Single Sign-On: Integrating AWS, OpenLDAP, and Shibboleth

A Step-by-Step Walkthrough

Matthew Berry, AWS Identity and Access Management

April 2015

Notices

This document is provided for informational purposes only. It represents AWS's current product offerings and practices as of the date of issue of this document, which are subject to change without notice. Customers are responsible for making their own independent assessment of the information in this document and any use of AWS's products or services, each of which is provided "as is" without warranty of any kind, whether express or implied. This document does not create any warranties, representations, contractual commitments, conditions or assurances from AWS, its affiliates, suppliers or licensors. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

Contents

Abstract

AWS Identity and Access Management (IAM) is a web service from Amazon Web Services (AWS) for managing users and user permissions in AWS. Outside the AWS cloud, administrators of corporate systems rely on the Lightweight Directory Access Protocol (LDAP)[1] to manage identities. By using role-based access control (RBAC) and Security Assertion Markup Language (SAML) 2.0, corporate IT systems administrators can bridge the IAM and LDAP systems and simplify identity and permissions management across on-premises and cloud-based infrastructures.

Introduction

In November 2013, the IAM team expanded identity federation[2] to support SAML 2.0. Instead of recreating existing user data in AWS so that users in your organization can access AWS, you can use AWS support for SAML to federate user identities into AWS. For example, in many universities professors can help students take advantage of AWS resources via the students' university accounts. Step-by-step instructions walk you through the use of AWS SAML 2.0 support with OpenLDAP, which is an implementation of LDAP. This walkthrough depicts a fictitious university moving to OpenLDAP. Because the university makes heavy use of Shibboleth identity provider (IdP) software, you will learn how to use Shibboleth as the IdP.

You will also learn the entire process of setting up LDAP. If your organization already has a functional LDAP implementation, you can review the schema and

then skip to the Install Tomcat[3] and Install Shibboleth IdP[4] sections. Likewise, if your organization already has Shibboleth in production, you can skip to the Configure Shibboleth IdP[5] section.

Assumptions and Prerequisites

This walkthrough describes using a Linux Ubuntu operating system and makes the following assumptions about your familiarity with Ubuntu and with services from AWS, such as Amazon Elastic Compute Cloud (Amazon EC2):

- You know enough about Linux to move between directories, use an editor (such as Vim), and run script commands.
- You have a Secure Shell (SSH) tool, such as OpenSSH or PuTTY, installed on your computer, and you know how to connect to a running Amazon EC2 instance. For a list of SSH tools, see Connect to Your Linux Instance[6] in the Amazon EC2 documentation.
- You have a basic understanding of what LDAP is and what an LDAP schema looks like.

LDAP Schema and Roles

A fictitious university called Example University is organized as shown in Figure 1. This university assigns a unique identifier (*uid*) to each individual, more commonly referred to as a *user name*. Each individual is also part of one or more organizational units (*OU* or *OrgUnit*). In our fictitious university, OUs correspond to departments, and one special OU named "People" contains everyone.

Each individual has a primary OU. The primary OU for everyone except managers is the People OU. The primary OU for managers is the department they manage.

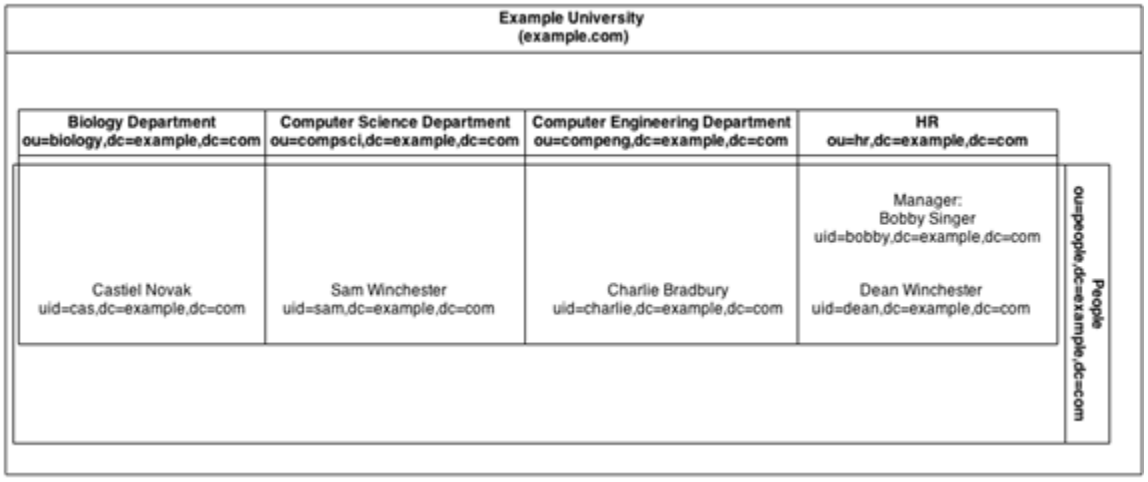

Figure 1: Schema for Example University

Software

For the example, use the following software. Although Ubuntu 14.04 Long Term Support (LTS) is illustrated, the instructions apply to most versions of Ubuntu and Linux (perhaps with minor modifications). In general the procedures work in Microsoft Windows or OS X from Apple, but they require alternate installation and configuration guides for OpenLDAP and Java virtual machine, which this walkthrough does not address.

Function	Software and version
Operating system	Ubuntu 14.04 LTS
Java virtual machine	OpenJDK 7u25 (IcedTea 2.3.10)
Web server	Apache Tomcat 7.0.59
Identity provider	Shibboleth IdP 2.4
Directory	SLAPD (OpenLDAP 2.4.28)

Step 1: Prepare the Operating System

These steps begin with an Amazon EC2 instance so that you can see a completely clean installation of all components.

The demo uses a t2.micro instance because it is free-tier eligible[7] (it will not cost you anything) and because this example installation does not serve any production traffic. You can complete this walkthrough with a t2.micro instance and stay in the free tier. You can use a larger instance size if you want. It makes no difference to the illustrated functionality, and larger sizes run faster. But note that you will be charged at standard rates if you use instances that are not in the free tier.

If you are new to Amazon EC2, you might want to read Getting Started with Amazon EC2 Linux Instances[8] for context before you begin.

Launch a New Amazon EC2 Instance

1. Sign in to the AWS Management Console and then go to the Amazon EC2 console.
2. Click **Launch Instance**, find **Ubuntu Server 14.04 LTS (HVM), SSD Volume Type**, and then click **Select**.
3. Select the **t2.micro** instance, which is the default.
4. Click through the **Next** buttons until you get to **Step 6: Configure Security Group**.

   Note: Restrict the IP address range in this step to match your organization's IP address prefix, or use the **My IP** option.
5. Click **Add Rule** and then select **HTTPS**. This opens up port 443 for SSL traffic.

Note: Restrict the IP address range in this step to match your organization's IP address prefix, or use the **My IP** option.

6. When you are finished, click **Review and Launch**, and then click **Launch**.
7. When prompted, create a new key pair for logging in to the Ubuntu instance. Give it a name (for example, `ShibbolethDemo`), and then download and save the key pair. See Figure 2. Then click **Launch Instances**.

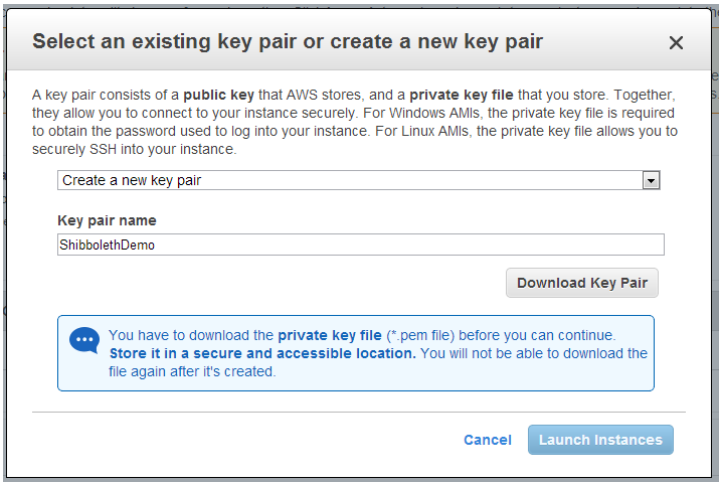

Figure 2: Select an Existing Key Pair or Create a New Key Pair

Important: Be sure to download your key pair. Otherwise, you will not be able to access your instance. For information about how to connect to an Amazon EC2 instance using SSH, see Connect to Your Linux Instance.[9]

8. Click **View Instances**. When the instance is running, find and copy the following values for the instance, which you'll need later:
   - The instance ID.
   - The public DNS of the instance.
   - The public IP address of the instance.

   You can find all of these values in the Amazon EC2 console when you select your instance, as shown in Figure 3.

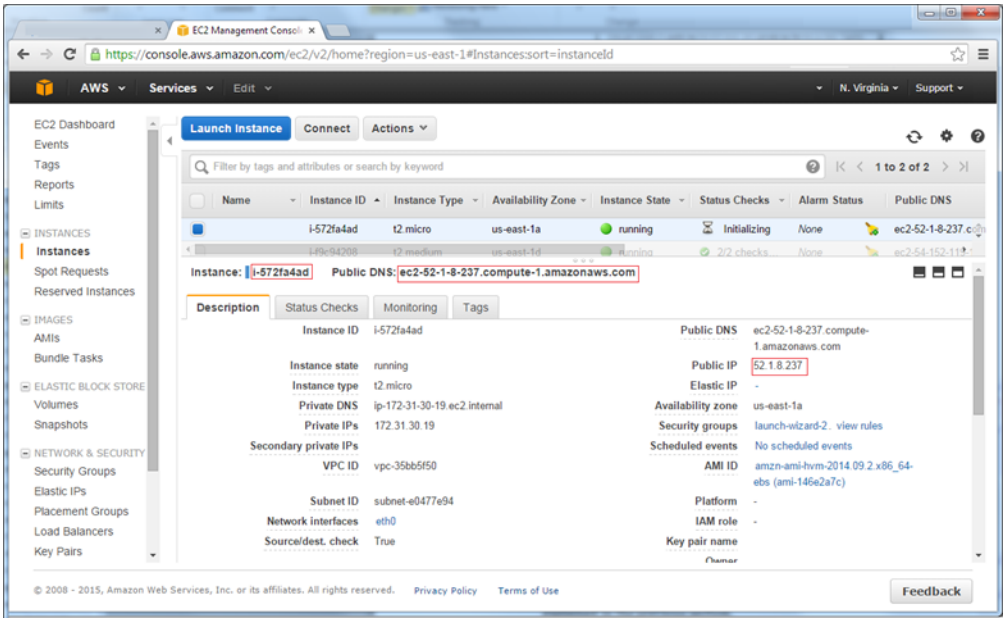

Figure 3: EC2 Instance Details, Showing Instance ID, Public DNS, and IP Address

Update Local Hosts File

In this walkthrough, various configuration values reference the DNS example.com or idp.example.com. Each Amazon EC2 instance has a unique IP address and DNS that are assigned when the instance starts, so you must update the hosts file on your local computer so that example.com and idp.example.com resolve to the IP address of your Amazon EC2 instance.

1. Make sure you know the public IP address of your Amazon EC2 instance, as explained in the previous section.
2. Open the hosts file on your local computer. Editing this file requires administrative privileges. These are the usual locations of the hosts file:
   - Windows: `%windir%\System32\drivers\etc\hosts`
   - Linux: `/etc/hosts`
   - Mac: `/private/etc/hosts`
3. Add the following mappings to the hosts file, using the public IP address of your own Amazon EC2 instance. When you are done, save and close the file.

```
nn.nnnn.nnn.nn example.com
nn.nnnn.nnn.nn idp.example.com
```

Create Directories

Using your SSH tool (OpenSSH, PuTTY, etc.), connect to your Amazon EC2 instance. Create directories for Tomcat, Shibboleth, and the demo files by running the following commands.

```
cd /home/ubuntu/
mkdir -p /home/ubuntu/server/tomcat/conf/Catalina/localhost
mkdir -p /home/ubuntu/server/tomcat/endorsed
mkdir /home/ubuntu/server/shibidp
mkdir -p /home/ubuntu/installers/shibidp
```

Step 2: Install and Configure OpenLDAP

OpenLDAP is an open-source implementation of the Lightweight Directory Access Protocol (LDAP).[10] This walkthrough assumes basic knowledge of LDAP and explains only what is required to complete it.

About LDAP

A small set of primitives that can be combined into a complex hierarchy of objects and attributes defines LDAP.

The core element of the LDAP system is an object, which consists of a key-value pair. Objects can represent anything that needs an identity in the LDAP system, such as people, printers, or buildings. Because you can reuse keys, sets of key-value pairs are grouped into object classes. These object classes are included by using special object class attributes, as shown in Figure 4.

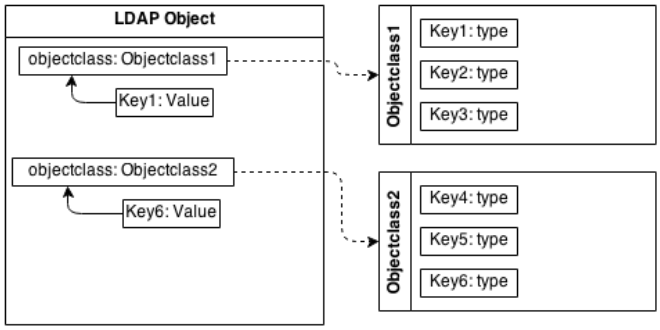

Figure 4: Including Object Classes with Special Object Class Attributes

Object classes make LDAP extensible. All the people at an organization have a core set of attributes that they share, such as name, address, phone, office, department, and job level. You can wrap these attributes into an object class so that the definition of a person in the directory can reference the object class and automatically get all the common attributes defined by it. Figure 5 shows an example of an object class.

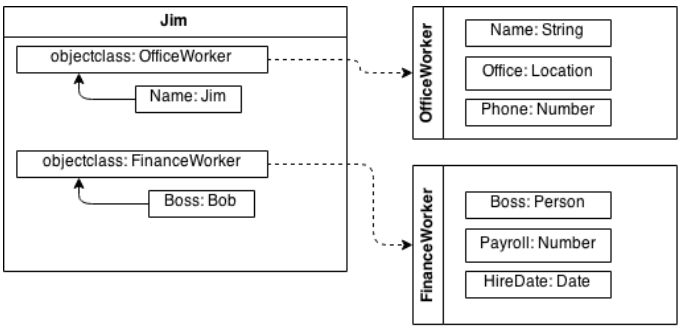

Figure 5: An Example of an Object Class

Install OpenLDAP

For this walkthrough, you need to install OpenLDAP on the Amazon EC2 instance that you launched.

1. Log in to the Amazon EC2 instance and enter the following commands to download and install OpenLDAP.

```
sudo apt-get -y update && sudo apt-get -y upgrade
```

   This command updates the package list on the host. The second half of the command updates all the packages on the host to the newest versions.

2. Type the following command to install OpenLDAP.

```
sudo apt-get -y install slapd ldap-utils
```

3. Type the following commands to set up shortcuts (aliases) for working with OpenLDAP.

```
echo "alias ldapsearch='ldapsearch -H ldapi:/// -x -W '" >>
~/.bashrc
echo "alias ldapmodify='ldapmodify -H ldapi:/// -x -W '" >>
~/.bashrc
# Adding $LDAP_ADMIN to either of the ldap commands binds
to admin account
echo "export LDAP_ADMIN='-D cn=admin,dc=example,dc=com '"
>> ~/.bashrc
source ~/.bashrc
```

   These commands add aliases to the `~/.bashrc` file, which is a file that contains commands that run each time the user signs in. The shortcuts add some common parameters to `ldapsearch` and `ldapmodify`, the two

most common LDAP utilities. The parameters for these commands are as follows:

- `-H ldapi:///` tells the command where the directory is located.
- `-x` tells the command to use simple authentication.
- `-W` tells the command to ask for the password (instead of listing it on the command line).
- `-D cn=admin,dc=example,dc=com` is a set of parameters to indicate that LDAP should run as an administrator.

4. Type the following command to tell the package manager to reconfigure OpenLDAP.

```
sudo dpkg-reconfigure slapd
```

When the command runs, you see the following prompts. Respond as noted.

- `Omit OpenLDAP server configuration?`
  Type `No`. You want to have a blank directory created.

- `DNS domain name:`
  Type `example.com`. You use this to construct the hierarchy of the LDAP directory. Use this domain for this walkthrough, because other aspects of the configuration depend on this domain name.

- `Organization name:`
  Type any name. This value is not used.

- `Administrator password: (and confirmation)`
  This is the LDAP administrator password. For the purposes of this walkthrough, use `password`. For production systems, consult your security best practices. You will need the password when you make changes to the LDAP configuration later.

- `Database backend to use:`
  This lets you specify the storage back end for LDAP information. Type `HDB`.

- `Do you want the database to be removed when slapd is purged?`
  Type `Yes`. This is a safety measure in case you purge a setup and start over. In that case, if you type `Yes`, the directory is backed up rather than deleted.

- `Move old database?`
  Type `Yes`. This is part of the safety measure from the previous prompt. By answering `Yes`, you cause OpenLDAP to make a backup of the existing directory before wiping it.

- `Allow LDAPv2 protocol?`
  Type `No`. LDAPv2 is deprecated.

Download LDAP Sample Data

For this walkthrough you need some data in the LDAP data store. For convenience the walkthrough provides files that contain sample data. To download these directly to the Amazon EC2 instance, run the following script inside your instance.

```
wget -O '/home/ubuntu/examples.tar.gz'
'https://s3.amazonaws.com/awsiammedia/public/sample/OpenLDA
PandShibboleth/examples.tar.gz'
tar -xf /home/ubuntu/examples.tar.gz
```

Configure OpenLDAP

Because LDAP is text based, it is easy to back up the directory and share attribute definitions (called *schemas*). However, this paper does not focus on LDAP, so it does not go into detail about the text format used to interact with LDAP. You just need to know that Lightweight Directory Interchange Format (LDIF) is a text-based export/import format for LDAP, and you can find the sample LDIFs for populating the directory in the files that you downloaded.
After you have downloaded the sample data files as described in the previous section, run the following script to insert information from the example files into the LDAP database. You need the LDAP administrator password that you specified when you installed and configured OpenLDAP.

```
sudo ldapmodify -Y EXTERNAL -H ldapi:/// -f examples/eduPerson201310.ldif

# Schema installation requires root, but all other changes only require admin

ldapmodify $LDAP_ADMIN -f examples/PEOPLE.ldif

ldapmodify $LDAP_ADMIN -f examples/BIO.ldif

ldapmodify $LDAP_ADMIN -f examples/CSE.ldif

ldapmodify $LDAP_ADMIN -f examples/HR.ldif
```

Step 3: Install Tomcat and Shibboleth IdP

The next step is to install Shibboleth. Because Shibboleth is a construction of Java Server Pages, it needs a container in which to run. We are using Apache

Tomcat.[11] You do not have to know much about Tomcat in order to use it in this walkthrough; we will show you the installation and configuration steps.

Install Tomcat

The Tomcat installation is simple. You just need to download and unzip a tarball. In order to run Tomcat, a Java SE Development Kit (JDK) is required. Log in to the Amazon EC2 instance and run the following script in order to install the JDK, download Tomcat, and extract it.

```
sudo apt-get -y install openjdk-7-jre-headless

wget -O 'installers/tomcat7.tar.gz' '
http://www.us.apache.org/dist/tomcat/tomcat-
7/v7.0.59/bin/apache-tomcat-7.0.59.tar.gz'


# Tomcat installation is simply to extract the tarball
tar -xzf installers/tomcat7.tar.gz -C server/tomcat/ --
strip-components=1
```

Install Shibboleth IdP

You can install Shibboleth by downloading a tarball and extracting it. You then need to set an environment variable and run the Shibboleth installer script. In the Amazon EC2 instance, run the following script.

```
wget -O 'installers/shibidp2.4.tar.gz' '
http://shibboleth.net/downloads/identity-
provider/2.4.0/shibboleth-identityprovider-2.4.0-
bin.tar.gz'
tar -xzf installers/shibidp2.4.tar.gz -C installers/shibidp
--strip-components=1

# This is needed for Tomcat and Shibboleth scripts
echo "export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64/"
>> ~/.bashrc
source ~/.bashrc

# Installation directory: /home/ubuntu/server/shibidp
# (don't use ~)
# Domain: idp.example.com
cd installers/shibidp; ./install.sh && cd -
```

Use the following answers when prompted.

- `Where should the Shibboleth Identity Provider software be installed?`
  Type `/home/ubuntu/server/shibidp`
- (This question may not appear)
  `The directory '/home/ubuntu/server/shibidp' already exists. Would you like to overwrite this Shibboleth configuration? (yes, [no])`
  Type `yes`
- `What is the fully qualified hostname of the Shibboleth Identity Provider server? [idp.example.org]`
  Type `idp.example.com`. (Use .com, *not* `.org`, because that is what the LDAP installation uses.) Note that this response assumes that you typed example.com as the domain earlier.
- `A keystore is about to be generated for you. Please enter a password that will be used to protect it.`
  This password protects a key pair that is used to sign SAML assertions. It is stored in a file in the Shibboleth directory. For purposes of this walkthrough, use `password` everywhere you are prompted. In a production system, be sure to consult your security best practices.

Configure Tomcat

Tomcat's default configuration does not quite suit our needs for this example IdP, so you need to edit the server's configuration file.

1. In the Amazon EC2 instance, use an editor such as Vim to edit the following file.

```
/home/ubuntu/server/tomcat/conf/server.xml
```

2. Comment out the block that starts with `<Connector port="8080"`. This stops Tomcat from listening on port 8080.
3. Find the block that begins with `<Connector port="8443"`, and replace it with the following block. Notice that the block you are searching for contains the port 8443, and is being replaced with port 443.

```
<Connector port="443" protocol="HTTP/1.1" SSLEnabled="true"
     maxThreads="150" scheme="https" secure="true"
     clientAuth="false" sslProtocol="TLS"

keystoreFile="/home/ubuntu/server/shibidp/credentials/idp.j
ks"
```

```
      keystorePass="password" />
```

4. Create the following file.

```
/home/ubuntu/server/tomcat/conf/Catalina/localhost/idp.xml
```

5. Add the following to the file you just created, and then save and close the file.

```
<Context docBase="/home/ubuntu/server/shibidp/war/idp.war"
     privileged="true"
     antiResourceLocking="false"
     antiJARLocking="false"
     unpackWAR="false"
     swallowOutput="true" />
```

   This tells Tomcat where Shibboleth's files are and how to use them.

6. Run the following command.

```
cp ~/installers/shibidp/endorsed/* ~/server/tomcat/endorsed
```

   This command tells Tomcat that it can run the Shibboleth library files by copying the contents of Shibboleth's endorsed directory to Tomcat's endorsed directory.

7. Edit the Tomcat user store file that is in the following location.

```
/home/ubuntu/server/tomcat/conf/tomcat-users.xml
```

8. Add a root user by adding the following line *just before* the `</tomcat-users>` tag (inside the `tomcat-users` element).

```
<user username="root" password="password" roles="manager-
gui" />
```

   This configures Tomcat as an administrative user so that Tomcat can start and stop Shibboleth.

9. Start the server by running the following startup commands.

```
sudo /home/ubuntu/server/tomcat/bin/startup.sh
tail -f /home/ubuntu/server/tomcat/logs/catalina.out
```

10. Wait for a line that says, "`INFO: Server startup in ### ms`", and then press CTRL+C.
11. To verify that Tomcat and Shibboleth started properly, from your main computer (not the Amazon EC2 instance), navigate to https://idp.example.com. If the server is working, Tomcat displays a welcome page after a brief warning about certificates and host names.
12. Click **Manager App** and type the root credentials. Verify that the Shibboleth software is running.

Step 4: Configure IAM

Now that you have set up Shibboleth as an IdP, configure AWS IAM so that it can act as a SAML service provider. This involves two tasks: the first is to create an IAM SAML provider that describes the IdP, and the second is to create an IAM role (in our case several roles) that a federated user can assume in order to get temporary security credentials for accessing AWS resources, such as signing in to the AWS Management Console.

Create an IAM SAML Provider

In order to support SAML identity federation from an external IdP, IAM must first establish a trust relationship with the provider. To do this, create an IAM SAML provider. SAML 2.0 describes a document called a metadata document that contains all the required information to configure communication and trust between two entities. You can get the metadata document by asking Shibboleth running on your instance to generate it.

1. In your Amazon EC2 instance, navigate to the following URL, download the metadata document, and save it with the name idp.example.com.xml (use this name, because later steps assume this name).

```
https://idp.example.com/idp/profile/Metadata/SAML
```

2. Sign in to AWS and navigate to the IAM console.[12]
3. In the navigation pane, click **Identity Providers**, and then click **Create Provider**. The **Create Provider** wizard starts.
4. Choose **SAML** as the provider type.
5. Type `ShibDemo` as the name.
6. Upload the metadata document you saved in Step 1 of this procedure as the **Metadata Document**, as shown in Figure 6.

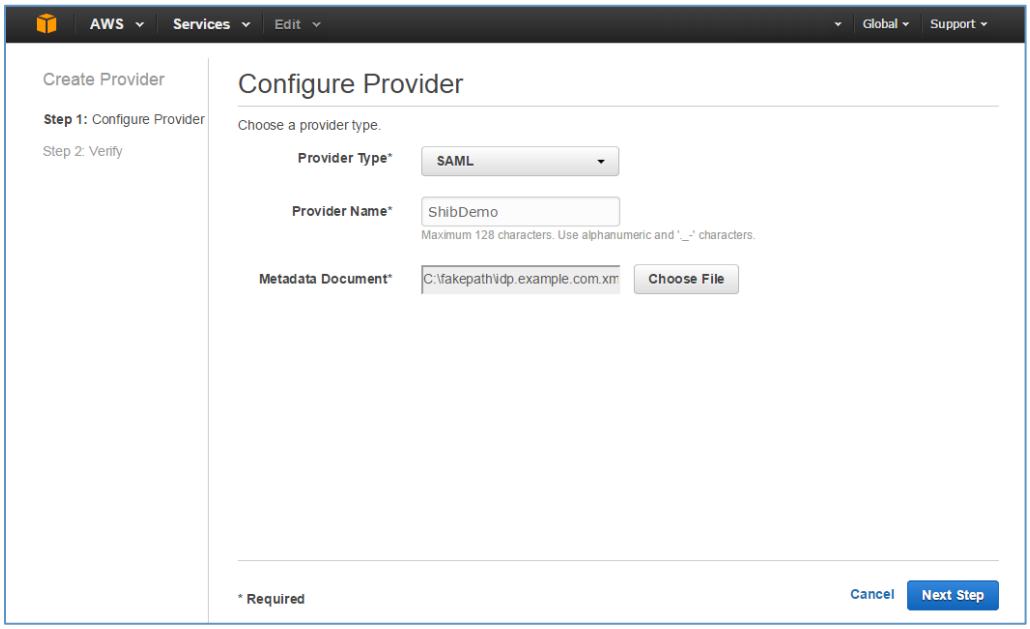

Figure 6: The Create Provider Wizard

7. Click **Next Step**.
8. Review the `Provider Name` and `Provider Type`, and then click **Create**.

Create IAM Roles

Next, you create IAM roles that federated users can assume. You create three roles for Example University: one for the biology department, one for the computer science and computer engineering departments to share, and one for the human resources department. Shibboleth controls access to the first two roles. The third role includes a condition so that Shibboleth and AWS manage access control (authorization).

In the IAM console, follow these steps:

1. In the navigation pane, click **Roles**, and then click **Create New Role**.
2. Type `BIO` for the name of the first role, and then click **Next Step**.
3. For role type, select **Role for Identity Provider Access**.
4. Select **Grant Web Single Sign-On (WebSSO) access to SAML providers**, as shown in Figure 7.

Figure 7: Grant WebSSO Access to SAML Providers

By default, the wizard selects the SAML provider that you created earlier (see Figure 8). The wizard also shows that the **Value** field is set to `https://signin.aws.amazon.com/saml`. This is a required value.

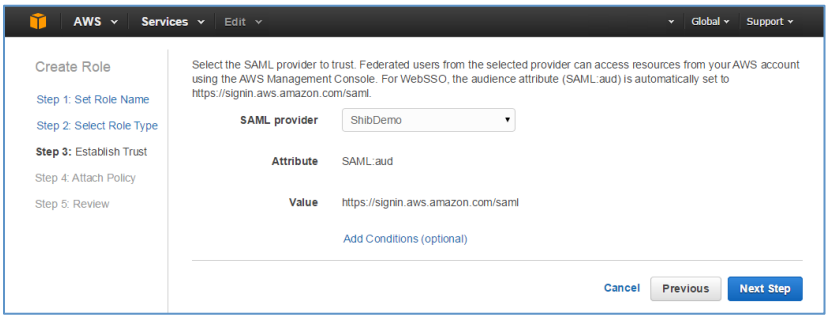

Figure 8: Create Role Wizard

5. Click **Next Step** and verify that the role trust policy matches the following example (except that your policy includes your AWS account number, instead of `000000000000`). When you have verified the policy, click **Next Step**.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sts:AssumeRoleWithSAML",
      "Principal": {
        "Federated": "arn:aws:iam::000000000000:saml-
provider/ShibDemo"
      },
      "Condition": {
        "StringEquals": {
          "SAML:aud": "https://signin.aws.amazon.com/saml"
        }
      }
    }
  ]
}
```

6. In the **Attach Policy** step, do not select any options. For this exercise, the role does not actually need to have any permissions. Instead, click **Next Step**. You see a summary of the role, as shown in Figure 9.

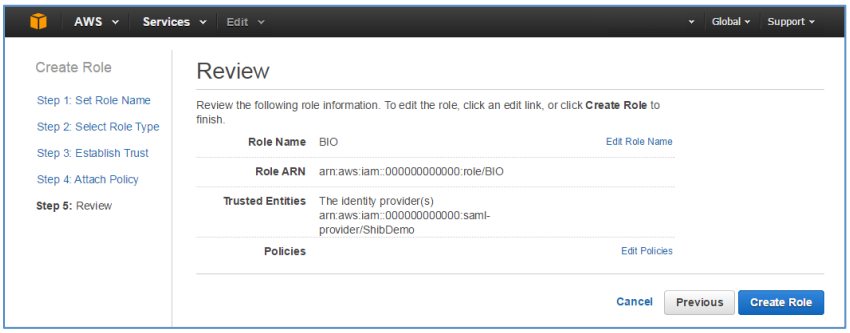

Figure 9: Summary of the Created Role

Note the role's Amazon Resource Name or ARN (`arn:aws:iam::0000000000000:role/BIO`). Later parts of this walkthrough assume that the ARNs of the roles you create in this procedure match the suggested names (`BIO`, `CSE`, and `HR`).

7. Click **Create Role** to finish creating this role.
8. Repeat steps 1–7 to create another role named `CSE`.
9. Repeat the steps again to create another role named `HR`. For the `HR` role, you need to add a condition to check that at least one of the values of the `SAML:eduPersonPrimaryOrgUnitDN` attribute is a string that is required.

   When you get to the **Verify Role Trust** step, copy and paste the following policy. Remember to replace `000000000000` with your AWS account number.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sts:AssumeRoleWithSAML",
      "Principal": {
        "Federated": "arn:aws:iam::000000000000:saml-
provider/ShibDemo"
      },
      "Condition": {
        "StringEquals": {
          "SAML:aud": "https://signin.aws.amazon.com/saml"
        },
        "ForAnyValue:StringEquals": {
          "SAML:eduPersonPrimaryOrgUnitDN":
"ou=hr,dc=example,dc=com"
        }
      }
```

```
        }
      ]
    }
```

The extra condition restricts the `HR` role to the manager of HR, because Example University uses the `eduPersonPrimaryOrgUnitDN` attribute to denote managers.

10. As with the `BIO` and `CSE` roles, do not select any policies to attach as the role's access policy, because no permissions are needed for this walkthrough.

Step 5: Configure Shibboleth IdP

Shibboleth IdP consumes data from a variety of sources and uses that data to both authenticate a user and communicate the authenticated identity to external entities. You can configure nearly every part of the process, and you can extend with code the portions of the IdP that do not support configuration settings.

About Shibboleth Data Connectors

The basic flow for attribute data through Shibboleth is the same, regardless of whether the data comes from a database, LDAP, or another source. A component called a *data connector* fetches attribute data from its source. The data connector defines a query or filter used to get the identity data. Predefined data connectors exist for relational databases, LDAP, and configuration files.

The results returned by the data connector persist into the next step in the process, which is the attribute definition. In this step, you can process the identity data pulled from the store (and potentially from other attributes defined earlier in the configuration) to produce attributes with the format you need. For example, an attribute can pull several columns of a relational database together with appropriate delimiters and format an email address. Like data connectors, Shibboleth supports predefined attribute definitions. One definition passes identity values through with no modification. With the mapped attribute definition, you can use regular expressions to transform the format of attributes. A number of special attribute definitions expose some of Shibboleth's internal mechanisms, which are interesting but will not be used here.

However, these attributes are still in a Shibboleth-specific internal format. You can attach attribute encoders to the attribute definitions so that you can serialize the internal attributes into whatever wire format you need. This walkthrough uses the SAML 2.0 string encoder to create the required XML for the SAML authentication responses.

After you have fetched, transformed, and encoded data into the correct format, you can use attribute filters to dictate which attributes to include in communication with various relying parties. Predefined attribute filter policies give you great flexibility in releasing attributes to relying parties. You can use filters to write attributes to specific relying parties only, and to write only specific

values of the attributes, specific users, or specific authentication methods. You can also string together Boolean combinations of all the above. A complete overview of the process appears in Figure 10.

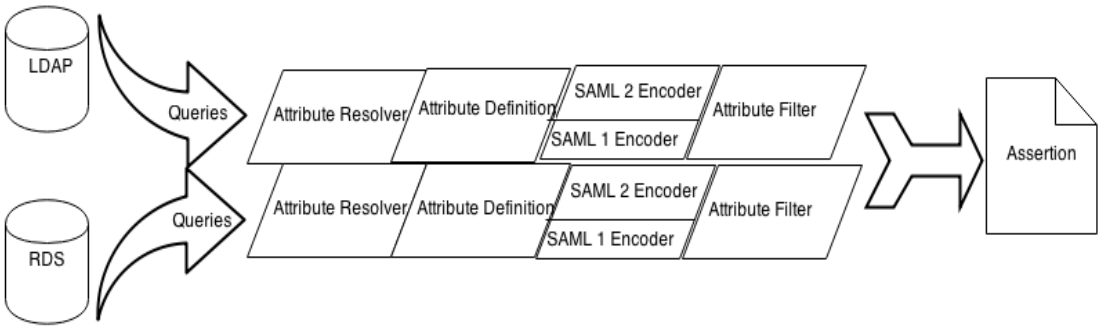

Figure 10: Attribute Pipeline in Shibboleth

Fetch Attributes from OpenLDAP

Much of the configuration for getting Shibboleth to communicate with OpenLDAP is already in existing files and just needs to be uncommented.

1. In your Amazon EC2 instance, open this file in your text editor.

```
/home/ubuntu/server/shibidp/conf/attribute-resolver.xml
```

2. In the file, find the section with the following heading.

```
#  <!-- Schema: eduPerson attributes -->
```

3. Uncomment that section. (The commented-out section ends *before* an element that has the ID `eduPersonTargetedID`.)

4. If you are using a newer schema that includes the definitions for `eduPersonPrincipalNamePrior` or `eduPersonUniqueId` (the `eduPerson` object class specification 201310), you can optionally add the following block *after* the block that you just uncommented.

```
<resolver:AttributeDefinition
  xsi:type="ad:Simple" id="eduPersonPrincipalNamePrior"
  sourceAttributeID="eduPersonPrincipalNamePrior">
    <resolver:Dependency ref="myLDAP" />
    <resolver:AttributeEncoder
      xsi:type="enc:SAML1String"
      name="urn:mace:dir:attribute-
def:eduPersonPrincipalNamePrior" />
    <resolver:AttributeEncoder
```

```
      xsi:type="enc:SAML2String"
      name="urn:oid:1.3.6.1.4.1.5923.1.1.1.12"
      friendlyName="eduPersonPrincipalNamePrior" />
</resolver:AttributeDefinition>

<resolver:AttributeDefinition
  xsi:type="ad:Simple" id="eduPersonUniqueId"
  sourceAttributeID="eduPersonUniqueId">
    <resolver:Dependency ref="myLDAP" />
    <resolver:AttributeEncoder xsi:type="enc:SAML1String"
      name="urn:mace:dir:attribute-def:eduPersonUniqueId"
/>
    <resolver:AttributeEncoder
      xsi:type="enc:SAML2String"
      name="urn:oid:1.3.6.1.4.1.5923.1.1.1.13"
      friendlyName="eduPersonUniqueId" />
</resolver:AttributeDefinition>
```

5. Find the section that begins with the following.

```
<!-- Example LDAP Connector -->
```

   This section has been commented out.
6. Replace that entire commented-out section with the following block, and then save and close the file.

```
<resolver:DataConnector
  id="myLDAP"
  xsi:type="dc:LDAPDirectory"
  ldapURL="ldap:///"
  baseDN="ou=people,dc=example,dc=com"
  authenticationType="ANONYMOUS" >
    <dc:FilterTemplate>
      <![CDATA[
        (uid=$requestContext.principalName)
      ]]>
    </dc:FilterTemplate>
</resolver:DataConnector>
```

About Attribute Definitions

The most relevant part of an LDAP data connector block is the filter template near the bottom of the definition. When Shibboleth requests attributes for a user, it runs this query on the OpenLDAP database. OpenLDAP needs to authenticate and needs to know where to search. This is what the `authenticationType` and `baseDN` attributes define. The reference `myLDAP` is used to refer to this specific OpenLDAP query. If there are other attributes in OpenLDAP that require a different query, you can copy this block, give it a different ID, and change the query.

The block contains the following `eduPerson` attribute definition.

```
<resolver:AttributeDefinition
  xsi:type="ad:Simple"
  id="eduPersonAffiliation"
  sourceAttributeID="eduPersonAffiliation">
    <resolver:Dependency ref="myLDAP" />
    <resolver:AttributeEncoder
      xsi:type="enc:SAML1String"
      name="urn:mace:dir:attribute-
def:eduPersonAffiliation" />
    <resolver:AttributeEncoder
      xsi:type="enc:SAML2String"
      name="urn:oid:1.3.6.1.4.1.5923.1.1.1.1"
      friendlyName="eduPersonAffiliation" />
</resolver:AttributeDefinition>
```

The `xsi:type="ad:Simple"` attribute in these definitions indicates that these attributes simply copy their values from the data connector as is. This is appropriate for attributes that map directly to single columns of a database, to single attributes from OpenLDAP, or to static configuration data.

The `id="eduPersonAffiliation"` portion gives this configuration section an internal name that can be referenced elsewhere in the configuration. It is never released to relying parties. The `sourceAttributeID="eduPersonAffiliation"` portion defines the name of the attribute released by the data connector to use as the source of data for this attribute definition. Because this attribute definition gets data from OpenLDAP, the configuration specifies a dependency on myLDAP, which is the ID that you assigned to the OpenLDAP data connector.

Finally, a number of encoders are attached. In the SAML 2.0 string encoder, the `name` and `friendlyName` are used to set the same portions of a SAML2 attribute.

Configuring AWS-specific attribute definitions

To use SAML identity federation with AWS, you must configure two AWS-specific attributes. The first is a simple attribute that sets the name of the session granted to users. This value is captured in logs and displayed in the console when the user signs in. Good candidates for this value are a user's login name or email address. Some format restrictions exist for the value:

- It must be between 2 and 32 characters in length.
- It can contain only alphanumeric characters, underscores, and the following characters: +=,.@-.
- It is typically a user ID (bobsmith) or an email address (bobsmith@example.com).
- It should not include blank spaces, such as often appear in a user's display name (Bob Smith).

This example uses the `uid` of the user from OpenLDAP by setting the `sourceAttributeID` to `uid` and adding a dependency on the OpenLDAP data connector.

The other attribute that needs to be set is the list of roles the user can assume. This could be as simple as a static value attached to all users in an organization or as complex as a per-user, per-department, ACL (access control list)–based value. This example uses a flexible option that is not difficult to implement. To configure the attributes, follow these steps.

1. Edit the following file.

```
/home/ubuntu/server/shibidp/conf/attribute-resolver.xml
```

2. Insert the following block immediately *after* the heading `"Attribute Definitions"`, and *before* `<!--Schema: Core schema attributes-->`.

   Note: Replace `000000000000` with your AWS account number. Note also that the block includes the ARNs of the roles that you created earlier (for example, `arn:aws:iam::000000000000:role/BIO`).

```
<resolver:AttributeDefinition
  id="awsRoles"
  xsi:type="ad:Mapped"
  sourceAttributeID="eduPersonOrgUnitDN">

  <resolver:Dependency ref="myLDAP"/>
  <resolver:AttributeEncoder
    xsi:type="enc:SAML2String"
    name="https://aws.amazon.com/SAML/Attributes/Role"
    friendlyName="Role" />
```

```
  <ad:ValueMap>
    <ad:ReturnValue>
arn:aws:iam::000000000000:role/BIO,arn:aws:iam::00000000000
0:saml-provider/ShibDemo
    </ad:ReturnValue>
    <ad:SourceValue>.*ou=biology.*</ad:SourceValue>
  </ad:ValueMap>

  <ad:ValueMap>
    <ad:ReturnValue>
arn:aws:iam::000000000000:role/CSE,arn:aws:iam::00000000000
0:saml-provider/ShibDemo
    </ad:ReturnValue>
    <ad:SourceValue>.*ou=computersci.*</ad:SourceValue>
    <ad:SourceValue>.*ou=computereng.*</ad:SourceValue>
  </ad:ValueMap>

  <ad:ValueMap>
    <ad:ReturnValue>
arn:aws:iam::000000000000:role/HR,arn:aws:iam::000000000000
:saml-provider/ShibDemo
    </ad:ReturnValue>
    <ad:SourceValue>.*ou=hr.*</ad:SourceValue>
  </ad:ValueMap>

</resolver:AttributeDefinition>

<resolver:AttributeDefinition
  id="awsRoleSessionName"
  xsi:type="ad:Simple"
  sourceAttributeID="uid">
    <resolver:Dependency ref="myLDAP"/>
    <resolver:AttributeEncoder
      xsi:type="enc:SAML2String"

name="https://aws.amazon.com/SAML/Attributes/RoleSessionNam
e"
      friendlyName="RoleSessionName" />
</resolver:AttributeDefinition>
```

With the mapped attribute definition, you can use a regular expression to map input values into output values. This example maps `eduPersonOrgUnitDN` to

an IAM role (depending on the organizational unit) in order to give entire departments access to resources by using existing access control rules. The attribute definition contains several value maps, each with its own pattern. Each of the values associated with the `eduPersonOrgUnitDN` (because it is multivalued) is checked against the patterns specified in the `SourceValue` nodes. If the check finds a match, the `ReturnValue` value is added to the attribute definition. The format of the `ReturnValue` is a role ARN and a provider ARN, separated by a comma. The order of the two ARNs does not matter. If you are using regular expressions in the `SourceValue` fields, you can use back references in the `ReturnValue` so that you can simplify the configuration by capturing the organizational unit and using a back reference, although delving into further possibilities of using pattern matching is beyond our scope.

Release Attributes to Relying Parties

Sometimes attributes can contain sensitive data that is useful for authentication within the organization. No one should release the sensitive data outside of the organization. The first part of an attribute filter defines to whom the filter applies. By using an `AttributeRequesterString` filter policy, an administrator can choose the relying parties to whom to release the attributes. This example uses the entity ID of AWS `"urn:amazon:webservices"`. This walkthrough uses a simple directory, so all possible values of all the `eduPerson` and AWS attributes are released to AWS. This allows you to write policies in IAM that can include conditions based on attributes that represent OpenLDAP information. You do this by including an `AttributeRule` element for each `eduPerson` entity or AWS attribute, and setting `PermitValueRule` to `basic:ANY`.

1. Edit the following file.

```
/home/ubuntu/server/shibidp/conf/attribute-filter.xml
```

2. Add the following block *inside* the element `AttributeFilterPolicyGroup` (*before* the closing `</afp:AttributeFilterPolicyGroup>` tag, and *after* the comments). When you are done, save and close the file.

```
<afp:AttributeFilterPolicy id="releaseEduAndAWSToAWS">
    <afp:PolicyRequirementRule
      xsi:type="basic:AttributeRequesterString"
      value="urn:amazon:webservices" />

    <afp:AttributeRule attributeID="eduPersonAffiliation">
        <afp:PermitValueRule xsi:type="basic:ANY"/>
```

```
    </afp:AttributeRule>
    <afp:AttributeRule attributeID="eduPersonEntitlement">
        <afp:PermitValueRule xsi:type="basic:ANY"/>
    </afp:AttributeRule>
    <afp:AttributeRule attributeID="eduPersonNickname">
        <afp:PermitValueRule xsi:type="basic:ANY"/>
    </afp:AttributeRule>
    <afp:AttributeRule attributeID="eduPersonOrgDN">
        <afp:PermitValueRule xsi:type="basic:ANY"/>
    </afp:AttributeRule>
    <afp:AttributeRule attributeID="eduPersonOrgUnitDN">
        <afp:PermitValueRule xsi:type="basic:ANY"/>
    </afp:AttributeRule>
    <afp:AttributeRule
attributeID="eduPersonPrimaryAffiliation">
        <afp:PermitValueRule xsi:type="basic:ANY"/>
    </afp:AttributeRule>
    <afp:AttributeRule
attributeID="eduPersonPrimaryOrgUnitDN">
        <afp:PermitValueRule xsi:type="basic:ANY"/>
    </afp:AttributeRule>
    <afp:AttributeRule
attributeID="eduPersonPrincipalName">
        <afp:PermitValueRule xsi:type="basic:ANY"/>
    </afp:AttributeRule>
    <afp:AttributeRule
attributeID="eduPersonScopedAffiliation">
        <afp:PermitValueRule xsi:type="basic:ANY"/>
    </afp:AttributeRule>
    <afp:AttributeRule attributeID="eduPersonAssurance">
         <afp:PermitValueRule xsi:type="basic:ANY"/>
    </afp:AttributeRule>
    <afp:AttributeRule attributeID="eduPersonTargetedID">
         <afp:PermitValueRule xsi:type="basic:ANY"/>
    </afp:AttributeRule>
    <afp:AttributeRule attributeID="awsRoles">
         <afp:PermitValueRule xsi:type="basic:ANY"/>
    </afp:AttributeRule>
    <afp:AttributeRule attributeID="awsRoleSessionName">
         <afp:PermitValueRule xsi:type="basic:ANY"/>
    </afp:AttributeRule>
</afp:AttributeFilterPolicy>
```

Enable Login Using OpenLDAP as a User Store

Shibboleth supports several authentication methods. By default, remote user authentication is configured, which passes through authentication from Tomcat. To authenticate against OpenLDAP, you must disable remote user authentication and enable user name/password authentication. User name/password authentication via JAAS and the `login.config` file are already defined in the configuration file; you just need to uncomment it. Follow these steps:

1. In the Amazon EC2 instance, edit the following file.

```
/home/ubuntu/server/shibidp/conf/handler.xml
```

2. Comment out the following block.

```
<ph:LoginHandler xsi:type="ph:RemoteUser">
```

3. Uncomment the following block, and then save and close the file.

```
<ph:LoginHandler xsi:type="ph:UsernamePassword ...>
```

4. Edit the following file in order to configure the OpenLDAP connection parameters.

```
/home/ubuntu/server/shibidp/conf/login.config
```

5. Find the block that begins with Example LDAP authentication. Replace the entire commented section (which begins with `edu.vt.middleware`) with the following block.

```
edu.vt.middleware.ldap.jaas.LdapLoginModule required
      ldapUrl="ldap://localhost"
      baseDn="ou=People,dc=example,dc=com"
      bindDn="cn=admin,dc=example,dc=com"
      bindCredential="password"
      userFilter="uid={0}";
```

Configure Shibboleth to Talk to AWS

Now you have an OpenLDAP directory and Shibboleth configured to use that identity store, and you have created IAM entities that AWS needs to establish

trust with Shibboleth. The only thing left is to establish trust between Shibboleth (as the IdP) and AWS (as a service provider). You do this by configuring Shibboleth with the location of the AWS SAML 2.0 metadata document. A metadata document contains all the information needed for two parties to communicate such as Internet endpoints and public keys. Shibboleth can automatically refresh AWS metadata when AWS changes it by using a `FileBackedHTTPMetadataProvider` object. Alternatively, if an administrator wants to control the relationship manually, the administrator can manually download the metadata and use a `FileSystemMetadataProvider`.

1. In your Amazon EC2 instance, edit the following file.

```
/home/ubuntu/server/shibidp/conf/relying-party.xml
```

2. In the Metadata Configuration section, just below the `IdPMD` entry, add the following.

```
<metadata:MetadataProvider
  id="AWS"
  xsi:type="metadata:FileBackedHTTPMetadataProvider"
  metadataURL="https://signin.aws.amazon.com/static/saml-
metadata.xml"

backingFile="/home/ubuntu/server/shibidp/metadata/aws.xml"
/>
```

   The file contains settings that cause Shibboleth to apply a set of default configurations to AWS. You can find these settings inside the `DefaultRelyingParty` and `AnonymousRelyingParty` blocks.

3. To change the configuration for a specific relying party, insert the following block *after* the `DefaultRelyingParty` block (after the closing `</DefaultRelyingParty>` tag).

```
<rp:RelyingParty
  id="urn:amazon:webservices"
  provider="https://idp.example.com/idp/shibboleth"
  defaultSigningCredentialRef="IdPCredential">

    <rp:ProfileConfiguration
      xsi:type="saml:SAML2SSOProfile"
      includeAttributeStatement="true"
      assertionLifetime="PT5M" assertionProxyCount="0"
```

```
        signResponses="never" signAssertions="always"
        encryptAssertions="never" encryptNameIds="never"
        includeConditionsNotBefore="true"
        maximumSPSessionLifetime="PT1H" />
</rp:RelyingParty>
```

With This configuration, you can specify the following:

- `defaultSigningCredentialRef` – The keys used to sign and encrypt requests.
- `ProfileConfiguration` – Which SAML 1.x or SAML 2.0 profiles to respond to. Keep in mind that AWS supports *only* SAML2SSOProfile.
- `assertionLifetime` – The length of time (expiration) for the user to provide the authentication information to AWS before it is no longer valid.
- `signResponses/signAssertions` – The portions of the response to sign.
- `maximumSPSessionLifetime` – The length of a session that AWS should provide based on the authentication information provided.

Test Configuration Changes by Using AACLI

You have configured Shibboleth! To apply the Shibboleth configuration changes, you must restart Tomcat. However, before you do that, it is best to test the configuration. You can use the attribute authority command line interface (AACLI) tool to simulate Shibboleth's attribute construction based on an arbitrary configuration directory. This allows you to copy a working configuration to a test directory, modify it, test it, and then copy it back. For the sake of this example, you set up AACLI to test the live configuration.

1. Edit the following file.

```
~/.bashrc
```

2. Add the following block to the file, and then save and close the file.

```
echo "alias aacli='sudo -E
/home/ubuntu/server/shibidp/bin/aacli.sh --
configDir=/home/ubuntu/server/shibidp/conf'" >> ~/.bashrc
```

3. Run the following source command.

```
source ~/.bashrc
```

4. Run the following AACLI command.

```
aacli --requester "urn:amazon:webservices" --principal
bobby
```

   The attributes that are constructed for a given principal can be tested by filling in a principal's OpenLDAP `uid`. (In this case, you use the principal `bobby`, which exists in the example LDAP database.)

   If all goes well, the command displays XML information that could be directly injected into a SAML 2.0 attribute statement block. If you see a series of stack traces instead, a misconfiguration is present. Check the settings for the OpenLDAP data connector and the syntax of all the XML configuration files.

5. After the AACLI begins returning attributes, stop and then restart Tomcat by using the following commands.

```
sudo /home/ubuntu/server/tomcat/bin/shutdown.sh
sudo /home/ubuntu/server/tomcat/bin/startup.sh
```

Ensure that no stack traces occur in Tomcat or in the Shibboleth logs.

Step 6: Test Shibboleth Federation

As soon as the previous testing is working, you can test federation to AWS. In the Amazon EC2 instance, open a browser and navigate to the following URL.

```
https://idp.example.com/idp/profile/SAML2/Unsolicited/SSO?p
roviderId=urn:amazon:webservices
```

This initiates the SSO flow to AWS, and you see the page shown in Figure 11.

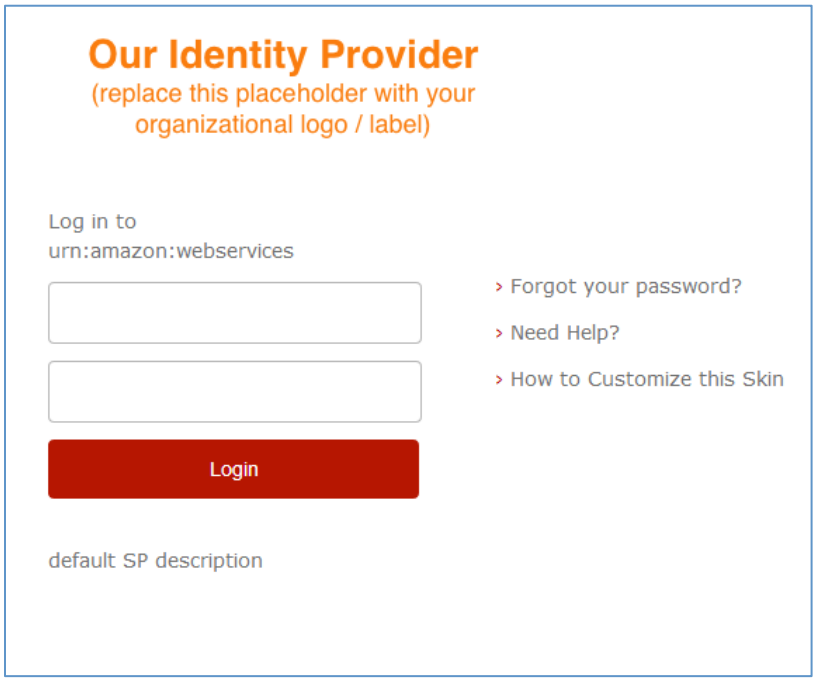

Figure 11: The Custom Login Page for the AWS Management Console

Type the user name `bobby`, and use `password` for the password. (In the sample LDAP data, all the passwords are `password`.) You then go to the AWS Management Console, as shown in Figure 12.

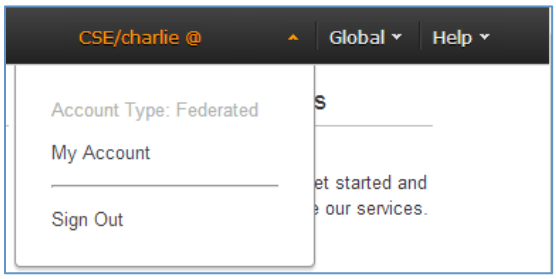

Figure 12: Console for a User Logged In as Charlie Using a Role Named CSE

To try a different user, log out by navigating to https://idp.example.com/idp/profile/Logout. Then try logging in as user `Dean`. Notice that this user is unable to federate. This is because the HR role policy specifies that the `SAML:eduPersonPrimaryOrgUnitDN` must be `ou=hr,dc=example,dc=com`.

The user `bobby` has this and can federate as a member of the HR department. However, Dean's primary organizational unit is `ou=People,dc=example,dc=com`.

As noted earlier, administrators have the flexibility to control access in two places. The first place is on the Shibboleth side in the attribute resolver by attaching specific AWS role attributes to specific users. The role that is associated with a user then determines what the user can do in AWS. The second place is in the IAM role trust policy, where you can add conditions based on SAML

attributes that limit who can assume the role. It is up to you to choose which of these two strategies to use (or both).
For a complete list of attributes that you can use in role trust policies, see the IAM documentation.[13]

Conclusion

Now that you have integrated your on-premises LDAP infrastructure into IAM, you can spend less time on synchronizing permissions between on-premises and the cloud. The combination of SAML attributes and RBAC means you can author fine-grained access control policies that address your LDAP user data and your AWS resources.

Further Reading

For more information about installing and configuring OpenLDAP and Shibboleth, see the following:

- Installing an OpenLDAP server[14]
- How To Install and Configure a Basic LDAP Server on an Ubuntu 12.04 VPS[15]
- LDIF examples[16]
- Edit the Tomcat Configuration File[17]
- Preparing Apache Tomcat for the Shibboleth Identity Provider[18]

For Shibboleth attributes and authentication responses, the Shibboleth documentation wiki provides extensive information. These topics contributed to the creation of this tutorial:

- LDAP Data Connector[19]
- Shibboleth attributes:
  - Define and Release a New Attribute in an IdP[20]
  - Simple Attribute Definition[21]
  - Mapped Attribute Definition[22]
  - Define a New Attribute Filter[23]
- Shibboleth User Name/Password Handler[24]
- Adding Metadata providers[25]
- Per-Service Provider Configuration[26]

Notes

[1] http://en.wikipedia.org/wiki/Ldap

[2] http://aws.amazon.com/about-aws/whats-new/2013/11/11/aws-identity-and-access-management-iam-adds-support-for-saml-security-assertion-markup-language-2-0/

[3] See the “Install Tomcat” section.

[4] See the “Install Shibboleth IdP” section.

[5] See the “Configure Shibboleth IdP” section.

[6] http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AccessingInstances.html

[7] http://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/billing-free-tier.html

[8] http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EC2_GetStarted.html

[9] http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AccessingInstances.html

[10] http://en.wikipedia.org/wiki/Ldap

[11] http://tomcat.apache.org/

[12] https://console.aws.amazon.com/iam/home?#home

[13] http://docs.aws.amazon.com/IAM/latest/UserGuide/AccessPolicyLanguage_ElementDescriptions.html#condition-keys-saml

[14] https://help.ubuntu.com/lts/serverguide/openldap-server.html#openldap-server-installation

[15] https://www.digitalocean.com/community/articles/how-to-install-and-configure-a-basic-ldap-server-on-an-ubuntu-12-04-vps

[16] http://www.zytrax.com/books/ldap/ch5/step4.html#step4-ldif

[17] http://tomcat.apache.org/tomcat-7.0-doc/ssl-howto.html#Edit_the_Tomcat_Configuration_File

[18] https://wiki.shibboleth.net/confluence/display/SHIB2/IdPApacheTomcatPrepare

[19] https://wiki.shibboleth.net/confluence/display/SHIB2/ResolverLDAPDataConnector

[20] https://wiki.shibboleth.net/confluence/display/SHIB2/IdPAddAttribute

[21] https://wiki.shibboleth.net/confluence/display/SHIB2/ResolverSimpleAttributeDefinition

[22] https://wiki.shibboleth.net/confluence/display/SHIB2/ResolverMappedAttributeDefinition

[23] https://wiki.shibboleth.net/confluence/display/SHIB2/IdPAddAttributeFilter

[24] https://wiki.shibboleth.net/confluence/display/SHIB2/IdPAuthUserPass

[25] https://wiki.shibboleth.net/confluence/display/SHIB2/IdPMetadataProvider

[26] https://wiki.shibboleth.net/confluence/display/SHIB2/IdPRelyingParty