSSecurityTokenServiceClient stsClient =
  new AWSSecurityTokenServiceClient(credentials);

GetFederationTokenRequest getFederationTokenRequest =
  new GetFederationTokenRequest();
getFederationTokenRequest.setDurationSeconds(1800);
getFederationTokenRequest.setName("UserName");

// A sample policy for accessing Amazon Simple Notification Service
(Amazon SNS) in the console.

String policy = "{\"Version\":\"2012-10-17\",\"Statement\":
[{\"Action\":\"sns:*\"," +
  "\"Effect\":\"Allow\",\"Resource\":\"*\"}]}";

getFederationTokenRequest.setPolicy(policy);

GetFederationTokenResult federationTokenResult =
  stsClient.getFederationToken(getFederationTokenRequest);

Credentials federatedCredentials =
federationTokenResult.getCredentials();

// The issuer parameter specifies your internal sign-in
// page, for example https://mysignin.internal.mycompany.com/.
// The console parameter specifies the URL to the destination
console of the
// AWS Management Console. This example goes to Amazon SNS.
// The signin parameter is the URL to send the request to.

String issuerURL = "https://mysignin.internal.mycompany.com/";
String consoleURL = "https://console.aws.amazon.com/sns";
String signInURL = "https://signin.aws.amazon.com/federation";

// Create the sign-in token using temporary credentials,
```

```
// including the access key ID,  secret access key, and session
token.
String sessionJson = String.format(
  "{\"%1$s\":\"%2$s\",\"%3$s\":\"%4$s\",\"%5$s\":\"%6$s\"}",
  "sessionId", federatedCredentials.getAccessKeyId(),
  "sessionKey", federatedCredentials.getSecretAccessKey(),
  "sessionToken", federatedCredentials.getSessionToken());

// Construct the sign-in request with the request sign-in token
action, a
// 12-hour console session duration, and the JSON document with
temporary
// credentials as parameters.

String getSigninTokenURL = signInURL +
                           "?Action=getSigninToken" +
                           "&DurationSeconds=43200" +
                           "&SessionType=json&Session=" +
                           URLEncoder.encode(sessionJson,"UTF-8");

URL url = new URL(getSigninTokenURL);

// Send the request to the AWS federation endpoint to get the sign-
in token
URLConnection conn = url.openConnection ();

BufferedReader bufferReader = new BufferedReader(new
  InputStreamReader(conn.getInputStream()));
String returnContent = bufferReader.readLine();

String signinToken = new
JSONObject(returnContent).getString("SigninToken");

String signinTokenParameter = "&SigninToken=" +
URLEncoder.encode(signinToken,"UTF-8");

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

```
String issuerParameter = "&Issuer=" + URLEncoder.encode(issuerURL,
"UTF-8");

// Finally, present the completed URL for the AWS console session
to the user

String destinationParameter = "&Destination=" +
URLEncoder.encode(consoleURL,"UTF-8");
String loginURL = signInURL + "?Action=login" +
                     signinTokenParameter + issuerParameter +
destinationParameter;
```

Example showing how to construct the URL (Ruby)

The following example shows how to use Ruby to programmatically construct a URL that gives federated users direct access to the AWS Management Console. This code snippet uses the AWS SDK for Ruby (http://aws.amazon.com/documentation/sdkforruby/) .

```
require 'rubygems'
require 'json'
require 'open-uri'
require 'cgi'
require 'aws-sdk'

# Create a new STS instance
#
# Note: Calls to AWS STS API operations must be signed using an
access key ID
# and secret access key. The credentials can be in EC2 instance
metadata
# or in environment variables and will be automatically discovered
by
# the default credentials provider in the AWS Ruby SDK.
sts = Aws::STS::Client.new()

# The following call creates a temporary session that returns
# temporary security credentials and a session token.
```

```
# The policy grants permissions to work
# in the AWS SNS console.

session = sts.get_federation_token({
  duration_seconds: 1800,
  name: "UserName",
  policy: "{\"Version\":\"2012-10-17\",\"Statement\":
{\"Effect\":\"Allow\",\"Action\":\"sns:*\",\"Resource\":\"*\"}}",
})

# The issuer value is the URL where users are directed (such as
# to your internal sign-in page) when their session expires.
#
# The console value specifies the URL to the destination console.
# This example goes to the Amazon SNS console.
#
# The sign-in value is the URL of the AWS STS federation endpoint.
issuer_url = "https://mysignin.internal.mycompany.com/"
console_url = "https://console.aws.amazon.com/sns"
signin_url = "https://signin.aws.amazon.com/federation"

# Create a block of JSON that contains the temporary credentials
# (including the access key ID, secret access key, and session
token).
session_json = {
  :sessionId => session.credentials[:access_key_id],
  :sessionKey => session.credentials[:secret_access_key],
  :sessionToken => session.credentials[:session_token]
}.to_json

# Call the federation endpoint, passing the parameters
# created earlier and the session information as a JSON block.
# The request returns a sign-in token that's valid for 15 minutes.
# Signing in to the console with the token creates a session
# that is valid for 12 hours.
get_signin_token_url = signin_url +
                       "?Action=getSigninToken" +
                       "&SessionType=json&Session=" +
                       CGI.escape(session_json)
```

```
returned_content = URI.parse(get_signin_token_url).read

# Extract the sign-in token from the information returned
# by the federation endpoint.
signin_token = JSON.parse(returned_content)['SigninToken']
signin_token_param = "&SigninToken=" + CGI.escape(signin_token)

# Create the URL to give to the user, which includes the
# sign-in token and the URL of the console to open.
# The "issuer" parameter is optional but recommended.
issuer_param = "&Issuer=" + CGI.escape(issuer_url)
destination_param = "&Destination=" + CGI.escape(console_url)
login_url = signin_url + "?Action=login" + signin_token_param +
  issuer_param + destination_param
```