Predix Quick reference

version 0.1

# Important Note

# This document is not a substitute of Predix IO site. This is for quick reference & rehearsing topics very quickly. You may find similar wordings and sentence from predix IO, but all important keywords are highlighted, organized by there relevancy. This document will be continuously enhanced as I go through all topic.

# Some place there are reference from external site to explain the topic in easy and better way.

# Predix API\_Endpoint

* Predix Basic

https://api.system.aws-usw02-pr.ice.predix.io

* Predix Select

https://api.system.asv-pr.ice.predix.io

* Predix Japan

https://api.system.aws-jp01-pr.ice.predix.io

* Predix UK

<https://api.system.dc-uk01-pr.ice.predix.io>

# Understanding Platform Service

1. Predix platform services use OAuth2 for authentication
2. services leverage the [User Account and Authentication (UAA)](https://www.predix.io/docs/?r=16722#SJzDWEsh) web service

# UAA

1. User Account and Authentication (UAA) is a web service provided by Cloud Foundry to manage users and OAuth2 clients.
2. OAuth2 provider
3. Issue token to application when application act on behalf of users
4. Authenticate users if there is a login server
5. provides endpoints for managing user accounts and for registering OAuth2 clients
6. Step 1 – have the instance of the UAA service as trusted issuer (***Tip: The maximum number of UAA instances that you can create in your space is 10. As a best practice, use the same UAA instance for your services***) ***An admin client is created for bootstrap purpose. You can create additional clients to use with your application.***
   1. cf cs predix-uaa tiered test-1 -c '{"adminClientSecret":"admin","subdomain":"ge-digital"}'
   2. the link will have <https://ge-digital.sb-uaa.grc-apps.svc.ice.ge.com/oauth/token>
7. Create a service instance with that issuer ID
   1. cf create-service <service\_name> <plan> <my\_instance> -c '{"trustedIssuerIds":[“**test-1**”, "<uaa\_instance2\_issuerID>"]}'
   2. you can have multiple issuer id , <uaa\_instance\_issuerID> is the issuerID of your trusted issuer (UAA instance)
8. The OAuth 2.0 protocol establishes a **client** as an application that **makes requests** for a **protected** **resource** on **behalf of** the **resource owner** and **with its authorization**
9. an admin client is automatically when create a UAA instance
10. can create additional clients for different applications
11. Create Client – 4 scopes
    1. 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.
    2. 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.
    3. 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**.
    4. 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. a refresh\_token is also assigned by default
12. Scopes are permissions associated with an OAuth Client for a user accessing a resource through an application.. Scopes are used for grant types authorization\_code, password and implicit.
13. 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 and there is no user involvement.  Used for grant type of client\_credentials
14. admin client is not assigned the authority to be able to change the user password by default. If you need the ability to update or change the user password, you must add the uaa.admin authority to your admin client.
15. UAA Dashboard to create additional clients, by default the client is created forclient\_credentials grant type
16. Using UAAC uaac target <https://11fa0273-9e2a-37e2-9d06-2c95a1f4f5ea.predix-uaa.run.aws-usw02-pr.ice.predix.io>
    1. uaac target <uaa\_instance\_url>
    2. uaac token client get admin
    3. create the OAuth2 client

uaac client add [**client\_name**]

--authorities "uaa.resource"

--scope "openid"

--autoapprove "openid"

--authorized\_grant\_types [authorization\_code|implicit|password|client\_credentials|refresh\_token]

--redirect\_uri [redirect\_uri\_1, redirect\_uri\_2, ...]

* 1. uaac client update admin --authorities <set\_of\_authorities> , <set\_of\_authorities> is your existing set of authorities plus authorities required for a platform service(s).
  2. uaac client get admin //will return the existing set of authorities
  3. uaac token client get admin //Get the token again with client credential grant.
  4. uaac token decode

1. changing the admin secret
   1. *after login issue the command* uaac secret change
2. After binding the service to UAA instance
   1. uaa\_instance\_issuerId = https://ff27c315-d027-4d1d-a30c-64f49b369ed9.predix-uaa.run.aws-usw02-pr.ice.predix.io/oauth/token
   2. uaa\_instance\_uri = https://ff27c315-d027-4d1d-a30c-64f49b369ed9.predix-uaa.run.aws-usw02-pr.ice.predix.io
   3. uaa\_instance\_GUID = ff27c315-d027-4d1d-a30c-64f49b369ed9
3. UAA includes feature like
   1. Identity management using SCIM – simple cloud identity management
   2. oAuth 2.0 authorization server
   3. login and logout
   4. saml federation capabilities
4. Creating OAuth2.0 client
   1. Client id & client id
   2. Authorization grant type
      1. client\_credentials : UAA accepts the client id and client secret and provides Access Tokens.
      2. Implicit : UAA directly issues an Access Token to the client without authenticating the client.
      3. Password : UAA accepts the username and password and provides Access Tokens
      4. authorization\_code : client directs the resource owner to UAA, which in turn directs the resource owner back to the client with the authorization code, refresh\_token is also assigned by default.
   3. Scopes
      1. Scopes are permissions associated with an OAuth Client
      2. determine if an application is allowed to access on behalf of the user
   4. 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 and there is no user involvement.
   5. Redirect URI : Specify a redirect URI to redirect client after login
5. Creating Group : group represents the privileges a user can potentially have. When a user is redirected to UAA for authentication, UAA adds the group information in the access token
6. Creating Users in UAA :  for authorization grant type such as Authorization Code, Implicit, and Resource Owner Password involves users that provide their username and password credentials to an application. The application uses UAA to authenticate the users using these credentials. For UAA to authenticate the users, the users must first be created in UAA.

***The admin client is not assigned the authority to be able to change the user password by default. If you need the ability to update or change the user password, you must add theuaa.admin authority to your admin client.***

* 1. uaac user add <my-user> --emails <my\_user>-user@ge.com --password <my\_password>

// create Group

uaac group add scim.read

uaac group add scim.write

// Add user to the group

uaac member add zones.<my\_uaa\_instance>.admin <my-user>

uaac member add scim.read <my-user>

uaac member add scim.write <my-user>

uaac member add clients.write <my-user>

uaac member add clients.read <my-user>

uaac member add clients.admin <my-user>

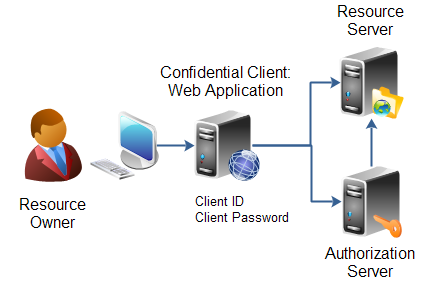
uaac member add clients.secret <my-user>

// Verify user is created with right group

uaac token owner get <my-oauth-client> <my-user>

uaac token decode

1. Using UAA to Obtain OAuth2 Access Tokens
   1. You can obtain access tokens for the following authorization grant types:
      1. Authorization Code
      2. Client Credentials
      3. Implicit
      4. Resource Owner Password Credentials
      5. JSON Web Token (JWT) Bearer Toke
2. **Obtaining Tokens Using Authorization Code Grant :** 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. The OAuth2 endpoint in UAA accepts authorization code to provide an Access Token.  the token is stored and managed on the server’s side. technical information present in the token (such as scope names), is not leaked to the user and the token is less prone to be misused. Additionally, with Authorization Code grant type, it is easier to terminate a user’s session without waiting for the token to expire its lifetime



The authorization grant is given to a client application by the resource owner, in cooperation with the authorization server associated with the resource server

=>The resource owner (user) accesses the client application. The client application tells the user to login to the client application via an authorization server (e.g. Facebook, Twitter, Google etc.)

=>To login via the authorizaion server, the user is redirected to the authorization server by the client application. The client application sends its client ID along to the authorization server, so the authorization server knows which application is trying to access the protected resources. **The request includes client identifier, requested scope, local state, and a redirection URI.**

=>The user logs in via the authorization server. After successful login the user is asked if she wants to grant access to her resources to the client application**. UAA authenticates the resource owner (via the user-agent) and establishes whether the resource owner grants or denies the client's access request**. If the user accepts, the user is redirected back to the client application. **UAA sends the authorization code back to the client using the redirection URI provided in previous step**

=>When redirected back to the client application, the authorization server sends the user to a specific redirect URI, which the client application has registered with the authorization server ahead of time. Along with the redirection, the authorization server sends an authorization code, representing the authorization.

=>When the redirect URI in the client application is accessed, the client application connects directly to the authorization server. The client application sends the authorization code along with its own client ID and and client secret. **The client requests an access token from the UAA token endpoint by including the authorization code received in the previous step.**

=>If the authorization server can accept these values, the authorization server sends back an access token. **UAA authenticates the client, validates the authorization code, and ensures that the redirection URI received matches the URI used to redirect the client. If the client is authenticated, UAA sends an access token back**.

=> The client application can now use the access token to request resources from the resource server. The access token serves as both authentication of the client, resource owner (user) and authorization to access the resources.

1. **Obtaining Tokens Using Implicit Grant**

=>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. However with this grant type, the access token is transmitted in the URI fragment, which can expose it to unauthorized parties. Additionally if the public clients use implicit flows, UAA cannot determine what client an access token was issued to.

=>improve the responsiveness and efficiency.

=>An implicit authorization grant is similar to an authorization code grant, except the access token is returned to the client application already after the user has finished the authorization. The access token is thus returned when the user agent is redirected to the redirect URI.

=>Implicit authorization grant is mostly used in a user agent or native client application.

1. **Using Resource Owner Password Credentials Grant**

=>The resource owner password credentials authorization grant method works by giving the client application access to the resource owners credentials. For instance, a user could type his Twitter user name and password (credentials) into the client application. The client application could then use the user name and password to access resources in Twitter.

=>OAuth2 endpoint in UAA accepts the username and password and provides Access Tokens.

=>You must use this grant type only when there is a high degree of trust between the resource owner and the client

=> When you use this grant type, the client does not need to store the resource owner credentials for future use. The client simply exchanges the credentials with a long-lived access token or refresh token.

1. **Using Client Credentials Grant**

=> Client credential authorization is for the situations where the client application needs to access resources or call functions in the resource server, which are not related to a specific resource owner . =>client credentials grant type, the OAuth2 endpoint in UAA accepts the client id and client secret and provides Access Tokens. The client credentials can be used when the authorization scope is limited to the protected resources under the control of the client, or to protected resources previously authorized with UAA

1. **Using JWT Bearer Token Grant**

UAA Support uses of JSON Web Token (JWT)  for requesting an OAuth 2.0 access token as well as for client authentication

Support RSA SHA256 algorithm

* 1. Application developers obtains an X509 certificate and registers it with their organization's certificate management
  2. application developers create a JWT token in compliance with the fields described in RFC 7523, and sign it with the private key of the certificate
  3. Client makes a request to UAA using the JWT Bearer authorization grant
  4. UAA validates the JWT token from device as described
  5. If the JWT is valid, UAA issues an Access Token

To use the JWT token authorization grant flow, the client/device creates a JWT token and makes a request to UAA token endpoint with following parameters

uaac curl **/oauth/token** -X POST \-H "content-type:application/x-www-form-urlencoded;

charset=utf-8" -H "accept:application/json;charset=utf-8" \

--data 'grant\_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Ajwt-bearer&assertion=<asdfjhasdfhasdjfkh

---truncatedfor brevity>

//grant\_type=urn:ietf:params:oauth:grant-type:jwt-bearer

//assertion==<jwt-token>

1. **Using UAA to Obtain ID Tokens**

=>OpenID Connect 1.0 is a identity layer on top of the OAuth 2.0 protocol. It enables the clients to verify the identity of the end-user based on the authentication performed by an authorization server, and obtain basic profile information about the end-user.

=>To use the OpenID Connect implementation, a client includes the **openid scope** value in the authorization request

=>The OAuth 2.0 authorization server returns the information about the authentication performed in a JSON Web Token (JWT) called **an ID Token**.

=>The ID Token contains claims about the **authentication of an end-user** by the authorization server, and can optionally contain other requested claims.

=>Valid for Authorization Code (response\_type=code), Implicit (response\_type=id\_token token or response\_type=id\_token) & Hybrid

1. **Using UAA for Token Validation**

After generating an Access Token, the client accessing your application **presents** the **bearer token** to **your application** for authentication. Your application validates this token using UAA.

* Validate the token remotely using UAA
* Validate the token locally using FastTokenService
* Local Validation can check Is the token tampered?, Is the token expired?, Is the token issued for a future date?, Is the token issued by UAA?, Is the issuing application still available?
* Was the token revoked?, Is the user still available and active? Can be checked Remotely

1. **Setting up Fast Token Validation**

**//POM**

<dependency>

    <groupId>com.ge.predix</groupId>

    <artifactId>uaa-token-lib</artifactId>

    <version>3.1.1</version>

    <type>pom</type>

</dependency>

**// Update Spring**

<bean id="fasttokenServices"

class="com.ge.predix.uaa.token.lib.FastTokenServices">

<property name="storeClaims" value="true" />

<property name="trustedIssuers">

<list>

<value>${issuerId}</value>

</list>

</property>

</bean>

// Update your Spring configuration to include reference to fastTokenService

<oauth:resource-server id="oauth2ServiceFilter"

token-services-ref="fasttokenServices" />

//In Code

ZoneOAuth2Authentication zoneAuth = (ZoneOAuth2Authentication)

SecurityContextHolder.getContext().getAuthentication();

String zoneId = zoneAuth.getZoneId();

# Access Control Services

* The **Policy** Management service provides CRUD operations for application policies.

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

* The **Attribute** Management service provides CRUD operations for user and resource attributes.

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.

* The **Policy Evaluation** service processes policy evaluation (such as access control) requests for an OAuth client.

The Policy Evaluation service is an internal ACS service that **evaluates** **policies** based on **web** **service** **requests** for authorization. A web service request is sent to the policy evaluation service, which then validates the request against a policy and defined attributes and returns a decision of permit or deny for the request.

The request from a web service consists of the following components:

* Resource information: Relative URI path of the request.
* Subject information: The Policy Evaluation service extracts this information from the OAuth2 token used to authenticate with the web service.
* Action information: HTTP method used for the request.

# Attributes of manifest.yml

TODO

# Common Commands

Cf push

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

# cf apps

cf create-service <service\_name> <plan> <my\_instance

// cf cs

cf marketplace

cf bind-service <your\_app\_name> <service\_instance\_name>

cf env <your\_app\_name>

cf unbind-service <your\_app\_name> <uaa\_instance\_name>

cf delete-service predix-uaa <uaa\_instance\_name>

# Action item

## Put a note on how to do the

## Add a Secure Shell (SSH) key to your GitHub account. For more information, see<https://help.github.com/articles/generating-an-ssh-key/>.

# Keywords

SCIM – system of cross domain identity management