Access Control Services

Contents

Copyright GE Digital

Access Control Services Overview

About Access Control Services

Access Control Services (ACS) are security services provided on predix platform for application developers to add granular authorization mechanisms to access web applications and services without having to add complex authorization logic to their code. ACS works in conjunction with the User Account and Authentication (UAA) security service.

Access Control Services provide the following benefits:

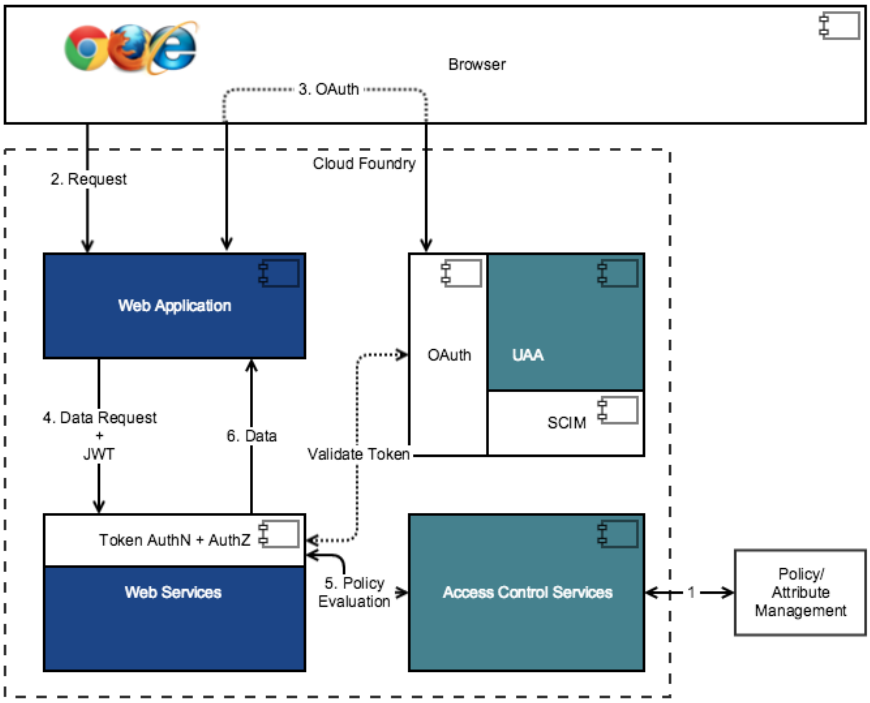
- Ability to maintain access-decision data as policies and attributes.
- Exclusive security for multiple clients using the service, since the ACS services are tenant-aware.
- Support for fine-grained authorization policies.

Additional Information

Exploring Access Control Service - ACS Guides

ACS Deployment Architecture

The following diagram shows how ACS services are deployed in cloud foundry.



In this flow:

1. An administrator sets up policies and user privileges using ACS.
2. An application user requests data using a browser.

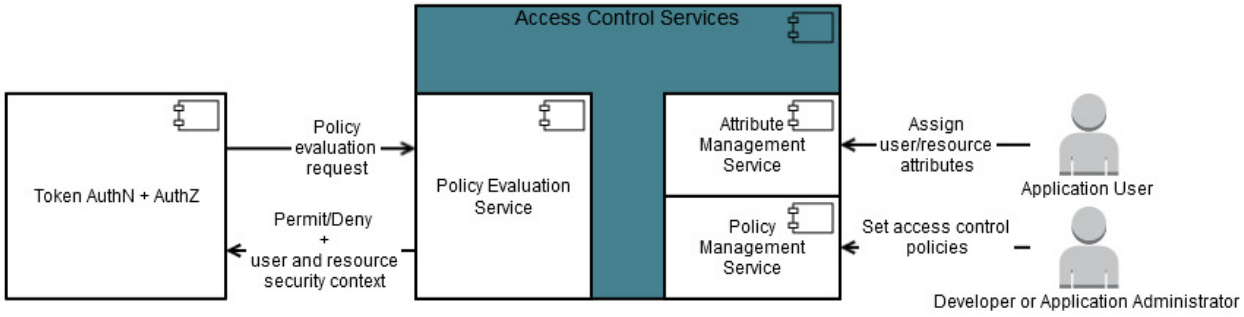
3. The web application sends the authentication request to UAA. If the user is set up in UAA, the authentication request is approved, the user is authenticated, and the token is issued to the web application. If the data request already contains a valid token, this step is not required.
4. The web application passes the data request and JWT token to the web services.
5. The web services use ACS REST APIs to authorize a user based on a policy evaluation.
6. Once the user is authorized, the data is returned to the web application.

Access Control Services Architecture

Access Control Services consist of the following primary services:

- The Policy Management service provides CRUD operations for application policies.
- The Attribute Management service provides CRUD operations for user and resource attributes.
- The Policy Evaluation service processes policy evaluation (such as access control) requests for an OAuth client.

The following diagram shows the Access Control Services architecture.



Attribute Management Service

The Attribute Management service allows you to create attributes for users and resources.

Attributes are characteristics of a user or resource that can be used to make access-control decisions. An attribute is identified by an issuer, the entity that asserts the attribute, and a name that describes the attribute. Some example of user and resource attributes include the organization, site, and group to which a resource belongs. Attributes are used in conjunction with access-control policies for user authorization.

Policy Management Service

The Policy Management service allows you (with required privileges) to create, read, update, and delete access-control policies.

An access control policy contains a set of rules that determine the required permissions for the specified users and resources. The rules can take into consideration the user attributes, the action the user wants to perform, the resource URI, and any resource attributes that further describe the resource.

Policy Evaluation Service

The Policy Evaluation service evaluates access-control policies based on web service requests for authorization.

A web service request consists of the following components:

- Resource information: Relative URI path of the request.
- Subject information: The Policy Evaluation service extracts this information from the JSON Web Token (JWT) used to authenticate with the web service.
- Action information: HTTP method used for the request.

The Policy Evaluation service leverages the centralized identity management feature offered by the User Account and Authentication (UAA) service. Once a user is set up in UAA, the JSON Web Token (JWT) issued to the user can be passed with a resource request to the Policy Evaluation service for authentication and authorization, The conditions defined in access-control policies are used to evaluate a resource request, and the requested resource is returned if a user is authorized.

ACS Attribute Connector Architecture

Extend Access Control Services to provide authorization from external attribute sources.

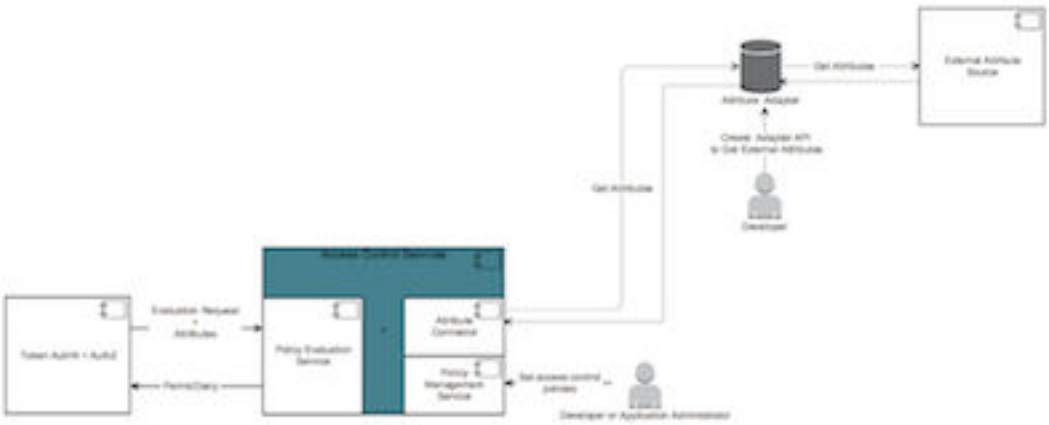
ACS support for external attribute sources provides the following benefits:

- Allow ACS policies to reference attributes of a protected resource from an external source (such as Predix Asset service).
- Allow ACS policies to reference attributes of a currently authenticated user from an external source (such as federated identity management):
    - Removes the need to store user attributes in the Attribute Management Service.
    - Attributes that are available in a Security Assertion Markup Language (SAML) assertion or OpenID Connector (OIDC) provider can be part of the ACS evaluation request that is pushed with the UAA validation token.

Access Control Services implements support for external attribute sources with the following functions:

- A custom ACS attribute adapter that pulls resource or subject attributes from a specific external attribute source. If an adapter is configured and is active, it will be used as the source for attributes instead of the built-in attribute store managed by the Attribute Management Service (see Access Control Services Architecture on page 2).
- A common ACS attribute connector that configures the endpoint and credentials for the ACS attribute adapter used when performing policy evaluations.

The following diagram shows the ACS attribute connector architecture.



There is a separate service call required to access external attributes for policy evaluation (see Example: ACS Attribute Connector Setup on page 58 for more information). Connectors should be used only when the additional cost is justified. For example:

- The system of record for attributes is an external resource that is not related to the business data store.
- It is not feasible for the resource service to get external attributes from the attribute data store.

Get Started with Access Control Services

Access Control Services Setup

Authentication for the Access Control services(ACS) is controlled by the designated trusted issuer and is managed by the User Account and Authentication (UAA) security service. You must set up a UAA service instance as the trusted issuer before getting started with the ACS services.

For information about authentication and authorization in Predix services, see About the User Account and Authentication Security Service.

Task Roadmap

#	Task	Information
1	(Optional) Configure your proxy settings if necessary.	Depending on your location and network configuration, you may need to configure your proxy settings to access remote resources. See Defining Proxy Connections to Remote Resources.
2	(Optional) Set up access to Predix platform Artifactory.	If you need access to Predix platform artifacts, you need to set up access for Artifactory. Predix provides ACS spring security extensions for integrating with spring security. The libraries are stored in Artifactory. See Defining Predix Platform Artifactory Access.
3	Deploy your application to Cloud Foundry.	For an example of deploying a Predix Hello World Web application to cloud foundry, see Creating and Deploying a Simple Web App to Cloud Foundry.
4	Create an instance of the trusted issuer.	Create an instance of User Account and Authentication (UAA) service. UAA is the authorization server that each platform service uses for authentication. For more information, see Creating a UAA Service Instance on page 6.
5	Create an instance of the ACS service.	For more information, see Creating an ACS Instance on page 9.
6	(Optional) Update an ACS instance that you created.	For more information, see Updating an ACS Instance on page 10.
7	Create OAuth2 clients to setup access to your service authenticated using UAA.	When you create a UAA instance, an admin client is automatically created for you to access UAA for additional configuration. You can create a new client for your service instance with specific scopes. If an Oauth2 client already exists, you can update the client to add your service instance. For more information, see Creating an OAuth2 Client on page 11.

#	Task	Information
8	Update the Oath2 client to add service specific scopes or authorities.	To enable your application to access a platform service, your JSON Web Token (JWT) must contain the scopes required for a platform service. For more information see Updating the OAuth2 Client for Services on page 14. For ACS specific scopes, see Authorities or Scopes Required for ACS Services on page 16.
9	Bind your application to the service instance.	To establish communication between your application and the platform service, you must bind the application to the service. See Binding an Application to the ACS Instance on page 16.
10	Start using the Access Control services.	See Using Access Control Services.

Creating a UAA Service Instance

You can create multiple instances of the UAA service in your space.

About This Task

As a best practice, first delete any older unused instances before creating a new one.

Procedure

1. Sign into your Predix account at https://www.predix.io.
2. Navigate to **Catalog** > **Services**, then click the **User Account and Authentication** tile.
3. Click **Subscribe** on the required plan.
4. Complete the fields on the **New Service Instance** page.

Field	Description
Org	Select your organization.
Space	Select the space for your application.
Service instance name	Enter a unique name for this UAA service instance.
Service plan	Select a plan.
Admin client secret	Enter a client secret (this is the admin password for this UAA instance). The client secret can be any alphanumeric string. **Note:** Record the client secret in a secure place for later use.
Subdomain	(Optional) Enter a subdomain you might need to use in addition to the domain created for UAA. You must not add special characters in the name of the subdomain. The value of sub-domain is case-insensitive.

5. Click **Create Service**.

Results

Your UAA instance is created with the following specifications:

- A client identifier (`admin`).

Note: An `admin` client is required for bootstrap purposes. You can create additional clients to use with your application.

- A client secret (that you specified while creating the service).

To retrieve additional details of your instance, you can bind an application to your instance.

Using the Command Line to Create a UAA Service Instance

Optional procedure for using the command line instead of the graphical user interface to create a UAA service instance.

About This Task

You can create up to 10 instances of UAA service in your space. If you need additional instances, you must delete an older unused instance and create a new one.

Procedure

1. Use the Cloud Foundry CLI to log into Cloud Foundry.

   ```
   cf login -a <API_Endpoint>
   ```

   Note: If you are a GE employee, you must use the `cf login --sso` command to log into Cloud Foundry. After you enter your SSO, you will receive a one-time passcode URL. Copy this URL and paste it in a browser to retrieve your one-time passcode. Use this code with the `cf` command to complete the CF login process.

   Depending on your Predix.io registration, the value of `<API_Endpoint>` is one of the following:

   - Predix US-West
     `https://api.system.aws-usw02-pr.ice.predix.io`
   - Predix US-East
     `https://api.system.asv-pr.ice.predix.io`
   - Predix Europe
     `https://api.system.aws-eu-central-1-pr.ice.predix.io`

   For example,

   ```
   cf login -a https://api.system.aws-usw02-pr.ice.predix.io
   ```

2. List the services in the Cloud Foundry marketplace by entering the following command.

   ```
   cf marketplace
   ```

   The UAA service, `predix-uaa`, is listed as one of the available services.

3. Create a UAA instance by entering the following command.

   ```
   cf create-service predix-uaa <plan> <my_uaa_instance> -c
   '{"adminClientSecret":"<my_secret>","subdomain":"<my_subdomain>"}'
   ```

   where:

   - `cf` stands for the CLI command, `cloud foundry`
   - `cs` stands for the CLI command `create-service`
   - `<plan>` is the plan associated with a service. For example, you can use the `tiered` plan for the `predix-uaa` service.

- `-c` option is used to specify following additional parameters.
  - `adminClientSecret` specifies the client secret.
  - `subdomain` specifies a sub-domain you might need to use in addition to the domain created for UAA. This is an optional parameter. You must not add special characters in the name of the sub-domain. The value of sub-domain is case insensitive.

Note: Cloud Foundry CLI syntax can differ between Windows and Linux operating systems. See the Cloud Foundry help for the appropriate syntax for your operating system. For example, to see help for the `create service` command, run `cf cs`.

Results

Your UAA instance is created with the following specification:

- A client identifier (`admin`).

  Note: An `admin` client is created for bootstrap purposes. You can create additional clients to use with your application.
- A client secret (that you specified while creating the service).

To retrieve additional details of your instance, you can bind an application to your instance.

Example

Create a predix-uaa service instance with client secret as admin and sub-domain as ge-digital:

```
cf cs predix-uaa tiered test-1 -c
'{"adminClientSecret":"admin","subdomain":"ge-digital"}'
```

This is how it appears in VCAP SERVICES when using the `cf env <app_name>` command:

```
"VCAP_SERVICES": {
"predix-uaa": [
   {
    "credentials": {
     "dashboardUrl": "https://uaa-dashboard.run.asv-
pr.ice.predix.io/#/login/04187eb1-
e0cf-4874-8218-9fb77a8b4ed9",
     "issuerId": "https://04187eb1-
e0cf-4874-8218-9fb77a8b4ed9.predix-uaa.run.asv-
pr.ice.predix.io/oauth/token",
     "subdomain": "04187eb1-e0cf-4874-8218-9fb77a8b4ed9",
     "uri": "https://04187eb1-
e0cf-4874-8218-9fb77a8b4ed9.predix-uaa.run.asv-
pr.ice.predix.io",
     "zone": {
      "http-header-name": "X-Identity-Zone-Id",
      "http-header-value": "04187eb1-
e0cf-4874-8218-9fb77a8b4ed9"
     }
    },
    "label": "predix-uaa",
    "name": "testuaa",
    "plan": "Tiered",
    "provider": null,
    "syslog_drain_url": null,
    "tags": [],
```

```
    "volume_mounts": []
   }
  ],
```

Creating an ACS Instance

You can create an instance of access control service for setting up fine-grained access permissions for users. You can create a maximum of 200 instances of ACS in your space.

Before You begin

An instance of the UAA service has been configured as your trusted issuer. See Task Roadmap: Setting Platform Services.

Procedure

1. Sign into your Predix account at https://www.predix.io.
2. Navigate to **Catalog > Services** tab, and click the **Access Control Service** tile.
3. Click **Subscribe** on the required plan.
4. On the **New Service Instance** page, enter:

Field	Description
Org	Select your org.
Space	Select the space for your application.
User Account & Authentication (UAA)	Choose an existing UAA instance or create a new instance of UAA. For more information, see Creating a UAA Service Instance on page 6.
Service instance name	Specify a unique name for your instance.
Service plan	Select a plan.

5. Click **Create Service**.

Using Command Line to Create an ACS Instance

Procedure

1. Use the Cloud Foundry CLI to log into Cloud Foundry.

```
cf login -a <API_Endpoint>
```

Note: If you are a GE employee, you must use the `cf login --sso` command to log into Cloud Foundry. After you enter your SSO, you will receive a one-time passcode URL. Copy this URL and paste it in a browser to retrieve your one-time passcode. Use this code with the `cf` command to complete the CF login process.

Depending on your Predix.io registration, the value of `<API_Endpoint>` is one of the following:

- Predix US-West
  `https://api.system.aws-usw02-pr.ice.predix.io`
- Predix US-East
  `https://api.system.asv-pr.ice.predix.io`

- Predix Europe
  `https://api.system.aws-eu-central-1-pr.ice.predix.io`

Note: If you are registered on the Predix Azure domain, you must use the command-line instructions to create your service.

For example,

```
cf login -a https://api.system.aws-usw02-pr.ice.predix.io
```

2. List the services in the Cloud Foundry marketplace.

```
cf marketplace
```

The ACS service, `predix-acs`, is listed as an available service.

3. Create an ACS service instance.

```
cf create-service predix-acs <plan> <my_acs_instance> -c
'{"trustedIssuerIds":["<uaa_instance1_issuerId>",
"<uaa_instance2_issuerID>"]}'
```

where:

- `<plan>` is the plan associated with a service. For example, you can use `basic` for the `predix-acs` service.
- `<uaa_instance_issuerID>` is the `issuerID` of your trusted issuer (UAA instance). For example, `https://13fa0384-9e2a-48e2-9d06-2c95a1f4f5ea.predix-uaa.run.aws-usw02-pr.ice.predix.io/oauth/token`. You can use a comma-separated list to specify multiple trusted issuers. You can retrieve this URL from the VCAP_SERVICES environment variable after binding an application to an UAA Instance..

Note: Cloud Foundry CLI syntax can differ between Windows and Linux operating systems. See the Cloud Foundry help for the appropriate syntax for your operating system. For example, to see help for the `create service` command, run `cf cs`.

The message on the screen indicates that the ACS instance is created. You can retrieve the URI of this instance from the VCAP_SERVICES environment variable after binding it to an application.

Updating an ACS Instance

Before You begin

You need the name of the ACS instance `<my_acs_instance>` that you created (see Creating an ACS Instance on page 9).

Note: Cloud Foundry CLI syntax can differ between Windows and Linux operating systems. See the Cloud Foundry help for the appropriate syntax for your operating system. For example, to see help for the `create service` command, run `cf cs`.

Procedure

1. Use the Cloud Foundry CLI to log into Cloud Foundry.

```
cf login -a <API_Endpoint>
```

Note: If you are a GE employee, you must use the `cf login --sso` command to log into Cloud Foundry. After you enter your SSO, you will receive a one-time passcode URL. Copy this URL and paste it in a browser to retrieve your one-time passcode. Use this code with the `cf` command to complete the CF login process.

Depending on your Predix.io registration, the value of `<API_Endpoint>` is one of the following:

- Predix US-West
  `https://api.system.aws-usw02-pr.ice.predix.io`
- Predix US-East
  `https://api.system.asv-pr.ice.predix.io`
- Predix Europe
  `https://api.system.aws-eu-central-1-pr.ice.predix.io`

For example,

```
cf login -a https://api.system.aws-usw02-pr.ice.predix.io
```

2. Update the ACS service instance `<my_acs_instance>` that you previously created.

```
cf update-service <my_acs_instance> -c '{"trustedIssuerIds":
["<uaa_instance1_url>", "<uaa_instance2_url>",...]}'
```

where:

- `<uaa_instance_url>` is the URL of your trusted issuer (UAA instance). For example, `https://13fa0384-9e2a-48e2-9d06-2c95a1f4f5ea.predix-uaa.run.aws-usw02-pr.ice.predix.io/oauth/token`.You can retrieve this URL from the VCAP_SERVICES environment variable after binding an application to an UAA Instance..

  Note:

  You can use a comma-separated list to specify multiple trusted issuers. Here is a sample that you can use as a template:

  ```
  cf update-service my-predix-acs -c '{"trustedIssuerIds":
  ["https://my-uaa-1.predix-uaa.run.aws-usw02-pr.ice.predix.io/oauth/
  token","https://my-uaa-2.predix-uaa.run.aws-usw02-pr.ice.predix.io/
  oauth/token"]}'
  ```

The update will not take effect immediately. The trusted issuers list will be updated every 24 hours.

Creating an OAuth2 Client

You can create OAuth2 clients with specific permissions for your application to work with Predix Platform services. Often this is the first step after creating an instance of a service.

About This Task

When you create an instance of UAA, the UAA Dashboard is available for configuring that instance of UAA. You can use the Client Management tab in the UAA Dashboard to create the OAuth2 clients.

If you are prefer using the UAA command-line interface (UAAC) instead of UAA Dashboard to create an OAuth2 client, see Using UAAC to Create an OAuth2 Client

Procedure

1. In the Predix.io Console view, select the Space where your services are located.
2. In the Services Instances page, select the UAA instance to configure.
3. Select the **Configure Service Instance** option.
4. In the UAA Dashboard login page, specify your admin client secret and click **Login**.
5. In UAA Dashboard, select the **Client Management** tab.

   The Client Management tab has two views, **Clients** and **Services** . The **Services** view displays the service instances that you have created for your services.

   Note: The service instances displayed in the Services view were created while using the UAA that you are trying to configure. Service instances that you created using other UAA instances are not displayed on this page.
6. Click **Create Client** to open the **Create Client** form.
7. Complete the **Create Client** form.

Field	Description
Client ID	Specify a name for the OAuth2 client you are creating.
Authorized Grant Types	Choose one or more of the following grant types: • **authorization_code** When you use the authorization code grant type, the client directs the resource owner to UAA, which in turn directs the resource owner back to the client with the authorization code. • **client_credentials** When you use the client credentials grant type, the OAuth2 endpoint in UAA accepts the client ID and client secret and provides Access Tokens. • **password** When you use the resource owner password credentials grant type, the OAuth2 endpoint in UAA accepts the username and password and provides Access Tokens. • **refresh_token** The refresh tokens are credentials used to obtain access tokens. You can choose this option to obtain refresh token from UAA. You can then use the refresh token to obtain a new access token from UAA when the current access token becomes invalid or expires, or to obtain additional access tokens with identical or narrower scope. • **implicit** When you use the implicit grant type, UAA directly issues an Access Token to the client without authenticating the client. This reduces the number of round trips required to obtain an access token. For more information on grant types, see RFC 6749.
Client Secret	Specify the password. It is important that you keep a note of this password. If lost, this password cannot be retrieved.
Confirm Client Secret	Reenter the client secret.

Field	Description
Redirect URI	Specify a redirect URI to redirect the client after login or logout (for example, `http://example-app.com/callback`). Use this URI when you start using UAA as the service provider for your external Identity provider. UAA uses the value of Redirect URI for `/oauth/authorize` and `/logout` endpoints. You must specify a Redirect URI value if you use the Authorization Code or Implicit authorization grant type. When you use the Authorization Code grant type, the Redirect URI is your application's endpoint or callback that expects user authorization code. When you use the Implicit grant type, the Redirect URI is the end point where UAA sends the bearer token. Unique Resource Identifier consists of: • Access Protocol, `http` or `https` • Domain or IP address • Access Port such as 80 or 443 • Path If you have a specific URL for your application callback, you can use that to set the Redirect URI value for the related client. For example, `https://your-app-domain.run.aws-usw02-pr.ice.predix.io/path1/path2/callback`. You can specify multiple values for Redirect URI as a list of allowed destinations that UAA server can redirect the users. For example, `https://yourappdomain1.run.aws-usw02-pr.ice.predix.io/path1/path2/callback`, `https://yourappdomain2.run.aws-usw02-pr.ice.predix.io/path1/path2/callback`. If the subdomain of your application is dynamic, you can set the value of Redirect URI using wilcards. For example, `https://*.your-app-domain.run.aws-usw02-pr.ice.predix.io/path1/path2/callback`. **Note:** You must only use '*' for a domain that is exclusive to your application (Such as `your-app-domain` in example above). This prevents the redirect to be routed to an application that you do not own. You cannot use `*` in the top domain and sub domain (such as `predix.io` in the example above).
Scopes	Scopes are permissions associated with an OAuth Client to determine user access to a resource through an application. The user permissions are for authorization grant types `authorization_code`, `password` and `implicit`. By default, the admin client is assigned all required scopes. For a new client, an administrator can select the scopes to be added based on client requirements. For a list of available scopes, see Scopes Authorized by the UAA. To use an OAuth2 client for your Predix Platform service instance, you must update your OAuth2 client to add scopes that are specific to each service after adding the client to the service instance.

Field	Description
Authorities	Authorities are permissions associated with the OAuth Client when an application or API is acting on its own behalf to access a resource with its own credentials, without user involvement. The permissions are for the `client_credentials` authorization grant type. By default, the admin client is assigned all required authorities. For a new client, an administrator can select the authorities to be added based on client requirements. The list of authorities matches the list of scopes. For a list of available UAA scopes, see Scopes Authorized by the UAA. To use an OAuth2 client for your Predix Platform service instance, you must update your OAuth2 client to add authorities that are specific to each service after adding the client to the service instance. **Note:** An admin client is not assigned the default authority to change the user password. To change the user password, you must add the `uaa.admin` authority to your admin client.
Auto Approved Scopes	Specify scopes that can be approved automatically for the client without explicit approval from a resource owner.
Allowed Providers	Specifies the names of the external identity providers, if any. This field is required if you are using external identity providers with UAA as a service provider.
Access Token Validity	Specifies the access-token expiration time in ms.
Refresh Token Validity	Specifies the refresh-token expiration time in ms.

Next Steps

Updating the OAuth2 Client for Services on page 14 for your service specific information.

Updating the OAuth2 Client for Services

To use an OAuth2 client for secure access to your Predix Platform service instance from your application, you must update your OAuth2 client to add additional authorities or scopes that are specific to each service.

About This Task

To enable your application to access a platform service, your JSON Web Token (JWT) must contain the scopes required for a platform service. For example, some of the scope required for Access Control service are `acs.policies.read acs.policies.write`.

The OAuth2 client uses an authorization grant to request an access token. Based on the type of authorization grant that you have used, you must update your OAuth2 client to generate the required JWT. For more information on how the OAuth2 client is created, see Creating OAuth2 client.

If you use the UAA Dashboard to create additional clients, the client is created for the default `client_credentials` grant type. Some required authorities and scopes are automatically added to the client. You must add additional authorities or scopes that are specific to each service.

In addition, the admin client is not assigned the default authority to change the user password. To change the user password, you must add the `uaa.admin` authority to your admin client.

Use the following procedure to update the OAuth2 client.

Procedure

1. In the Console view, select the Space where your services are located.

2. In the Services Instances page, select the UAA instance to configure.
3. Select the **Configure Service Instance** option.
4. In the UAA Dashboard login page, specify your admin client secret and click **Login**.
5. In UAA Dashboard, select the **Client Management** tab.

   The Client Management tab has two views, **Clients** and **Services**. The **Services** view displays the service instances that you have created for your services.

   Note: The service instances displayed in the **Services** view are the instances that you created using the UAA that you are trying to configure. The service instances that you created using some other UAA instance are not displayed on this page.
6. Select the **Switch to Services View** option.
7. In the **Services** view, select the service that you need to update.
8. Choose an existing client or choose the **Create a new client** option. If you chose to create a new client, follow the steps in Creating an OAuth2 Client on page 11.
9. Click **Submit**.
10. Click on the **Switch to Clients View** option.
11. In the **Clients** view, click the edit icon corresponding to the client added in the previous step.
12. Complete the **Edit Client** form.

Field	Description
Authorized Grant Types	Choose one or more of the following grant types: • **authorization_code** When you use the authorization code grant type, the client directs the resource owner to UAA, which in turn directs the resource owner back to the client with the authorization code. • **client_credentials** When you use the client credentials grant type, the OAuth2 endpoint in UAA accepts the client ID and client secret and provides Access Tokens. • **password** When you use the resource owner password credentials grant type, the OAuth2 endpoint in UAA accepts the username and password and provides Access Tokens. • **refresh_token** The refresh tokens are credentials used to obtain access tokens. You can choose this option to obtain refresh token from UAA. You can then use the refresh token to obtain a new access token from UAA when the current access token becomes invalid or expires, or to obtain additional access tokens with identical or narrower scope. • **implicit** When you use the implicit grant type, UAA directly issues an Access Token to the client without authenticating the client. This reduces the number of round trips required to obtain an access token. For more information on grant types, see RFC 6749.
Redirect URI	Specify a redirect URI to redirect the client after login (for example, `http://example-app.com/welcome`). This URI is used when you start using UAA as service provider for your external Identify provider.
Scopes	By default, the client is assigned a few required scopes. For a new client, an administrator can select the scopes to be added based on the selected grant type. If you select the `authorization_code`, `password` and `implicit` grant type, you must update the scopes with service specific scopes. For a complete list of required scopes, see Authorities or Scopes Required for Platform Services. For a list of available UAA scopes, see Scopes Authorized by the UAA.

Field	Description
Authorities	By default, the client is assigned a few required authorities. For a new client, an administrator can select the authorities to be added based on the selected grant type. If you select the `client_credentials` grant type, you must update the authorities with service specific authorities. For a complete list of scopes to be added for each service, see Authorities or Scopes Required for Platform Services. For a list of available UAA authorities, see Scopes Authorized by the UAA.
Auto Approved Scopes	Specify scopes that can be approved automatically for the client without explicit approval from the resource owner.
Allowed Providers	Specify the names of the external identity providers, if any. This field is required if you are using external identity providers with UAA as a service provider.
Access Token Validity	Specifies the access token expiration time in ms.
Refresh Token Validity	Specifies the refresh token expiration time in ms.

Next Steps

You can complete the following additional tasks in UAA Dashboard:

- If you are using authorization grant type as Authorization Code, Implicit, or Resource Owner Password, you can manage users in UAA.
- You can create password policies for user passwords.
- You can set up external identity provider or use UAA as an identity provider. See Managing Identity Providers.

If you have completed your OAuth2 client setup, you can bind your application to your service instance.

Authorities or Scopes Required for ACS Services

List of scopes and authorities specific to ACS service that you must add to your OAuth2 client.

When you create a new OAuth2 client, the client is assigned default scopes and authorities. You must add additional authorities or scopes that are specific to each service.

- acs.policies.read
- acs.policies.write
- acs.attributes.read
- acs.attributes.write
- predix-acs.zones.<acs_instance_guid>.user
  This value is added by default if you use UAA Dashboard. It is also generated in the VCAP_SERVICES environment variable as `oauth-scope` when you bind your application to your ACS service instance.

Binding an Application to the ACS Instance

About This Task

You must bind your application to your ACS instance to provision its connection details in the VCAP_SERVICES environment variable. Cloud Foundry runtime uses VCAP_SERVICES environment variable to communicate with a deployed application about its environment.

You can retrieve the following ACS instance details from the VCAP_SERVICES environment variable:

- A `acs_instance_uri` for your instance.
- HTTP header information to access your ACS instance. It includes:
  - `http-header-name` as `Predix-Zone-Id`
  - `http-header-value`
- An `oauth-scope` for your instance. The scope is required in the end-user token to access a specific ACS instance.

Note: The following steps are performed using the Cloud Foundry CLI. To complete the steps in a web browser, follow the instructions on the service page in the Predix Catalog.

Procedure

1. Bind your application to the new ACS instance.

   ```
   cf bind-service <your_app_name> <acs_instance_name>
   ```

   The `<acs_instance_name>` instance is bound to your application, and the following message is returned:

   ```
   Binding service <acs_instance_name> to app <your_app_name> in org
   predix-platform / space predix as userx@ge.com...
   OK
   TIP: Use 'cf restage' to ensure your env variable changes take effect
   ```

2. Verify the binding:

   ```
   cf env <your_app_name>
   ```

   Messages that are similar to the following messages are returned:

   ```
   Getting env variables for app myApp in org predix-platform / space
   security as userx@ge.com...
   OK
   ...
     ],
     "predix-acs": [
           {
           "credentials":
           {
            "uri": "http://predix-acs.run.aws-usw02-pr.ice.predix.io",
             "zone": {
                       "http-header-name": "Predix-Zone-Id",
                       "http-header-value": "9378e3db-e683-46a2-97c2-
   ccd11d75869d"
                       "oauth-scope": "predix-acs.zones.9378e3db-
   e683-46a2-97c2-ccd11d75869d.user"
                     }
           },
           "label":
           "predix-acs",
           "name":
           "my_acs_instance",
           "plan":
           "free",
           "tags":
           []
   ```

```
            }
            ],
```

Using Access Control Services

Creating Attributes for Resources and Subjects

Use the Attribute Management service to define subject and resource attributes for access-control policies.

Before You begin

To use the Attribute Management service, the user/client must authenticate using a JSON Web Token (JWT) that includes the `acs.attributes.read`, `acs.attributes.write` and `predix-acs.zones.<acs_instance_guid>.user` scopes.

You must specify the `Predix-Zone-Id` header when you use this REST API to communicate with a deployed application about its environment.. For more information, see Binding an Application to the ACS Instance on page 16.

Select a REST client (such as Postman) that can execute API requests for the Attribute Management service. For more information about this API, see the API Documentation (Predix.io home page – Documentation – Service APIs).

About This Task

Predix administrators define attributes about users and resources to evaluate conditions for authentication and authorization in access control policies.

Creating Attributes for a Resource

An administrator creates attributes for a resource based on the following API request.

```
HTTP PUT
https://<acs_url>/v1/resource/{resourceIdentifier}
```

The HTTP request body can have the following elements:

```
{
  "attributes": [
    {
      "issuer": "string",
      "name": "string",
      "value": "string"
    }
  ],
  "parents": [
    {
      "identifier": "string",
      "scopes": [
        {
          "issuer": "string",
          "name": "string",
          "value": "string"
        }
      ]
    }
  ],
```

```
  "resourceIdentifier": "string"
}
```

For example:

```
HTTP PUT
https://predix-acs.run.aws-usw02-dev.ice.predix.io/v1/resource/
asset123
{ "attributes": [ { "issuer": "https://acs.attributes.int", "name":
"site", "value": "sanfrancisco" } ], "resourceIdentifier":
"asset123" }
```

Creating Attributes for a Subject

An administrator creates attributes for a subject based on the following API request.

```
HTTP PUT
http://<host>/v1/{subjectIdentifier}
```

The HTTP request body can have the following elements:

```
{
  "attributes": [
    {
      "issuer": "string",
      "name": "string",
      "value": "string"
    }
  ],
  "parents": [
    {
      "identifier": "string",
      "scopes": [
        {
          "issuer": "string",
          "name": "string",
          "value": "string"
        }
      ]
    }
  ],
  "subjectIdentifier": "string"
}
```

The HTTP request body specifies a collection of attributes that a subject must possess for the policy to be considered. At run time, the Policy Evaluation service compares the attributes of the user making a service request against the criteria specified in the `subject` specification of an access-control policy to determine if it applies to the incoming request.

Procedure

1. Use the following REST API in a REST client to define the attributes of a resource:

```
HTTP PUT
https://<host>/v1/{resourceIdentifier}
```

For example:

```
HTTP PUT
https://acs.attributes.int/v1/Site1
```

2. Enter the following information in the HTTP request body:

```
{
  "attributes": [
    {
      "issuer": "string",
      "name": "string",
      "value": "string"
    }
  ],
  "parents": [
    {
      "identifier": "string",
      "scopes": [
        {
          "issuer": "string",
          "name": "string",
          "value": "string"
        }
      ]
    }
  ],
  "resourceIdentifier": "string"
}
```

For example:

```
{
  "attributes": {
          "name": "Site 1",
          "attributes": [
              {
                  "uriTemplate": "/customers/{customer_id:\\w*}/
sites/{site_id:\\w*}"
              },
      ]
```

A 201 status code is returned if the resource is created successfully.

3. In a REST client, use the following REST API to define the attributes of a subject:

```
HTTP PUT
http://<host>/v1/{subjectIdentifier}
```

For example:

```
HTTP PUT
https://acs.attributes.int/v1/Boss
```

4. Enter the following information in the HTTP request body:

```
{
  "attributes": [
    {
      "issuer": "string",
      "name": "string",
```

```
      "value": "string"
    }
  ],
  "parents": [
    {
      "identifier": "string",
      "scopes": [
        {
          "issuer": "string",
          "name": "string",
          "value": "string"
        }
      ]
    }
  ],
  "subjectIdentifier": "string"
}
```

For example:

```
{
"attributes": {
          "name": "Site Director role",
          "attributes": [
            {
              "issuer": "https://acs.attributes.int",
              "name": "role"
            }
          ]
```

A 201 status code is returned if the subject `Boss` is created successfully with `Site Director role` attributes.

Next Steps

See Creating Access-Control Policies on page 22 for information on how to set up access-control policies based on resource and subject attributes.

Creating Access-Control Policies

Use the Policy Management service to create, read, update, and delete access-control policies.

Before You begin

To use the Policy Management service, the user/client must authenticate using a JSON Web Token (JWT) that includes the `predix-acs.zones.<acs_instance_guid>.user` scope along with `acs.policies.read` scope for reading the policies or the `acs.policies.write` scope for writing the policies.

You must specify the `Predix-Zone-Id` header when you use this REST API to communicate with a deployed application about its environment.. For more information, see Binding an Application to the ACS Instance on page 16.

Select a REST client (such as Postman) that can execute API requests for the Attribute Management service. For more information about this API, see the API Documentation (Predix.io home page – Documentation – Service APIs).

About This Task

An access-control policy contains a set of rules to determine the required permissions for the specified subjects and resources. The rules can take into consideration the user attributes, the action the user wants to perform, the resource URI, and any resource attributes that further describe the resource.

An administrator creates an access-control policy as a JSON string so that the Policy Evaluation service can handle web service requests for authorization. An administrator can combine access-control policy sets as ordered lists.

An administrator creates an access-control policy based on the following API request.

```
HTTP PUT
/v1/policy-set/{policySetId}
```

The HTTP request body can have the following elements:

```
{
  "name": "string",
  "policies": [
    {
      "conditions": [
        {
          "condition": "string",
          "name": "string"
        }
      ],
      "effect": "PERMIT",
      "name": "string",
      "target": {
        "action": "string",
        "name": "string",
        "resource": {
          "attributeUriTemplate": "string",
          "attributes": [
            {
              "issuer": "string",
              "name": "string",
              "value": "string"
            }
          ],
          "name": "string",
          "uriTemplate": "string"
        },
        "subject": {
          "attributes": [
            {
              "issuer": "string",
              "name": "string",
              "value": "string"
            }
          ],
          "name": "string"
        }
      }
    }
  ]
}
```

The main elements of an access-control policy are listed in the following table:

Element	Description
`name`	Optional information to identify a policy.

`target`	Specifies the matching criteria to determine if a particular policy is applicable to an incoming evaluation request. This element is optional. A target contains the following basic components that must match the context of an incoming request to make the policy applicable:
	• `resource`
	Specifies a URI template that the resource must match, and a collection of attributes the resource must possess, for the policy to apply. At run time, the Policy Evaluation service compares the attributes of a requested resource against the criteria in the `resource` specification within the policy `target` to determine if the policy is applicable to the incoming request.
	For example,
	`...`
	`"resource" : {`
	`"uriTemplate" : "/v1/region/report/asset/{asset_id}",`
	`"attributeUriTemplate": "/v1/region/report{attribute_uri}",`
	`"attributes" : [`
	`{ "issuer" : "https://acs.attributes.int",`
	`"name" : "site" }`
	`]`
	`}`
	`...`
	The resource criteria can contain the following values:
	◦ `uriTemplate`: Specifies the URI template that the resource URI in the evaluation request must match.
	In the example above, the `uriTemplate` is defined as `/v1/region/report/asset/{asset_id}`. Therefore this policy will be considered for an evaluation request with resource URI as `/v1/region/report/asset/1234`. For more information on URI templates, see Spring Framework.
	◦ `attributeUriTemplate`: Specifies the attribute URI template to extract a contiguous subset of resource URI.
	In the example above, the `attributeUriTemplate` is defined as `/vi/region/report{attribute_uri}`. An evaluation request for this policy with resource URI as `/v1/region/report/asset/1234` will refer to resource attributes from `/asset/1234` instead of `/v1/region/report/asset/1234`.
	◦ `attributes`: Specifies the attributes that the resources must have for the policy to be considered.
	• `action`
	Specifies the following action in the RESTful endpoint that the policy permits for a specified resource.
	◦ `GET`

Element	Description
	◦ `PUT, POST` ◦ `DELETE` You can specify multiple actions at a time as a comma separated list. If you specify multiple actions, the service uses any of the actions that matches the policy. If you do not specify an action, all operations are considered. • `subject` Specifies a collection of attributes that the user must possess for the policy to apply. At run time, the Policy Evaluation service compares the attributes of the user/agent making a service request against the criteria specified in the `subject` specification of the policy `target` to determine if the policy applies to the incoming request.
`effect`	Specifies the access-control decision the Policy Evaluation service returns when the policy target matches a request, and all policy conditions return `true`. The `effect` can be either `DENY` or `PERMIT`. **Note:** If a policy condition returns `false`, the policy does not apply, and the overall state of the access control decision is `NOT_APPLICABLE`.

`condition`	Specifies a predicate that must be satisfied for a rule to be assigned its effect. This element is optional.
	If you specify multiple conditions, the default Boolean expression used between conditions is AND.
	You can use the following evaluation methods:
	• `attributes(attributeIssuer, attributeName)`
	Returns a set of all values of a specific attribute. For example, the following condition returns all groups a subject belongs to:
	`"condition" : "subject.attributes('https://acs.attributes.int', 'group')"`
	Similarly the following condition evaluates to true if the subject and resource belong to the same groups:
	`"condition" : "subject.attributes('https://acs.attributes.int', 'group').equals(resource.attributes('https://acs.attributes.int', 'group'))"`
	For complete list of operations, see Groovy JDK API Documentation
	• `match.single`
	Returns true if attribute value is present in the source attribute values. For example, the following condition evaluates to true if subject has the records-viewer role:
	`"condition" : "match.single(subject.attributes('https://acs.attributes.int', 'role'), 'records-viewer')"`

Element	Description

	• `match.any`
	Returns true if the intersection of source attributes and target attributes is non-empty. For example, the following condition evaluates to true if subject and resource belong to at least one common group:
	`"condition" : "match.any(subject.attributes('https://acs.attributes.int', 'group'), resource.attributes('https://acs.attributes.int', 'group'))"`
	Similarly, the following condition evaluates to true if subject name is listed among resource owners:
	`"condition" : "match.any(subject.attributes('https://acs.attributes.int', 'name_id'), resource.attributes('https://acs.attributes.int', 'owner'))"`
	• `resource.uriVariable`
	Returns a value for the path parameter in the resource URL.
	For example, the following condition validates that the user (subject) and resource are assigned to the same customer. The user can be assigned to multiple customers. However, the condition in this sample validates that at least one customer matches the resource URL.
	`"condition" : "match.single(subject.attributes('https://acs.attributes.int', 'customer'), resource.uriVariable('customer_id'))"`
	This example assumes that the following URI template defines the resource URL in the policy target:
	`"/customers/{customer_id}/sites/{site_id}"`
	• `and`, `haveSame()`
	Used together to validate that subject and resource have common attributes. For example, the following condition evaluates to true if subject and resource belong to at least one common group.
	`"condition" : "resource.and(subject).haveSame('acs.example.org', 'group').result()"`
	You can use the following classes in a condition:
	• `com.ge.predix.acs.commons.policy.condition.groovy.GroovySecureExtension`
	• `com.ge.predix.acs.commons.policy.condition.AbstractHandler`

Element	Description
	• com.ge.predix.acs.commons.policy.condition.AbstractHandlers • com.ge.predix.acs.commons.policy.condition.ResourceHandler • com.ge.predix.acs.commons.policy.condition.SubjectHandler • com.ge.predix.acs.commons.policy.condition.groovy.AttributeMatcher • java.lang.Boolean • java.lang.Integer • java.lang.Iterable • java.lang.Object • java.lang.String • java.util.Collection • java.util.Set The following methods are not allowed in a condition: • java.lang.System • groovy.util.Eval • java.io • execute

Procedure

1. In a REST client, use the following REST API to create an access-control policy set for a zone:

   ```
   HTTP PUT
   /v1/policy-set/{policySetId}
   ```

   where `{policySetId}` is the name of the policy set.
2. Enter one of the the following sample JSON strings iin the HTTP request body to create a specific access-control policiy set:
   - The following sample policy denies access to everything.

     ```
     {
       "name" : "simple-policy-1",
       "policies" : [
           {
              "name" : "deny-everything",
              "effect" : "DENY"
           }
        ]
     }
     ```
   - The following sample policy permits access to a specified resource:

     ```
     {
       "name" : "simple-policy-1",
       "policies" : [
           {
              "name" : "deny-everything",
     ```

```
          "effect" : "DENY"
       }
    ]
}
```

- The following sample policy permits a `GET` action for a specified resource:

```
{
    "name" : "simple-policy-3",
    "policies" : [
        {
            "name" : "permit-get-to-public-records",
            "target" : {
                "name" : "",
                "action":"GET",
                "resource" : {
                    "name" : "public-records-resource",
                    "uriTemplate" : "/api/public-records/
{record_id}"
                }
            },
            "effect" : "PERMIT"
        }
    ]
}
```

- The following sample policy permits both `GET` and `POST` actions for a specified resource:

```
{
    "name" : "simple-policy-3a",
    "policies" : [
        {
            "name" : "permit-get-and-post-to-public-records",
            "target" : {
                "name" : "",
                "action":"GET,POST",
                "resource" : {
                    "name" : "public-records-resource",
                    "uriTemplate" : "/api/public-records/
{record_id}"
                }
            },
            "effect" : "PERMIT"
        }
    ]
}
```

- The following policy permits a `GET` action for a specified resource by a specified subject (role). The policy contains a condition that matches the role of the user.

```
{
    "name" : "simple-policy-4",
    "policies" : [
        {
            "name" : "permit-get-to-public-records",
            "target" : {
                "name" : "",
                "resource" : {
                    "name" : "public-records-resource",
                    "uriTemplate" : "/api/public-records/
{record_id}"
```

```
                },
                "action":"GET",
                "subject" : {
                    "name" : "has-role",
                    "attributes" : [
                        { "issuer" : "https://acs.attributes.int",
                          "name" : "role" }
                    ]
                }
            },
            "conditions" : [
                { "name" : "",
                  "condition" :
"match.single(subject.attributes('https://acs.attributes.int',
'role'), 'records-viewer')" }
            ],
            "effect" : "PERMIT"
        }
    ]
}
```

Next Steps

See Evaluating Access-Control Policies on page 34 for information on how to make access-control policy decisions..

Using Resource URI Templates in Access-Control Policies

A uniform resource identifier (URI) template allows you to define a set of structurally similar URIs. URI Templates are composed of two parts, a path and a query. A path consists of a series of segments delimited by a slash (/). Each segment can have a literal value or a variable value. Segments enclosed within curly braces `{}` are variables. You can omit the query expression entirely. When present, the query expression specifies an unordered series of name/value pairs. Elements of the query expression are either literal pairs (?x=2) or variable pairs (?x={val}). For more information on URI template, see Spring Framework UriTemplate .

In ACS, `resource URI template` is specified in the resource specification of an access-control policy target. Resource URI template specifies a template that the resource URI of the incoming evaluation request must match to determine if the policy is applicable to the request. The ACS Policy Evaluation service tests a candidate URI from the incoming evaluation request at run time to see if it matches a URI template in the policy to determine if the policy is applicable to the request. If the URI is a match, then template variables are resolved to the actual values from the URI and the rest of the policy criteria is evaluated. For example, consider the following policy:

```
{
  "name": "Agents can access a site if they are stationed at the
site.",
  "target": {
    "name": "When an agent accesses a site",
    "resource": {
      "name": "Site",
      "uriTemplate": "/sites/{site_id}"
    },
    "action": "GET",
    "subject": {
      "name": "Agent",
```

```
      "attributes": [
        {
          "issuer": "acs.example.org",
          "name": "site"
        }
      ]
    }
  },
  "conditions": [
    {
      "name": "is assigned to site",
      "condition": "match.single(subject.attributes('acs.example.org',
'site'), resource.uriVariable('site_id'))"
    }
  ],
  "effect": "PERMIT"
}
```

In the above example, the `uriTemplate` is defined as `"/sites/{site_id}"`. Therefore, this policy may be considered for evaluation requests with resource URI as `"/sites/siteA"`, `"/sites/siteB"` because these resource URIs match the template.

URI template variable `{site_id}` is resolved to the actual ID of the site from the URI: siteA or siteB, and is used to evaluate policy condition. If condition evaluates to TRUE, a PERMIT decision is returned.

However, note that the above `uriTemplate` will also match requests for `"/sites/siteA/financial-reports"` and `"/sites/siteB/financial-statements"` which might require a more restrictive permissions to access. Current implementation of ACS access-control policies allows the flexibility of allowing or blocking `subpaths`when matching resource URI by using regular expressions in URI template.

Allowing or Blocking Access to Subpaths

To control access to the resource URL `subpaths`, you can use regular expressions as part of the URI template. Using regular expressions as part of the URI template is optional. When used with the URI template, regular expressions allow you to implement a more fine-grained control on which resources can be accessed and which cannot. The syntax for using regular expressions in URI template is, `{variable:regex}`. The `regex` must be a valid expression following the Java Regular Expression specifications.

For example, to disallow access to the `subpaths` in resource URIs, change `uriTemplate` in the policy above to `"/sites/{site_id:\\w*}"`. This will ensure that this policy is considered for the requests where resource URI contains only alphanumeric characters after `"/sites/"`, disallowing any **slashes** and blocking the access to any **subpaths**. The Examples: URI Templates for Policy Matching on page 30 lists some examples that show how to use URI Templates to block access to various resources.

Examples: URI Templates for Policy Matching

The following examples list the URI Template Descriptions and Policy Examples that you can view to understand how you can modify your URI template to allow or block `subpaths`.

Example 1 — Allow or block subpaths at the end of resource URL

This policy will be considered by ACS Evaluation Service for HTTP GET requests to the resource URI that matches the following template: `/customers/{customer_id}`

```
{
  "name": "Administrators can access all the customers",
```

```
  "target": {
    "name": "When an administrator accesses customers",
    "resource": {
      "name": "Customers",
      "uriTemplate": "/customers/{customer_id}"
    },
    "action": "GET",
    "subject": {
      "name": "Administrator Role",
      "attributes": [
        {
          "issuer": "https://acs/attributes.int",
          "name": "role"
        }
      ]
    }
  },
  "conditions": [
    ...
  ],
  "effect": "PERMIT"
}
```

Problem — This URI template allows access to the URLs listed below as well as the `subpaths` that you may not want to be included in the list of accessible URLs:

- `/customers/12345` and `/customers/12345/sites`
- `/customers/12345/sites/siteA`
- `/customers/12345/sites/siteB`
- `/customers/all_possible_subpaths_after_this`

Solution — You can **disallow** access to the `subpaths` with the following modifications to the URI templates:

`"uriTemplate":"/customers/{customer_id:\\w*}"`

`\w` matches any alphanumeric character including underscore equivalent to `[A-Za-z0-9_]`

`"uriTemplate":"/customers/{customer_id:[^/]+}"`

Adding `"[^/]+"` matches any character after `/customers/` but should not contain `/` because including the slash may translate it as a `subpath` call.

These URI templates **match** the following URL patterns:

- `/customers/12345/` and `customers/customer1`
- `/customers/abc_123`

These URI templates **do not match** the following URL patterns:

- `/customers/12345/sites`
- `/customers/12345/sites/siteA`
- `/customers/12345/sites/siteB`
- `/customers/all_possible_subpaths_after_this`

Example 2 — Allow or block slash (/) at the end of resource URL

This policy will be considered by ACS Evaluation Service for HTTP GET requests to the resource URI that matches the following template: `/customers`

```
{
  "name": "Administrators can access all the customers",
  "target": {
    "name": "When an administrator accesses customers",
    "resource": {
      "name": "Customers",
      "uriTemplate": "/customers"
    },
    "action": "GET",
    "subject": {
      "name": "Administrator Role",
      "attributes": [
        {
          "issuer": "https://acs/attributes.int",
          "name": "role"
        }
      ]
    }
  },
  "conditions": [
    ...
  ],
  "effect": "PERMIT"
}
```

Problem —The `"uriTemplate": "/customers"` template will block access to `/customers/`

Solution — To allow access to both `/customers` and `/customers/`, the URI Template must be modified to the following:

`"uriTemplate":"customers{optionalSlash:/?}"` This URI now **matches** `/customers` and `/customers/` but it **does not match** `/customers/all_possible_subpaths_after_this`.

Example 3 — Allow or block subpaths in the middle and at the end of resource URL

This policy will be considered by ACS Evaluation Service for HTTP GET requests to the resource URI that matches the following template: `/customers/{customer_id}/sites/{site_id}`

```
{
  "name": "Administrators can access all the customers",
  "target": {
    "name": "When an administrator accesses customers",
    "resource": {
      "name": "Customers",
      "uriTemplate": "/customers/{customer_id}/sites/{site_id}"
    },
    "action": "GET",
    "subject": {
      "name": "Administrator Role",
      "attributes": [
        {
          "issuer": "https://acs/attributes.int",
          "name": "role"
        }
```

```
      ]
    }
  },
  "conditions": [
    ...
  ],
  "effect": "PERMIT"
}
```

Problem — The `"uriTemplate": "/customers/{customer_id}/sites/{site_id}"` template **matches** the URL patterns including the `subpaths`, for example:

- `/customers/12345/sites/siteA`
- `/customers/123345/sites/siteB`
- `/customers/12345/sites/siteA/assets/asset-id`
- `/customers/all_possible_subpaths_here/sites/siteB`
- `/customers/all_possible_subpaths_here/sites/all_possible_subpaths_here`

Solution — To block access to the `subpaths` consider using the URI template with some modification as shown in the following example:

`"uriTemplate":"/customers/{customer_id:\\w*}/sites/{site_id:\\w*}"`

Using `\w` matches any alphanumeric character including underscore, equivalent to `[A-Za-z0-9_]`

`"uriTemplate":"/customers/{customer_id:[^/]+}/sites/{site_id:[^/]+}"`

Including the expression `"[^/]+"` in the template, matches any character after `/customers/` and `/sites/` but should not contain `/`which may be translated as a `subpaths` call.

These new URI templates **match** the following URL patterns:

- `/customers/12345/sites/siteA`
- `/customer/12345/sites/siteB`
- `/customer/customer1/sites/siteA`

These URI templates **do not match** the following URL patterns:

- `/customers/12345/sites`
- `/customers/12345/sites/siteA/assets/asset-id`
- `/customers/all_possible_subpaths_here/sites/siteB`
- `/customers/all_possible_subpaths_here/sites/all_possible_subpaths_here`

Example 4 — Allow or block subpaths inside resource URL

This policy will be considered by ACS Evaluation Service for HTTP GET requests to the resource URI that matches the following template: `/customers/{customer_id:\\w*}/sites`

```
{
  "name": "Administrators can access all the customers",
  "target": {
    "name": "When an administrator accesses customers",
    "resource": {
      "name": "Customers",
      "uriTemplate": "/customers/{customer_id:\\w*}/sites"
    },
    "action": "GET",
    "subject": {
      "name": "Administrator Role",
      "attributes": [
```

```
        {
          "issuer": "https://acs/attributes.int",
          "name": "role"
        }
      ]
    }
  },
  "conditions": [
    ...
  ],
  "effect": "PERMIT"
}
```

Problem — —This `"uriTemplate": "/customers/{customer_id:\\w*}/sites"` template matches all URL patterns and `subpaths` and allow all possible `subpaths` inside a resource URL .

Solution — Using `\\w*` ensures that the URI template **matches** only the following URL pattern examples:

- `/customers/12345/sites`
- `/customers/abcd/sites`

The URI template **do not match** the following URL pattern:

`/customers/all_possible_subpaths_here/sites`

Evaluating Access-Control Policies

Use the Policy Evaluation service to make access-control policy decisions.

Before You begin

You must specify the `Predix-Zone-Id` header when you use this REST API to communicate with a deployed application about its environment.. For more information, see Binding an Application to the ACS Instance on page 16.

Select a REST client (such as Postman) that can execute API requests for the Attribute Management service. For more information about this API, see the API Documentation (Predix.io home page – Documentation – Service APIs).

About This Task

The Policy Evaluation service performs access-control policy evaluation based on the following API request.

```
HTTP POST /v1/policy-evaluation
```

The HTTP policy evaluation request body can have the following elements:

```
{
  "action": "string",
  "policySetsEvaluationOrder": [
    "string"
  ],
  "resourceAttributes": [
    {
      "issuer": "string",
      "name": "string",
      "value": "string"
```

```
    }
  ],
  "resourceIdentifier": "string",
  "subjectAttributes": [
    {
      "issuer": "string",
      "name": "string",
      "value": "string"
    }
  ],
  "subjectIdentifier": "string"
}
```

where:

- `action`: Specifies the action in the RESTful endpoint that the policy permits for a specified resource. (see Creating Access-Control Policies on page 22).
- `policySetsEvaluationOrder`: Specifies the order that access-control policies are evaluated. The first access-control policy set that satisfies the specified conditions is executed
- `resourceAttributes`: Specifies a URI template that the resource must match, and a collection of attributes the resource must possess, for the policy to apply. At run time, the Policy Evaluation service compares the attributes of a requested resource against the criteria in the resource specification of an access-control policy to determine if the policy is applicable to the incoming request..
- `resourceIdentifier`: The name of the resource,
- `subjectAttributes` : Specifies a collection of attributes that the subject must possess for the policy to be considered. At run time, the Policy Evaluation service compares the attributes of the user making a service request against the criteria specified in the subject specification of an access-control policy to determine if it applies to the incoming request..
- `subjectIdentifier` : The name of the subject,

Procedure

1. In a REST client, use the following REST API to evaluate an access-control policy:

```
HTTP POST
/v1/policy-evaluation
```

2. Enter the following information in the HTTP request body:

```
{
  "action": "string",
  "policySetsEvaluationOrder": [
    "string"
  ],
  "resourceAttributes": [
    {
      "issuer": "string",
      "name": "string",
      "value": "string"
    }
  ],
  "resourceIdentifier": "string",
  "subjectAttributes": [
    {
      "issuer": "string",
      "name": "string",
      "value": "string"
    }
```

```
  ],
  "subjectIdentifier": "string"
}
```

For example:

```
{
  "resourceIdentifier": "/customers",
  "subjectIdentifier": "pbadmin",
  "action": "GET",
  "policySetsEvaluationOrder" : ["pb-policy-set-1","pb-policy-
set-2","pb-policy-set-3"],
  "subjectAttributes": [
    {
      "name": "role",
      "value": "PB_Admin",
      "issuer": "https://acs.attributes.int"
    }
  ],
  "tenantId": ""
}
```

Related concepts

Example: Access Control Services Simple Use Case on page 36

Example: Access Control Services Hierarchical Attribute Use Case on page 51

Related reference

Example: ACS Attribute Connector Setup on page 58

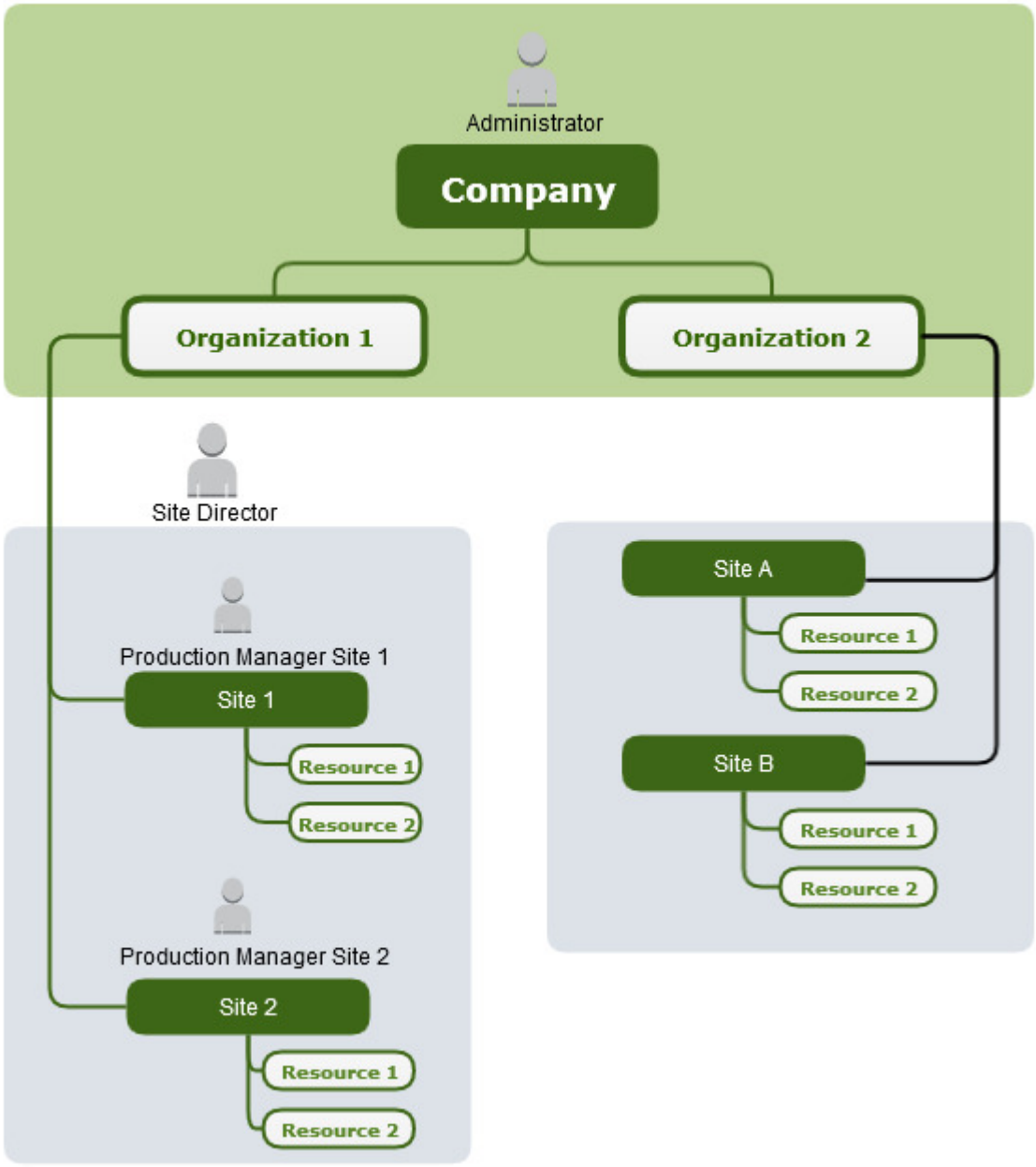
Configure access to subject and resource attributes from external resources for policy evaluation.

Example: Access Control Services Simple Use Case

A company has the following access-control requirements:

- An administrator should be able to add, remove or modify users in Organization 1 and Organization 2.
- A Site Director in Organization 1 should be able to access all the resources in all the sites in Organization 1 (Site 1 and Site 2).
- A Production Manager of Site 1 should only be able to access resources in Site 1.
- A Production Manager of Site 2 should only be able to access resources in Site 2.

The following diagram shows the company structure:



To implement this flow, an administrator can use the Policy Management service and the Attribute Management service APIs to specify the required policies and attributes and Policy Evaluation service to determine the access for web service calls.

The following sections demonstrate the use of ACS services for this use case:

- Policy Management
- Hierarchical Attribute Management
- Policy Evaluation

Before You Begin

As a developer, you should review the following resources to understand the conceptual, task, and reference information needed to demonstrate the use of ACS services for this use case::

- ACS Deployment Architecture on page 1
- Access Control Services Architecture on page 2
- Creating Access-Control Policies on page 22
- Creating Attributes for Resources and Subjects on page 19
- Evaluating Access-Control Policies on page 34

Select a REST client (such as Postman) that can execute API requests. For more information about these APIs, see the API Documentation (Predix.io home page – Documentation – Service APIs).

Policy Management

In a REST client, enter the following REST API to create an access control policy for a zone:

```
HTTP PUT
/v1/policy-set/{policySetId}
```

Note:

You must specify the `Predix-Zone-Id` header when you use the REST APIs.

In the response body, enter one of the following sample JSON strings to create a specific access-control policy:

- A policy that gives an administrator `HTTP GET` access to the `/customers` resource. It contains a condition that validates that the user has the `Administrator` role.

```
{
  "name": "sample-policy-set",
  "policies": [
    {
      "name": "Administrator can access all the customers.",
      "target": {
        "name": "When an Administrator accesses customers",
        "resource": {
          "name": "Customers",
          "uriTemplate": "/customers"
        },
        "subject": {
          "name": "Administrator role",
          "attributes": [
            {
              "issuer": "https://acs.attributes.int",
              "name": "role"
            }
          ]
        }
      },
      "conditions": [
        {
          "name": "is an Administrator",
          "condition":
    "match.single(subject.attributes('https://acs.attributes.int',
'role'),
    'Administrator')"
        }
```

```
      ],
      "effect": "PERMIT"
    }
    ]
}
```

- A policy that gives site directors `HTTP GET` access to the `/sites` resource. It contains a condition that validates that the user has the `Site_Director` role.

```
{
      "name": "Site Directors can read a sites/site if they are
assigned to the site.",
      "target": {
        "name": "When an Site Director reads his/her sites.",
        "resource": {
          "name": "Sites",
          "uriTemplate": "/sites"
        },
        "action": "GET",
        "subject": {
          "name": "Site Director role",
          "attributes": [
            {
              "issuer": "https://acs.attributes.int",
              "name": "role"
            }
          ]
        }
      },
      "conditions": [
        {
          "name": "is a Site Director",
          "condition":
    "match.single(subject.attributes('https://acs.attributes.int',
'role'),
    'Site_Director')"
        }
      ],
      "effect": "PERMIT"
    }
```

- A policy that gives site directors `HTTP GET` access to customers correlating to their sites. It contains the following conditions:
  - A condition that validates that the user has the `Site_Director` role.
  - A condition that validates that the user is assigned to the same customer. The user can be assigned to multiple customers. However, the condition in this sample validates that at least one customer matches the resource URL.

  Note:

  If you specify multiple conditions, the default Boolean expression used between conditions is AND.

```
{
      "name": "Site Directors can read a sites/site if they have
access to Customer",
      "target": {
        "name": "When an Site Director reads his/her sites.",
        "resource": {
          "name": "Customer Sites",
          "uriTemplate":
```

```
    "/customers/{customer_id:\\w*}/sites/{site_id}"
        },
        "action": "GET",
        "subject": {
          "name": "Site Director role",
          "attributes": [
            {
              "issuer": "https://acs.attributes.int",
              "name": "role"
            },
            {
              "issuer": "https://acs.attributes.int",
              "name": "customer"
            }
          ]
        }
      },
      "conditions": [
        {
          "name": "is a Site Director",
          "condition":
    "match.single(subject.attributes('https://acs.attributes.int',
'role'),
    'Site_Director')"
        },
        {
          "name": "is assigned to same customer",
          "condition":
    "match.single(subject.attributes('https://acs.attributes.int',
'customer'),
    resource.uriVariable('customer_id'))"
        }
      ],
      "effect": "PERMIT"
    }
```

- A policy that gives a production manager `HTTP GET` access to the customers correlated to a specific site. It contains the following conditions:
  - A condition that validates that the user has the `Production_Manager` role.
  - A condition that validates that the user is assigned to the same customer. The user can be assigned to multiple customers. However the condition in this sample validates that at least one customer matches the resource URL.

```
        {
      "name": "Production Managers can read a site if they have
access to Customer",
      "target": {
        "name": "When an Production Manager reads a site---this
policy def assumes
    the site_id is unique in the system.",
        "resource": {
          "name": "Site",
          "uriTemplate":
  "/customers/{customer_id:\\w*}/sites"
        },
        "action": "GET",
        "subject": {
          "name": "Production Manager Role",
          "attributes": [
```

```
            {
              "issuer": "https://acs.attributes.int",
              "name": "role"
            },
            {
              "issuer": "https://acs.attributes.int",
              "name": "customer"
            }
          ]
        }
      },
      "conditions": [
        {
          "name": "is a Production Manager",
          "condition":
    "match.single(subject.attributes('https://acs.attributes.int',
'role'),
    'Production_Manager')"
        },
        {
          "name": "is assigned to same customer",
          "condition":
    "match.single(subject.attributes('https://acs.attributes.int',
'customer'),
    resource.uriVariable('customer_id'))"
        }
      ],
      "effect": "PERMIT"
    }
```

- A policy that gives a production manager `HTTP GET` access to an assigned customer and site. It contains the following conditions:
  - A condition that validates that the user is assigned to a specific site.
  - A condition that validates that the user has the `Production_Manager` role.
  - A condition that validates that the user is assigned to the same customer. The user can be assigned to multiple customers. However, the condition in this sample validates that at least one customer matches the resource URL.

```
{
      "name": "Production Managers can read a site if they are
assigned to the site and Customer.",
      "target": {
        "name": "When an Production Manager reads a site---this
policy def assumes
    the site_id is unique in the system.",
        "resource": {
          "name": "Site",
          "uriTemplate":
    "/customers/{customer_id:\\w*}/sites/{site_id:\\w*}"
        },
        "action": "GET",
        "subject": {
          "name": "Production_Manager",
          "attributes": [
            {
              "issuer": "https://acs.attributes.int",
              "name": "site"
            },
            {
```

```
              "issuer": "https://acs.attributes.int",
              "name": "role"
            },
            {
              "issuer": "https://acs.attributes.int",
              "name": "customer"
            }
          ]
        }
      },
      "conditions": [
        {
          "name": "has permissions to do a GET on the
  site",
          "condition":
    "match.single(subject.attributes('https://acs.attributes.int',
'site'),
    resource.uriVariable('site_id'))"
        },
        {
          "name": "is a Production Manager",
          "condition":
    "match.single(subject.attributes('https://acs.attributes.int',
'role'),
    'Production_Manager')"
        },
        {
          "name": "is assigned to same Customer",
          "condition":
    "match.single(subject.attributes('https://acs.attributes.int',
'customer'),
    resource.uriVariable('customer_id'))"
        }
      ],
      "effect": "PERMIT"
    }
```

- A default DENY policy if none of the other policies apply.

```
{
            "name" : "Deny all other operations by
  default",
            "effect" : "DENY"
        }
```

The following sample access-control policy shows all the access-control policies assembled for the organization:

```
{
  "name": "sample-policy-set",
  "policies": [
    {
      "name": "Administrator can access all the customers.",
      "target": {
        "name": "When an Administrator accesses customers",
        "resource": {
          "name": "Customers",
          "uriTemplate": "/customers"
        },
        "subject": {
          "name": "Administrator role",
```

```
          "attributes": [
            {
              "issuer": "https://acs.attributes.int",
              "name": "role"
            }
          ]
        }
      },
      "conditions": [
        {
          "name": "is an Administrator",
          "condition":
    "match.single(subject.attributes('https://acs.attributes.int',
'role'),
    'Administrator')"
        }
      ],
      "effect": "PERMIT"
    },
    {
      "name": "Site Directors can read a sites/site if they are
assigned to the site and Customer.",
      "target": {
        "name": "When an Site Director reads his/her sites.",
        "resource": {
          "name": "Sites",
          "uriTemplate": "/sites"
        },
        "action": "GET",
        "subject": {
          "name": "Site Director role",
          "attributes": [
            {
              "issuer": "https://acs.attributes.int",
              "name": "role"
            }
          ]
        }
      },
      "conditions": [
        {
          "name": "is a Site Director",
          "condition":
    "match.single(subject.attributes('https://acs.attributes.int',
'role'),
    'Site_Director')"
        }
      ],
      "effect": "PERMIT"
    },
    {
      "name": "Site Directors can read a sites/site if they have
access to Customer",
      "target": {
        "name": "When an Site Director reads his/her sites.",
        "resource": {
          "name": "Customer Sites",
          "uriTemplate":
    "/customers/{customer_id:\\w*}/sites/{site_id}"
        },
        "action": "GET",
```

```
        "subject": {
          "name": "Site Director role",
          "attributes": [
            {
              "issuer": "https://acs.attributes.int",
              "name": "role"
            },
            {
              "issuer": "https://acs.attributes.int",
              "name": "customer"
            }
          ]
        }
      },
      "conditions": [
        {
          "name": "is a Site Director",
          "condition":
    "match.single(subject.attributes('https://acs.attributes.int',
'role'),
    'Site_Director')"
        },
        {
          "name": "is assigned to same customer",
          "condition":
    "match.single(subject.attributes('https://acs.attributes.int',
'customer'),
    resource.uriVariable('customer_id'))"
        }
      ],
      "effect": "PERMIT"
    },
        {
      "name": "Production Managers can read a site if they have access
to Customer",
      "target": {
        "name": "When an Production Manager reads a site---this policy
def assumes
    the site_id is unique in the system.",
        "resource": {
          "name": "Site",
          "uriTemplate":
  "/customers/{customer_id:\\w*}/sites"
        },
        "action": "GET",
        "subject": {
          "name": "Production Manager Role",
          "attributes": [
            {
              "issuer": "https://acs.attributes.int",
              "name": "role"
            },
            {
              "issuer": "https://acs.attributes.int",
              "name": "customer"
            }
          ]
        }
      },
      "conditions": [
        {
```

```
          "name": "is a Production Manager",
          "condition":
    "match.single(subject.attributes('https://acs.attributes.int',
'role'),
    'Production_Manager')"
        },
        {
          "name": "is assigned to same customer",
          "condition":
    "match.single(subject.attributes('https://acs.attributes.int',
'customer'),
    resource.uriVariable('customer_id'))"
        }
      ],
      "effect": "PERMIT"
    },
    {
      "name": "Production Managers can read a site if they are
assigned to the site and Customer.",
      "target": {
        "name": "When an Production Manager reads a site---this policy
def assumes
    the site_id is unique in the system.",
        "resource": {
          "name": "Site",
          "uriTemplate":
    "/customers/{customer_id:\\w*}/sites/{site_id:\\w*}"
        },
        "action": "GET",
        "subject": {
          "name": "Production_Manager",
          "attributes": [
            {
              "issuer": "https://acs.attributes.int",
              "name": "site"
            },
            {
              "issuer": "https://acs.attributes.int",
              "name": "role"
            },
            {
              "issuer": "https://acs.attributes.int",
              "name": "customer"
            }
          ]
        }
      },
      "conditions": [
        {
          "name": "has permissions to do a GET on the
  site",
          "condition":
    "match.single(subject.attributes('https://acs.attributes.int',
'site'),
    resource.uriVariable('site_id'))"
        },
        {
          "name": "is a Production Manager",
          "condition":
    "match.single(subject.attributes('https://acs.attributes.int',
'role'),
```

```
    'Production_Manager')"
        },
        {
          "name": "is assigned to same Customer",
          "condition":
    "match.single(subject.attributes('https://acs.attributes.int',
'customer'),
    resource.uriVariable('customer_id'))"
        }
      ],
      "effect": "PERMIT"
    },
        {
            "name" : "Deny all other operations by
  default",
            "effect" : "DENY"
        }
  ]
}
```

Attribute Management

In this example, several resources and subjects are defined to meet the access-control requirement.

In a REST client, enter the following REST API to create the resources:

```
HTTP POST
http://<host>/v1/resource
```

In the response body, enter the following sample JSON string to create the resources:

```
    [
         {
            "resourceIdentifier": "customers"
          },
          {
            "resourceIdentifier": "sites"
          },
          {
            "resourceIdentifier":
      "customers/customer1/sites/site1"
          },
          {
            "resourceIdentifier":
      "customers/customer1/sites/site2"
          },
          {
            "resourceIdentifier":
      "customers/customer2/sites/site1"
          }

        ]
```

In a REST client, enter the following REST API to create the subjects:

```
HTTP POST
http://<host>/v1/subject
```

In the response body, enter one of the following sample JSON strings to create different subjects:

- The following sample creates two users, `Acme Admin` and `Acme User`. While the `Acme Admin` user can access the `/customer` URL, the `Acme User` user cannot.

```
    [
      {
        "subjectIdentifier": "/subject/Acme Admin",
        "attributes": [
        {
          "name": "role",
          "value": "Administrator",
          "issuer": "https://acs.attributes.int"
        }
      ],
      },
{
        "subjectIdentifier": "/subject/Acme User",
        "attributes": [
        {
          "name": "role",
          "value": "User1",
          "issuer": "https://acs.attributes.int"
        }
      ],
      }
    ]
```

- The following sample creates two users, `Acme Site Director` and `Acme Site User`. While the `Acme Site Director` user can access the `/site` URL and access the `/customers` correlated to `/site`, the `Acme Site User` user cannot.

```
    [
      {
        "subjectIdentifier": "/subject/Acme Site Director",
        "attributes": [
        {
          "name": "role",
          "value": "Site_Director",
          "issuer": "https://acs.attributes.int"
        },
        {
          "name": "customer",
          "value": "customer1",
          "issuer": "https://acs.attributes.int"
        }
      ],
      },
{
        "subjectIdentifier": "/subject/Acme Site User",
        "attributes": [
        {
          "name": "role",
          "value": "User",
          "issuer": "https://acs.attributes.int"
        },
        {
          "name": "customer",
          "value": "customer1",
          "issuer": "https://acs.attributes.int"
        },
        {
```

```
            "name": "customer",
            "value": "customer2",
            "issuer": "https://acs.attributes.int"
          }
        ],
        }
    ]
```

- The following sample creates two users, `Acme Production Manager` and `Acme Production User`. While the `Acme Production Manager` user can access the `/site` URL and access the `/customers` correlated to `/site`, the `Acme Production User` user cannot.

```
    [
       {
        "subjectIdentifier": "/subject/Acme Production Manager",
        "attributes": [
        {
          "name": "role",
          "value": "Production_Manager",
          "issuer": "https://acs.attributes.int"
        },
        {
          "name": "site",
          "value": "site1",
          "issuer": "https://acs.attributes.int"
        },
        {
          "name": "customer",
          "value": "customer1",
          "issuer": "https://acs.attributes.int"
        }
      ],
      },
{
        "subjectIdentifier": "/subject/Acme Site User",
        "attributes": [
        {
          "name": "role",
          "value": "User",
          "issuer": "https://acs.attributes.int"
        },
        {
          "name": "customer",
          "value": "customer1",
          "issuer": "https://acs.attributes.int"
        },
        {
          "name": "customer",
          "value": "customer2",
          "issuer": "https://acs.attributes.int"
        },
        {
          "name": "site",
          "value": "site1",
          "issuer": "https://acs.attributes.int"
        },
        {
          "name": "site",
          "value": "site2",
          "issuer": "https://acs.attributes.int"
```

```
        }
      ],
      }
    ]
```

Policy Evaluation

The Policy Evaluation service performs policy evaluation based on the web service requests. The following samples show web service calls and their corresponding policy evaluation results based on the policies and attributes defined in this use case.

In a REST client, use the following REST API to evaluate an access-control policy:

```
HTTP POST
/v1/policy-evaluation
```

- The following sample call shows how an administrator can access a resource:

  ```
  {
    "resourceIdentifier": "/customers",
    "subjectIdentifier": "/subject/Acme Admin",
    "action": "GET"
  }
  ```

  The call returns the following policy evaluation result:

  ```
  {
  "effect": "PERMIT",
  "subjectAttributes": [
  {
  "issuer": "https://acs.attributes.int",
  "name": "role",
  "value": "Administrator"
  }
  ],
  "resourceAttributes": [ ]
  }
  ```

- The following sample call shows how a site director can access resources for a site:

  ```
  {
    "resourceIdentifier": "/sites",
    "subjectIdentifier": "/subject/Acme Site Director",
    "action": "GET"
  }
  ```

  The call returns the following policy evaluation result:

  ```
  {
  "effect": "PERMIT",
  "subjectAttributes": [
  {
  "issuer": "https://acs.attributes.int",
  "name": "customer",
  "value": "customer1"
  },
  {
  "issuer": "https://acs.attributes.int",
  "name": "role",
  ```

```
"value": "Site_Director"
}
],
"resourceAttributes": [ ]
}
```

- The following sample call shows how a site director can access a resource in a site:

```
{
  "resourceIdentifier": "/customers/customer1/sites/site1",
  "subjectIdentifier": "/subject/Acme Site Director",
  "action": "GET"
}
```

The call returns the following policy evaluation result:

```
{
"effect": "PERMIT",
"subjectAttributes": [
{
"issuer": "https://acs.attributes.int",
"name": "customer",
"value": "customer1"
},
{
"issuer": "https://acs.attributes.int",
"name": "role",
"value": "Site_Director"
}
],
"resourceAttributes": [ ]
}
```

- The following sample call shows how a production manager can access a resource on a specific site:

```
{
  "resourceIdentifier": "/customers/customer1/sites",
  "subjectIdentifier": "/subject/Acme Production Manager",
  "action": "GET"
}
```

The call returns the following policy evaluation result:

```
{
"effect": "PERMIT",
"subjectAttributes": [
{
"issuer": "https://acs.attributes.int",
"name": "site",
"value": "site1"
},
{
"issuer": "https://acs.attributes.int",
"name": "customer",
"value": "customer1"
},
{
"issuer": "https://acs.attributes.int",
"name": "role",
"value": "Production_Manager"
```

```
}
],
"resourceAttributes": [ ]
}
```

- The following sample call shows how a production manager can access a specific resource in a specific site:

```
{
  "resourceIdentifier": "/customers/customer1/sites/site1",
  "subjectIdentifier": "/subject/Acme Production Manager",
  "action": "GET"
}
```

The call returns the following policy evaluation result:

```
{
"effect": "PERMIT",
"subjectAttributes": [
{
"issuer": "https://acs.attributes.int",
"name": "site",
"value": "site1"
},
{
"issuer": "https://acs.attributes.int",
"name": "customer",
"value": "customer1"
},
{
"issuer": "https://acs.attributes.int",
"name": "role",
"value": "Production_Manager"
}
],
"resourceAttributes": [ ]
}
```

Example: Access Control Services Hierarchical Attribute Use Case

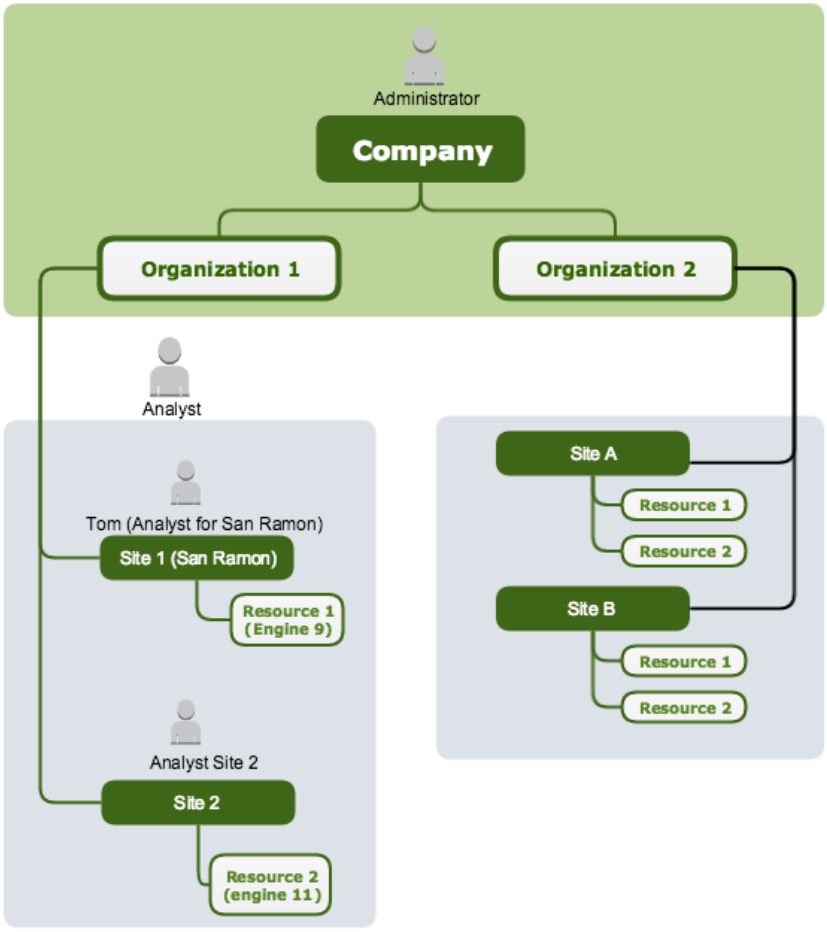
This use case shows the use of hierarchical subject and resource attributes in Access Control Services (ACS).

Note: To use the hierarchical subject and resource attributes, you must use the ACS instance with Beta plan. To subscribe to this plan, click on the Access Control Service tile in the predix.io catalog and select the Beta plan option.

A company has the following security requirements:

- Define an Analyst role for all sites in the organization.
- Define a sub-category of Analyst role as Data Scientist. A data scientist inherits attributes of Analyst role.
- Tom, who is a Data Scientist at Site 1, should be able to access all the resources at that site.
- Tom should not have access to resource at site 2.

The following diagram shows the company structure:



To implement this flow, an administrator can use the Policy Management service and Attribute Management service APIs to specify the required policies and hierarchical attributes and Policy Evaluation service to determine the access for web service calls.

Note: ACS service instances cache subject and resource attribute information in memory, as a performance optimization, up to a maximum of one second. This implies that there can be up to a 1 second delay between the time when a update to resource or subject is completed, and the time when the updated value is seen by a subsequent request (if the request goes to a different instance of ACS deployment).

The following sections demonstrate the use of ACS for:

- Hierarchical Attribute Management
  - Defining role and attributes
    This section shows use of Attribute Management Service APIs to create:
    - Analysts role with an attribute of Data Scientist
    - User (Tom) that inherits all attributes of Analyst
    - Resource site (San Ramon)
    - Resources (engine 9 and engine 11) at the resource sites
  - Policy Management
    This section shows use of Policy Management APIs to define a policy where data scientist can access data at the resource site.
  - Policy evaluation with hierarchical attributes

This section evaluates policy to permit data scientist (Tom) to access data at a site. In this case, Tom can access both site 1 and site 2.

- Hierarchical Scope Management
  To limit Tom's access to site 1 (San Ramon) only, an administrator can define scoped attributes.
  - Defining scoped attributes
    This section shows use of Attribute management Service API to define scopes that further qualify access a particular site.
  - Policy Evaluation with hierarchical and scoped attributes
    This section evaluates policy to deny access to resource at site 2 to Tom.

Before You Begin

Review the following resources to understand the concepts, tasks, and reference information needed to demonstrate the use of ACS services for this use case::

- ACS Deployment Architecture on page 1
- Access Control Services Architecture on page 2
- Creating Attributes for Resources and Subjects on page 19
- Creating Access-Control Policies on page 22
- Evaluating Access-Control Policies on page 34

Select a REST client (such as Postman) that can execute API requests. For more information about these APIs, see the API Documentation (Predix.io home page – Documentation – Service APIs).

Defining Roles and Attributes Using Attribute Management Service APIs

- Create an Analyst Role (`role-analyst`) with an attribute named `group` with a value of `Data Scientist`.
  In a REST client, enter the following REST API to create an Analyst role:

  ```
  HTTP PUT
  http://<host>/v1/subject/role-analyst
  ```

  In the response body, enter the following sample JSON string to create the role of an Analyst, `role-analyst`. The Analysts has an attribute of Data Scientist.

  ```
  {
    "subjectIdentifier" : "role-analyst",
    "attributes" : [
      {
        "issuer" : "https://acs.predix.io",
        "name"   : "role",
        "value"  : "analyst"
      },
    {
        "issuer" : "https://acs.predix.io",
        "name"   : "group",
        "value"  : "Data Scientist"
    }
    ]
  }
  ```

- Create a User, Tom, who inherits analyst role and all its attributes (group as Data Scientist).

In a REST client, enter the following REST API to create a user to inherit the role of an Analyst:

```
HTTP PUT
http://<host>/v1/subject/tom@acme.com
```

In the response body, enter the following sample JSON string to create the subject user:

```
{
  "subjectIdentifier" : "tom@acme.com",
  "parents" : [
    {
      "identifier" : "role-analyst"
    }
  ]
}
```

- Create a resource site, San Ramon.
  In a REST client, enter the following REST API:

  ```
  HTTP PUT
  http://<host>/v1/resource/%2fsites%2fsan-ramon
  ```

  In the response body, enter the following sample JSON string to create the resource site:

  ```
  {
    "resourceIdentifier" : "/sites/san-ramon",
    "attributes" : [
      {
        "issuer" : "https://acs.predix.io",
        "name"   : "site",
        "value"  : "san-ramon"
      }
    ]
  }
  ```

- Create a resource called `engine 9` at San Ramon site.
  In a REST client, enter the following REST API:

  ```
  HTTP PUT
  http://<host>/v1/resource/engines/9
  ```

  In the response body, enter the following sample JSON sample strong to create the resource:

  ```
  {
    "resourceIdentifier" : "/engines/9",
    "parents" : [
      {
        "identifier" : "/sites/san-ramon"
      }
    ]
  }
  ```

- Create a resource called `engine 11`. This resource resides outside of San Ramon site.
  In a REST client, enter the following REST API:

  ```
  HTTP PUT
  http://<host>/v1/resource/%2fengines%2f11
  ```

In the response body, enter the following sample JSON string to create the resource:

```
{
  "resourceIdentifier" : "/engines/11"
}
```

Policy Management

The administrator adds the policies to allow access to resources for analysts that are Data Scientists:

- In a REST client, enter the following REST API to create an access control policy for a zone:

```
HTTP PUT
/v1/policy-set/default
```

Note:

You must specify the `Predix-Zone-Id` header when you use the REST APIs.

- In the response body, enter the following sample JSON string to create the policy:

```
{
  "name" : "default",
  "policies" : [
    {
      "name" : "Analysts can access engines if they belong to the
same group.",
      "target" : {
        "resource" : {
          "name" : "Engine",
          "uriTemplate" : "/engines/{engine_id}"
        },
        "action" : "GET",
        "subject" : {
          "name" : "Data Scientists Group",
          "attributes" : [
            {
              "issuer" : "https://acs.predix.io",
              "name"   : "group",
              "value"  : "Data Scientist"
            }
          ]
        }
      },
      "conditions" : [
        {
          "name"      : "is an analyst",
          "condition" : "match.single(subject.attributes('https://
acs.predix.io', 'group'), 'Data Scientist')"
        }
      ],
      "effect" : "PERMIT"
    },
    {
      "name" : "Deny all other requests.",
      "effect" : "DENY"
    }
  ]
}
```

Policy Evaluation with Hierarchical Attributes

The Policy Evaluation service performs policy evaluation based on the web service requests. The following samples display web service calls and their corresponding policy evaluation results based on the policies and attributes defined in this use case.

In a REST client, use the following REST API to evaluate an access-control policy:

```
HTTP POST
/v1/policy-evaluation
```

In the response body, enter one of the following sample JSON strings to evaluate web service calls and their corresponding policy evaluation results based on the policies and attributes defined in this use case.:

- The following sample call shows how Tom can access a resource at any of the sites. Tom will be able to access resources at both Site 1 and Site 2.

  ```
  {
    "action" : "GET",
    "resourceIdentifier" : "/engines/9",
    "subjectIdentifier" : "tom@acme.com"
  }
  ```

  The call returns the following policy evaluation result:

  ```
  {
    "timestamp": 0,
    "resolvedResourceUris": [
      "\/engines\/9"
    ],
    "resourceAttributes": [
      {
        "value": "san-ramon",
        "name": "site",
        "issuer": "https:\/\/acs.predix.io"
      }
    ],
    "subjectAttributes": [
      {
        "value": "analyst",
        "name": "role",
        "issuer": "https:\/\/acs.predix.io"
      }
    ],
    "effect": "PERMIT"
  }
  ```

- The following sample call shows that Tom also has access to a resource at the other site:

  ```
  {
    "action" : "GET",
    "resourceIdentifier" : "/engines/11",
    "subjectIdentifier" : "tom@acme.com"
  }
  ```

  The call returns the following policy evaluation result:

  ```
  {
    "timestamp": 0,
    "resolvedResourceUris": [
  ```

```
    "\/engines\/11"
  ],
  "resourceAttributes": [],
  "subjectAttributes": [
    {
      "value": "analyst",
      "name": "role",
      "issuer": "https:\/\/acs.predix.io"
    }
  ],
  "effect": "PERMIT"
}
```

Defining Scoped Attributes

In this example, the attributes of a subject are defined to meet the access-control requirement.

- The user Tom inherits analyst role and all its attributes (group as Data Scientist). In addition, Tom has access to resources at the San Ramon Site only.
  In a REST client, use the following REST API to create a user to inherit the role of an Analyst and add scope to limit access to a site:

  ```
  HTTP PUT
  http://<host>/v1/subject/tom@acme.com
  ```

  In the response body, enter the following sample JSON string to create the subject user:

  ```
  {
    "subjectIdentifier" : "tom@acme.com",
    "parents" : [
      {
        "identifier" : "role-analyst",
        "scopes" : [
          {
            "issuer" : "https://acs.predix.io",
            "name"   : "site",
            "value"  : "san-ramon"
          }
      }
    ]
  }
  ```

Policy Evaluation with Hierarchical and Scoped Attributes

The following samples shows web service calls and their corresponding policy evaluation results based on the policies and attributes defined in this use case.

In a REST client, use the following REST API to evaluate an access-control policy:

```
HTTP POST
/v1/policy-evaluation
```

In the response body, enter one of the following sample JSON strings to evaluate web service calls and their corresponding policy evaluation results based on the policies and attributes defined in this use case.:

- The following sample call shows that Tom who is an analyst at site 1 (San Ramon) cannot access a resource (resource 11) at other site:

```
{
  "action" : "GET",
  "resourceIdentifier" : "/engines/11",
  "subjectIdentifier" : "tom@acme.com"
}
```

  The call returns the following policy evaluation result:

```
{
  "timestamp": 0,
  "resolvedResourceUris": [
    "\/engines\/11"
  ],
  "resourceAttributes": [],
  "subjectAttributes": [],
  "effect": "DENY"
}
```

Example: ACS Attribute Connector Setup

Configure access to subject and resource attributes from external resources for policy evaluation.

Purpose

As a developer, you can extend Access Control Services to access remote attributes from external sources for policy evaluation:

- Allow ACS policies to reference attributes of a protected resource from an external source (such as Predix Asset service).
- Allow ACS policies to reference attributes of a currently authenticated user from an external source (such as federated identity management).

Before You Begin

As a developer, you should review the following resources to understand the conceptual, task, and reference information needed to extend Access Control Services to support external attribute resources:

- ACS Deployment Architecture on page 1
- Access Control Services Architecture on page 2
- ACS Attribute Connector Architecture on page 3
- Creating Attributes for Resources and Subjects on page 19
- Evaluating Access-Control Policies on page 34

Task Roadmap

The following tasks show how to extend Access Control Services to support external attribute sources.

#	Task	Information
1	Create a custom ACS adapter to manage requests for remote attributes from a specific external resource.	See Creating An ACS Attribute Adapter on page 59.
2	Create a common ACS connector to manage requests for remote attributes for policy evaluation.	See Creating an ACS Attribute Connector on page 60.

Creating An ACS Attribute Adapter

Create a custom ACS adapter to manage requests for remote attributes from a specific external resource.

Before You begin

Select a REST client (such as Postman) to create a REST API that can interface with the Attribute Connector Management API to retrieve remote attributes from an external resource. For more information about Attribute Connector Management API, see the API Documentation (Predix.io home page – Documentation – Service APIs).

Note:

You must specify the `Predix-Zone-Id` header when you use this REST API to communicate with a deployed application about its environment.. For more information, see Binding an Application to the ACS Instance on page 16.

About This Task

As a developer, you need to create a REST API that follows a common scheme to retrieve remote attributes from an external resource for evaluation in access-policy conditions. For example:

```
GET {adapterEndpoint}/v1/attribute?id={id}
```

where

- `{adapterEndpoint}` is the adapter URL to retrieve remote attributes from an external resource.
- `{id}` is the identifier to retrieve remote attributes .

Note:

An error will occur if the REST API does not match the `/v1/attribute?id={id}` syntax.

The response to the GET call would return the value of the remote attributes associated with `{id}`:. For example:

```
{
 "attributes": [
 {
 "issuer": "string"
 "name": "string",
 "value": "string"
 }
 ],
 "id": "string"
 }
```

Creating an ACS Attribute Connector

Create a common ACS connector to manage requests for external attributes for policy evaluation.

Before You begin

Select a REST client (such as Postman) that can execute API requests for the Attribute Connector Management service. For more information about this API, see the API Documentation (Predix.io home page – Documentation – Service APIs).

Note:

You must specify the `Predix-Zone-Id` header when you use this REST API to communicate with a deployed application about its environment.. For more information, see Binding an Application to the ACS Instance on page 16.

About This Task

The syntax to configure an ACS attribute connector to manage requests for external attributes from an external source is listed:

```
{
  "adapters": [
   {
    "adapterEndpoint":"string",
    "uaaTokenUrl":"string",
    "uaaClientId":"string",
    "uaaClientSecret":"string"
    }
    ]
    "isActive":"boolean",
    "maxCachedIntervalMinutes":"number",
}
```

The ACS attribute connector can have the following values:

- `adapterEndpoint`: ACS attribute adapter URL to retrieve attribute data from the external source, If an adapter is configured and is active, it will be used as the source for attributes instead of the built-in attribute store managed by the Attribute Management Service (see Access Control Services Architecture on page 2).
- `uaaTokenUrl` UAA URL used to request an access token for the adapter.
- `uaaClientId`: OAuth2 client used to obtain an access token for the adapter.
- `uaaClientSecret`: OAuth client secret used to obtain an access token for the adapter ,
- `maxCachedIntervalMinutes` : Maximum time (in minutes) before external attributes are refreshed.
- `isActive` : A flag that either enables or disables the retrieval of external attributes.

Here a sample request body for the `PUT /v1/connector/resource` API that can be used as a template

```
{
    "adapters": [
        {
            "adapterEndpoint": "https://
<adapter.fully.qualified.domain>/v1/attribute",
            "uaaClientId": "<connector_client_id>",
            "uaaClientSecret": "<connector_client_secret>",
```

```
            "uaaTokenUrl": "https://<uaa-instance-guid>.<uaa-service-
id>.run.aws-usw02-pr.ice.predix.io/oauth/token"
        }
    ],
    "isActive": true,
    "maxCachedIntervalMinutes": 480
}
```

Integrating Access Control Service With Your Application

Access Control Services Spring Security Extensions

Predix platform provides Spring Security Extensions that can be used for integrating with spring security, to implement fine-grained access control of application resources, using the Access Control Service (ACS).

Note:

If you do not use Java using spring framework, you can use the ACS REST APIs directly.

Securing Resources With ACS Using Spring Security Extensions

About This Task

To enforce authentication and authorization, you can restrict access to the RESTful endpoints exposed with Spring MVC.

Procedure

Include the `isAcsAuthorized()` expression in the `access` attribute on `http` element for resources that need to be protected with ACS.

For example,

- Set `use-expressions` attribute to `true`.
- Configure the ACS Expression handler `<expression-handler ref="acsExpressionHandler"/>`.
- Use `isAcsAuthorized()` expression to add authorization with ACS. If your ACS instance has more than one policy set configured, you must provide an ordered list of the policy set names to be used for evaluation. You can specify the policy set names as method parameters. If there is just one policy set, this parameter is optional.

```
<http pattern="/**"
     request-matcher="ant" xmlns="http://www.springframework.org/
schema/security"
 disable-url-rewriting="true"
 use-expressions="true"
 entry-point-ref="preAuthenticationEntryPoint" >
 <expression-handler ref="acsExpressionHandler" />

 <!-- /greeting not authorized with ACS -->
 <intercept-url pattern="/greeting"
    access="isFullyAuthenticated()"/>

 <!-- /sites/** authorized with ACS based on 'policy-set-2' -->
 <intercept-url pattern="/sites/**"
    access="isAcsAuthorized( 'policy-set-2') and
isFullyAuthenticated()" />

 <!-- /resources/** authorized with ACS based on provided ordered list
of available policy sets -->
 <intercept-url pattern="/resources/**"
```

```
    access="isAcsAuthorized('policy-set-1', 'policy-set-2') and
isFullyAuthenticated()" />
 <anonymous enabled="false" />
 <custom-filter ref="oauth2ClientFilter" before="PRE_AUTH_FILTER" />
 <custom-filter ref="oauth2ServiceFilter" position="PRE_AUTH_FILTER" />
</http>
```

Getting Started With the Spring Security Extensions

About This Task

Add the ACS spring security extensions dependency and import the spring configuration bean.

Procedure

1. Set up access to Predix platform Artifactory.
2. Add the `acs-spring-security-extensions` library dependency to your application POM file.

```
<dependency>
    <groupId>com.ge.predix</groupId>
    <artifactId>acs-spring-security-extensions</artifactId>
    <version>5.1.0</version>
</dependency>
```

3. Add the following import in your application configuration beans xml file:

```
<import resource="classpath:acs-spring-config.xml" />
```

Configuring ACS Details

ACS spring security extensions require ACS instance details to delegate resource authorization to ACS.

You can use the ACS spring security extensions in one of the following two ways:

- Use the default implementation, `com.ge.predix.acs.spring.security.config.AcsClientConfigurationProvider`. The default implementation relies on the UAA and ACS services information to be available in your application property file.
- Override the default implementation by creating your own implementation of the `com.ge.predix.acs.spring.security.config.AcsClientConfigurationProvider` interface. You can use this option if your application requires multiple OAuth2 clients accessing ACS.

Configure Java Properties to Use Default Implementation

Update your application properties file (`application.properties`) to include the UAA and ACS service instance details.

Update your application properties file for the following values:

Variable	Description
`acsServiceInstanceName`	Specify the name of your ACS instance. If you specify the name of the ACS instance, the ACS spring security extension uses your application's

Variable	Description
	VCAP_SERVICES environment variable to retrieve the values for `acsZone` and `acsPolicyEvaluationTokenScope`. If the details are specified both in VCAP_SRVICES and in the properties file, the values in VCAP_SERVICES take precedence. • `acsZone` Specifies the name of the ACS instance zone. It is the `http-header-name` value generated in the VCAP_SERVICES environment variable. For example, `Predix-Zone-Id`. • `acsPolicyEvaluationTokenScope` Specifies the ACS zone scope. It is the `oauth-scope` value generated in the VCAP_SERVICES environment variable. For example, `predix-acs.zones.9378e3db-e683-46a2-97c2-ccd11d75869d.user`.
`accessTokenEndpointUrl`	Specify the UAA instance issuer identifier. For example, `https://ff27c315-d027-4d1d-a30c-64f49b369ed9.predix-uaa.run.aws-usw02-pr.ice.predix.io/oauth/token`.
`clientId`	Specify the client identifier for your UAA instance.
`clientSecret`	Specify the client secret for your UAA instance.

For example, the following properties file shows the updated values:

```
accessTokenEndpointUrl=https://ff27c315-d027-4d1d-
a30c-64f49b369ed9.predix-uaa.run.aws-usw02-pr.ice.predix.io/oauth/
token

clientId=<client_id>
clientSecret=<client_secret>

# The following properties are used by acs-spring-security-extensions
to invoke ACS for policy evaluation.
# acsServiceInstanceName takes precedence over acsUri, when that
instance is available in the application's
# VCAP_SERVICES
#Specify ACS endpoint (http://host:port) to use when cloud foundry
VCAP services are not available.
acsUri=${ACS_URL}

#Specify ACS instance name to bind. This is used by acs-spring-
security-extension.
acsServiceInstanceName=

acsZone=9378e3db-e683-46a2-97c2-ccd11d75869d

acsPolicyEvaluationTokenScope=predix-acs.zones.9378e3db-e683-46a2-97c2-
ccd11d75869d.user
```

Configure Custom Provider

If you need to override the default implementation, you can create your own implementation as follows:

- Implement the `com.ge.predix.acs.spring.security.config.AcsClientConfigurationProvider` interface.
- Annotate the implementation with `@Primary` and `@Component`. This ensures that your implementation is used while calling ACS.

Providing Additional Subject Attributes for Policy Evaluation

By default, ACS uses the subject attributes that you specify for policy evaluation. Some applications require to specify additional subject attributes during runtime. ACS spring security extensions provide a way for your application to provide additional subject attributes that you can use for the policy evaluation of the current request.

To provide additional attributes:

- Implement the `com.ge.predix.acs.spring.security.extensions.policy.evaluation.AcsPolicyEvaluationRequestCustomizer` interface.
- Annotate the implementation with `@Primary` and `@Component` to ensure it is used when calling ACS.

Access Control Services Release Notes

Access Control Services Release Notes

Q1 2017 Release

The following new feature was added.

Added Connector APIs to Support External Subject and Resource Attributes

ACS now provides APIs to support configuration of subject and resource connectors.

Added Support for Audit Logging

ACS now support for audit logging in Predix US West Domain.

Q4 2016 Release

New Features

The following new feature was added.

Hierarchical Attribute and Scopes

You can now define hierarchical attributes for entities such as roles and groups. The users can inherit attributes from the ones that you define.

For an example of using hierarchical attributes, see Access Control Services Hierarchical Attribute Use Case.

Support for Multiple Policy Sets

You can now use multiple policy sets with the Access Control services (ACS). The ACS spring security extensions were updated to include support for multiple policy sets.

For more information, see Securing Resources With ACS Using Spring Security Extensions on page 62.