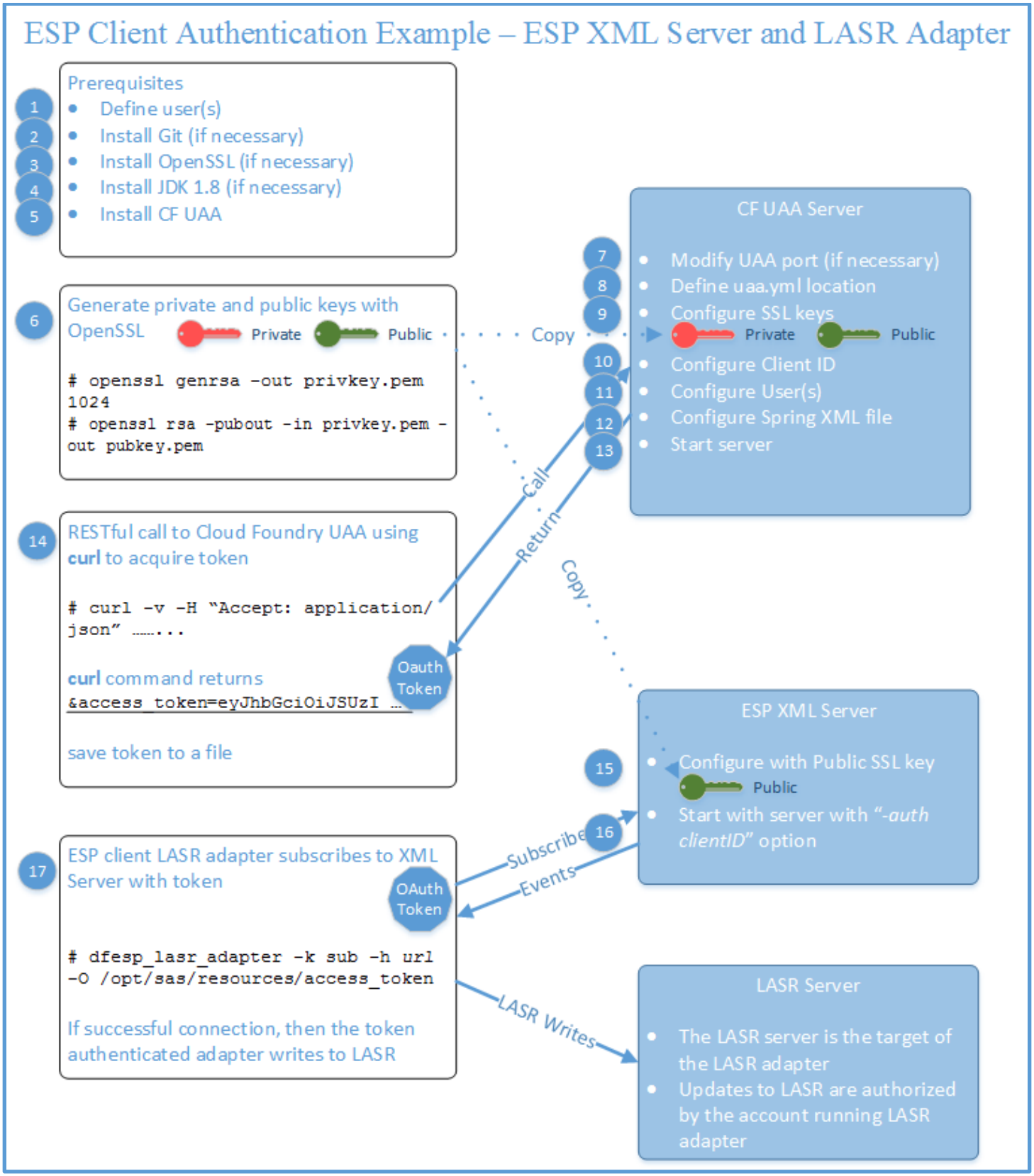
Enabling SAS Event Stream Processing Authentication

One of the new features of SAS Event Stream Processing 3.2 was the introduction of optional authentication between clients and servers. Authentication was added to the product's network interfaces such as the XML server API and the Java/C Pub/Sub APIs and adapters. It was also added to the SAS ESP Streamviewer, a tool that allows the user to display events streaming through a model. However, authentication was not added to SAS Event Stream Processing Studio, as that tool is intended for development and testing new models before they are deployed to production.

Enabling ESP authentication

The high-level steps for enabling authentication are available in Chapter 26 of the SAS Event Stream Processing 3.2 User's Guide, but the details required to implement these steps may not be so obvious. The following diagram provides an overview of the example used in the document by showing the steps required for a user to authenticate a LASR adapter to an ESP XML server. In this document the details required for each step of the example are revealed. The diagram may not make sense at this point, but hopefully by the time you reach the end of the document the pieces will come together.



Cloud Foundry UAA

Authentication between a client and server in SAS Event Stream Processing requires a token obtained from a supported OAuth2 Authorization server. The only supported OAuth2 server at present is Cloud Foundry (CF) User Account and Authentication (UAA). These tokens are based on the JSON Web Token and are digitally signed by UAA. UAA is an identity management service from Cloud Foundry, but it does not require a full deployment of Cloud Foundry's Platform

as a Service (PaaS) to access its services. As a result CF UAA may be deployed as a standalone service. Cloud Foundry services are Open Source Software and therefore are readily available for download and deployment.

Generating and making use of the token

As noted earlier, the SAS Event Stream Processing 3.2 User's Guide identifies the major steps required to generate a token. However, these steps are fairly broadly defined to meet the needs of a diverse customer base. This section takes those steps and provides more concrete actions. The following steps were taken on a Red Hat machine that is host to SAS Visual Analytics 7.3 and SAS Event Stream Processing 3.2. Commands are highlighted in grey and key parameters and values are highlighted in blue.

1. Define users for authentication

It is first necessary to define those accounts that will be used to authenticate ESP clients to ESP servers. Enabling authentication on an ESP server is a global and permanent setting. Therefore all clients connecting via TCP or pub/sub APIs to that server must use the authentication token. In order to generate a token a user account must be defined.

In most deployments there will be only a few accounts defined for authentication. Since ESP engines are sensitive to latency, it is important to minimize the impact on running models. This translates to maintaining strict control of the clients used to interact with the ESP server and engine. For this step I created one account to be defined within UAA. This account and the associated authentication token can be used by multiple clients. Separate accounts may be required in a situation where various types or classifications of client access are required. Currently this is beyond the scope of this document.

The account user name created for testing (**markt**) is simply made up of letters in my name, although it could be a name that represents the role (e.g. **espclient**). This account name is then specified as part of CF UAA configuration in a subsequent step.

2. Install GIT (if not installed) - required for CF UAA download

Git is a widely-used source code management system for software development. It is a distributed revision control system with an emphasis on speed, data integrity, and support for distributed, non-linear workflows. The source for CF UAA is stored in an open source Git repository and therefore Git is required to download the software. Install Git on the same machine you expect to install CF UAA.

First check to see if it is already installed. If not, use **yum** to install it using the root account or sudo access. **Note:** Not all messages of **yum** installation are shown below.

```
[root@sasserver01 java]# rpm -q git
package git is not installed
[root@sasserver01 java]# yum install git
Loaded plugins: refresh-packagekit, rhnplugin
This system is receiving updates from RHN Classic or RHN Satellite.
Setting up Install Process
Resolving Dependencies
.
. .
. . . . . .
Install       3 Package(s)

Total download size: 4.7 M
Installed size: 0
Is this ok [y/N]: y
Downloading Packages:
(1/3): git-1.7.1-3.el6_4.1.x86_64.rpm
| 4.6 MB     00:00

.
. .
```

```
. . . . . .

Installed:
  git.x86_64 0:1.7.1-3.el6_4.1

Dependency Installed:
  perl-Error.noarch 1:0.17015-4.el6
perl-Git.noarch 0:1.7.1-3.el6_4.1

Complete!
[root@sasserver01 ]$ rpm -q git
git-1.7.1-3.el6_4.1.x86_64
```

3. Install OpenSSL (if not installed) - required for generating keys

OpenSSL is required to generate private/public key pair for verifying and signing the token. In most cases the OpenSSL package will be installed. It can be queried with the following command.

```
[root@sasserver01 java]# rpm -q openssl
openssl-1.0.1e-16.el6_5.15.x86_64
```

If it doesn't exist, it can be installed with **yum**.

```
[root@sasserver01 java]# yum install openssl
```

If **SAS Event Stream Processing Authentication and Encryption** was selected and installed along with the ESP engine and ESP studio, then **openssl** will be installed in the **/bin** directory of the ESP engine (i.e. **$SASHOME/SASEventStreamProcessingEngine/3.2.0/bin**).

4. Install JDK 1.8 (if not installed) - required by the CF UAA web server

CF UAA requires JDK 1.8. CF UAA will fail to build the web server using versions prior to 1.8. Therefore if it doesn't exist, it will be necessary to download it and install it. There are a few different ways to install it. In this example the gzip tar file was downloaded to **/usr/java** and uncompressed.

http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
Download jdk-8u65-linux-x64.tar.gz for Linux 64 bit and move **/usr/java**

```
[root@sasserver01 java]# cd /usr/java
[root@sasserver01 java]# tar -xzvf jdk-8u65-linux-x64.tar.gz
[root@sasserver01 java]# ll
total 12
lrwxrwxrwx. 1 root root   16 Mar  5  2012 default -> /usr/java/latest
drwxr-xr-x. 9 root root 4096 Mar  5  2012 jdk1.6.0_21
drwxr-xr-x. 8 root root 4096 Dec 18  2013 jdk1.7.0_51
drwxr-xr-x. 8 uucp  143 4096 Oct  6 20:29 jdk1.8.0_65
lrwxrwxrwx. 1 root root   11 Mar 11  2014 latest -> jdk1.7.0_51
[root@sasserver01 java]# export JAVA_HOME=/usr/java/jdk1.8.0_65
```

5. Download and install CF UAA using GIT

Using the Git package that was downloaded in step 2, download (clone) the CF UAA software to a directory of your choice. In this example the software was downloaded to **/opt/sas/uaa**. If you receive a connection error, then it is likely related to network or proxy configuration. Try using the https URI instead of git.

```
[root@sasserver01 sas]# git clone git://github.com/cloudfoundry/uaa.git
Initialized empty Git repository in /opt/sas/uaa/.git/
github.com[0: 192.30.252.130]: errno=Connection refused
fatal: unable to connect a socket (Connection refused)
[root@sasserver01 sas]# git clone https://github.com/cloudfoundry/uaa.git
Initialized empty Git repository in /opt/sas/uaa/.git/
remote: Counting objects: 59021, done.
remote: Compressing objects: 100% (218/218), done.
remote: Total 59021 (delta 60), reused 0 (delta 0), pack-reused 58713
Receiving objects: 100% (59021/59021), 27.04 MiB | 10.36 MiB/s, done.
Resolving deltas: 100% (22665/22665), done.
```

6. Generate SSL keys

SSL keys are necessary to verify and sign tokens. Use the following commands to generate keys. Be sure to save them in a secure location.

```
[root@sasserver01 ~]# openssl genrsa -out privkey.pem 1024
Generating RSA private key, 1024 bit long modulus
....++++++
............++++++
e is 65537 (0x10001)
[root@sasserver01 ~]# openssl rsa -pubout -in privkey.pem -out pubkey.pem
writing RSA key
[root@sasserver01 ~]# ll *.pem
-rw-r--r--. 1 root root 887 Jan 18 13:59 privkey.pem
-rw-r--r--. 1 root root 272 Jan 18 13:59 pubkey.pem
```

7. Modify UAA port number (if currently in use)

CF UAA uses a Tomcat server to create the UAA server. The default port used by this server is 8080. If you install CF UAA on a system where an existing service is already using port 8080, such as SASServer1_1, then it will be necessary to modify the port. If you there is a conflict, an error message similar to the following will be presented:

```
org.codehaus.cargo.container.ContainerException: Port number 8080 (defined with the
property cargo.servlet.port) is in use. Please free it on the system or set it to a
different port in the container configuration.
:cargoRunLocal FAILED
```

In the UAA home directory, edit **build.gradle** and search for "port = 8080". Change it to an available port. In this example it was changed to 8079. Save the file.

8. Define location for UAA configuration updates and set associated environment variable

UAA configuration settings are defined in the YAML language. The configuration file for the CF UAA server is **uaa.yml**. A default **uaa.yml** file is laid down by the deployment, but it is recommended that it not be changed. In this example the configuration file was stored in **/opt/sas/resources**.

The format of the **uaa.yml** file involves configuration keywords followed by associated values. Details of the format are beyond the scope of this document, although several examples are shown in following steps. In order to pick up the settings it is necessary to create the **CLOUD_FOUNDRY_CONFIG_PATH** environment variable and assign it to the location where the **uaa.yml** file is saved.

```
[root@sasserver01 ~]# export CLOUD_FOUNDRY_CONFIG_PATH=/opt/sas/resources
```

9. Configure SSL keys in UAA configuration file

In order to include the SSL keys in the UAA configuration, add the private and public keys to the **uaa.yml** file. The public key will become the verification key and the private key will be the signing key. Substitute your public and private keys in the appropriate section and save your updates.

```
jwt:
  token:
    verification-key: |
       -----BEGIN PUBLIC KEY-----
       MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDL+VIH+MDJx0jnXhujzCxKdPU9
       pqexsBEj+FxdPKzmuZxxO/6vSynk0os/eXyeGTa8PgiO7EVzMbBZxeB9BQ56Crms
       qyZbJ071TRnL7SsMBjetA+0tLhoeycVp4AzETxVtTJqVI8J/INHb+CUs8fO1jPjm
       uhEP1YTtBFqe0AankQIDA/AB
       -----END PUBLIC KEY-----
    signing-key: |
       -----BEGIN RSA PRIVATE KEY-----
```

```
MIICXQIBAAKBgQDL+VIH+MDJx0jnXhujzCxKdPU9pqexsBEj+FxdPKzmuZxxO/6v
Synk0os/eXyeGTa8PgiO7EVzMbBZxeB9BQ56CrmsqyZbJ071TRnL7SsMBjetA+0t
LhoeycVp4AzETxVtTJqVI8J/INHb+CUs9fO1jPjmuhEP1YTtBFfe0AankQIDAQAB
AoGAfjo7jp1dSUGflzcA6dvXaPh6L/3rdyXbZaQ0HKsI4PvqmZSkJvzf3zCAZfab
rQczfUjyaOhEjmYomVXAd/3iF39faU0wpgF1GFsnIOeQUnEj/KJ0Te4Ug1dAmm2k
F+1mYx5B0eLEVkEjX8KR9IdwkQKfPxTaODjUUGa+/4PpgCUCQQDvFwvyKODd7aoc
JK7rBs7WbG2ajsk1pNa2oZo+wQ6umboFWg5Fhe9DK/Khrc42SODWFJrb7luwv8CS
qvBZADYrAkEA2mZ2gONzntHvNBD34E6oBDFiU86UVNoW6Ujgs+gmwW/MeODSYeY7
L5m5M7axRIT5cYBO5Ztw7ppICkl7j1oXMwJAKPnqMjaPU3nIDcq7r8wa5uTuY+7U
zTzTD7nilZ7AxWvTVdd5WrD5sKl3i/4duXAEGKyvXcIcNM6oUnq5dodywQJBAI66
08tb2l36+Qbf5/7xqKA6aRY4NXxWow6GkENC/sEAHXMKCrPsUNXE8uI3DRaoiJSC
tA0eTunAbkA60cNXrz0CQQCov5kKF9V1DyNGh2VIomAiGoNl59m4TkyTcz6LJBWl
UHJab9hs2PfBcO0FilD2NNFvMO8/pAq1s2FwvxvaTAHn
-----END RSA PRIVATE KEY-----
```

10. Configure CF client ID in UAA configuration file

It is necessary to define a client ID that will be referenced when generating a token. The same client ID is also used when starting an ESP server.
In the example below a default ID (**cf**) is created for Cloud Foundry, and an ESP-specific ID (**espadmin**) is specified for ESP-related tokens. Add the following lines to the **uaa.yml** configuration file.

```
oauth:
  clients:
    cf:
      id: cf
      authorized-grant-types: implicit
      scope: cloud_controller.read,cloud_controller.write,openid,password.write
      authorities: uaa.none
      resource-ids: none
      redirect-uri: https://uaa.cloudfoundry.com/redirect/cf
    espadmin:
      id: espadmin
      authorized-grant-types: implicit
      scope: cloud_controller.read,cloud_controller.write,openid,password.write
      authorities: uaa.login
      resource-ids: none
      redirect-uri: http://localhost/hello
```

11. Configure CF users in UAA configuration file

In this example Cloud Foundry users are defined in the **uaa.yml** configuration file. Users can also be added via the command line. User account data and encrypted password and secret values are stored in an internal database, hsqldb, or UAA can be configured to store them in a PostgreSQL or MySQL database. CF UAA can also be configured to connect to external user stores through LDAP and SAML.

The format for entering users in **uaa.yml** is as follows:

```
username|password|email|first_name|last_name(|comma-separated-authorities)
```

The users shown here, **marissa** and **markt**, are examples of defining users to UAA which by default uses an internal store for users and passwords. Add a line similar to the **markt** entry to define a user to be used when generating a token.

```
scim:
   users:
      - marissa|koala|marissa@test.org|Marissa|Blogger|scim.write,scim.read,openid
      - markt|koala|Mark.Thomas@sas.com|Mark|Thomas|scim.write,scim.read,openid
```

12. Update oauth-clients.xml with client ID

Although not noted in the documentation, it is necessary to update the Spring configuration to reflect the client ID configured in UAA. Add the following entries to the ***UAAHome*/uaa/src/main/webapp/WEB-INF/spring/oauth-clients.xml** file, where ***UAAHome*** is the install location created in step 6.

```
<entry key="espadmin">
       <map>
            <entry key="id" value="espadmin" />
            <entry key="authorized-grant-types" value="implicit" />
            <entry key="scope" value="openid,scim.read,scim.write" />
            <entry key="authorities" value="uaa.none" />
            <entry key="autoapprove" value="true" />
            <entry key="access-token-validity" value="#{new Integer(31536000)}" />
            <entry key="refresh-token-validity" value="#{new Integer(31536000)}" />
       </map>
 </entry>
```

The **access-token-validity** and **refresh-token-validity** parameters that specify expiration period in this example are set to 365 days. These values, specified in seconds, should be adjusted to meet the customer's security requirements.

13. Start CF UAA web service (using Java 1.8)

At this point we are ready to start the UAA service. Starting the service is simple. Change directory to the ***UAAHome*** directory and then issue the **gradlew run** command to build the web applications and start the web service. There are a significant number of messages at startup. When the message "`Press Ctrl-C to stop the container…`" is displayed the service is available.

```
[root@sasserver01 uaa]# ./gradlew run
:compileJava UP-TO-DATE
:compileJava took 6ms
:processResources UP-TO-DATE
.
. .
. . . …
:cleanCargoConfDir UP-TO-DATE
:cleanCargoConfDir took 0ms
:cargoRunLocal
Press Ctrl-C to stop the container...
> Building 98% > :cargoRunLocal
```

If you forget to set JAVA_HOME you may receive the following messages. Simply set JAVA_HOME as indicated in step 4.

```
FAILURE: Build failed with an exception.

* What went wrong:
Execution failed for task ':cloudfoundry-identity-client-lib:compileJava'.
> Could not find tools.jar
```

14. Generate and save an OAuth token

Now that the UAA web service is running it is possible to generate a token. Token generation is accomplished via the **`curl`** command that specifies the clientID, user name and password within the REST call, while the token is returned to the user's shell. An example of this command is displayed in the ESP user's guide. Highlighted below is the command used during my testing and the associated response. You will need the client ID, user account and password defined in earlier steps.

The token is passed back to the session and can be found in the **`&access_token`** portion of the **`Location`** field of the REST response.

```
[root@sasserver01 vwap_xml]# curl -v -H "Accept: application/json" -H "Content-Type:
application/x-www-form-urlencoded"
"http://localhost:8079/uaa/oauth/authorize?client_id=espadmin&response_type=token&scope=o
penid&redirect_uri=http://localhost:8079/hello" -d
"credentials=%7B%22username%22%3A%22markt%22%2C%22password%22%3A%22koala%22%7D"
* About to connect() to localhost port 8079 (#0)
*   Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 8079 (#0)
```

```
> POST
/uaa/oauth/authorize?client_id=espadmin&response_type=token&scope=openid&redirect_uri=htt
p://localhost:8079/hello HTTP/1.1
> User-Agent: curl/7.19.7 (x86_64-redhat-linux-gnu) libcurl/7.19.7 NSS/3.15.3 zlib/1.2.3
libidn/1.18 libssh2/1.4.2
> Host: localhost:8079
> Accept: application/json
> Content-Type: application/x-www-form-urlencoded
> Content-Length: 77
>
< HTTP/1.1 302 Found
< Server: Apache-Coyote/1.1
< Cache-Control: no-cache
< Pragma: no-cache
< Expires: Thu, 01 Jan 1970 00:00:00 GMT
< X-XSS-Protection: 1; mode=block
< X-Frame-Options: DENY
< X-Content-Type-Options: nosniff
< Cache-Control: no-store
<
Location:http://localhost:8079/hello#token_type=bearer&access_token=eyJhbGciOiJSUzI1NiJ9.
eyJqdGkiOiIxYjg0NThmMy03YzJmLTQzZGEtODk3Yy0zZmVhODZjZWY5MTMiLCJzdWIiOiJlMWYyZjk1Ny1mODQwL
TQyZjktOTdmMi02MGY3MWU4M2FkOWQiLCJzY29wZTI6WyJvcGVuaWQiXSwiY2xpZW50X2lkIjoiZXNwYWRtaW4iLC
JjaWQiOiJlc3BhZG1pbiIsImF6cCI6ImVzcGFkbWluIiwidXNlcl9pZCI6ImUxZjJmOTU3LWY4NDAtNDJmOS05N2Y
yLTYwZjcxZTgzYWQ5ZCIsIm9yaWdpbiI6InVhYSIsDnVzZXJfbmFtZSI6Im1hcmt0IiwiZW1haWwiOiJNYXJrLlRo
b21hc0BzYXMuY29tIiwiYXV0aF90aW1lIjoxNDUzMTMxMjA1LCJyZXZfc2lnIjoiMjQ1YzYzMjUiLCJpYXQiOjE0N
TMxMzEyMDUsImV4cCI6MTQ4NDY2NzIwNSwiaXNzIjoiaHR0cDovL2xvY2FsaG9zdDo4MDgwL3VhYS9vYXV0aC90b2
tlbiIsInppZCI6InVhYSIsImF1ZCI6WyJlc3BhZG1pbiIsIm9wZW5pZCJdfQ.R6321Ybn0KdP1jR2aI7HYgqMLzhy
EkFaB1_Rhl4nPGdNntp2QbKC0Gl2zenmg0dEQkZoMzvKA_YrVDlbDEs89rmBzhxq-38P-67Tsgsar-
ce4Z2jNq08CYSs8DR4wdJs_eSHISc8uI1b2KgCy7W9H8vV9E7s8Pl1WCK-
eehoJxY&expires_in=31535999&jti=1b8458f3-7c2f-43da-897c-3fea86cef913
< Content-Language: en-US
< Content-Length: 0
< Date: Mon, 18 Jan 2016 15:33:24 GMT
<
* Connection #0 to host localhost left intact
* Closing connection #0
```

Once the access token is available, it should be stored in a secure file. The file containing the token will then be referenced in future steps when the defining client connection to the ESP server. In this test the token was stored in **/opt/sas/resources/access_token.**

15. Configure public SSL key in the ESP Server

In order for the ESP server to authenticate users, the public SSL key created in an earlier step used to sign the token must be stored in a newly-created directory within the SAS ESP installation. Place this key file in **$DFESP_HOME/etc/oauth/pubkey.pem**. It will be necessary to create the **oauth** directory as it does not exist by default. Saving the key in this location makes the public key available to all ESP servers, but is only used when authentication is enabled at ESP server start.

16. Start the ESP server with authentication

Starting an ESP XML server with authentication requires that the **-auth *clientID*** parameter be specified at startup. The following is an example of starting the XML server using the vwap_xml model with authentication using the client ID created earlier.

```
[sasinst@sasserver01 vwap_xml]$  $DFESP_HOME/bin/dfesp_xml_server -model
file:///opt/sas/sashome/SASEventStreamProcessingEngine/3.2.0/examples/xml/vwap_xml/model.
xml -auth espadmin -http-pubsub 5557
```

17. Pass token as part of client call to the ESP server to authenticate

Finally, the last step is to invoke a client connection to the server. In the examples below the LASR adapter is used to subscribe to a window in the XML server started in the prior step.

The first example shows an attempt to connect to the XML server without specifying the token. Notice that the corresponding message clearly indicates that authentication is enabled on the server but the client did not specify the token.

```
[lasradm@sasserver01 ~]$ $DFESP_HOME/bin/dfesp_lasr_adapter -k sub -h
dfESP://sasserver01:5555/project/contQuery/aggW1hourRet?snapshot=true -H
sasserver01:10010 -t hps.aggW1hourRet -X
/opt/sas/sashome/SASFoundation/9.4/utilities/bin/tklasrkey -n true -a 500 -A 25
Jan 18, 2016 8:55:47 AM com.sas.esp.api.server.ReferenceIMPL.dfESPlibrary logMessage
SEVERE: queryMeta(): Authentication error: authentication is enabled on the server but
pubsub url did not contain oauth token
Mon Jan 18 08:55:47 2016 FATAL  Schema query
(dfESP://sasserver01:5555/project/contQuery/aggW1hourRet?get=schema) failed
```

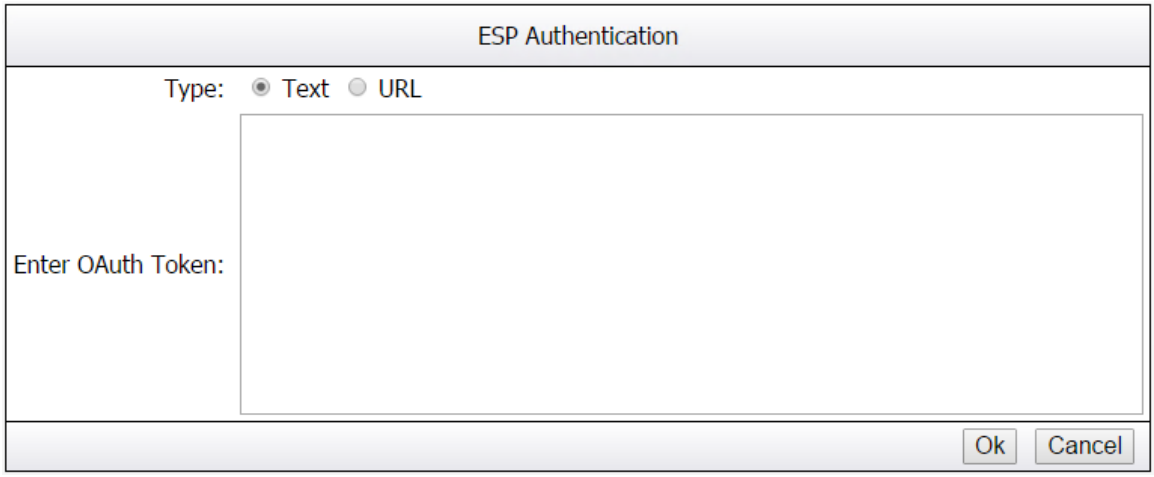
If the access token is then added to the LASR adapter command using the -O option, the connection is successful and the adapter loads data to the LASR table. Authentication success! Albeit without indication of authentication.

```
[lasradm@sasserver01 ~]$ $DFESP_HOME/bin/dfesp_lasr_adapter -k sub -h
dfESP://sasserver01:5555/project/contQuery/aggW1hourRet?snapshot=true -H
sasserver01:10010 -t hps.aggW1hourRet -X
/opt/sas/sashome/SASFoundation/9.4/utilities/bin/tklasrkey -n true -a 500 -A 25 -O
/opt/sas/resources/access_token
Tue Jan 19 15:33:15 2016 INFO   Start loading data from ESP window(s) to SAS LASR
table...
Tue Jan 19 15:33:15 2016 INFO   Setting up SAS LASR Client...
Tue Jan 19 15:33:15 2016 INFO   Obtaining LASR server version...
Tue Jan 19 15:33:15 2016 INFO   Checking if table <hps.aggW1hourRet> is loaded to the
LASR Server...
Tue Jan 19 15:33:15 2016 INFO   Table is loaded. Gathering table columns information...
Tue Jan 19 15:33:15 2016 INFO   Table <hps.aggW1hourRet> is loaded to the LASR Server
<newtable> flag was set. Recreating table...
Tue Jan 19 15:33:15 2016 INFO   <strict> flag is set. Validating adapter schema with ESP
event schema...
Tue Jan 19 15:33:15 2016 INFO   Analyzing 500 ESP events before creating empty LASR
table...
Tue Jan 19 15:33:16 2016 INFO   Analysis has been completed
Tue Jan 19 15:33:16 2016 INFO   Creating empty LASR table...
Tue Jan 19 15:33:16 2016 INFO   Empty LASR table is created successfully. Start pushing
data in...
Tue Jan 19 15:33:16 2016 INFO   Table <hps.aggW1hourRet> was reloaded successfully
Tue Jan 19 15:33:16 2016 INFO   LASR Client was set up successfully
Tue Jan 19 15:33:17 2016 INFO   Processed 1000 events. Target LASR table contains 1000
observations. Time taken: 467 ms
Tue Jan 19 15:33:17 2016 INFO   Processed 2000 events. Target LASR table contains 2000
observations. Time taken: 260 ms
Tue Jan 19 15:33:17 2016 INFO   Processed 3000 events. Target LASR table contains 3000
observations. Time taken: 229 ms
```

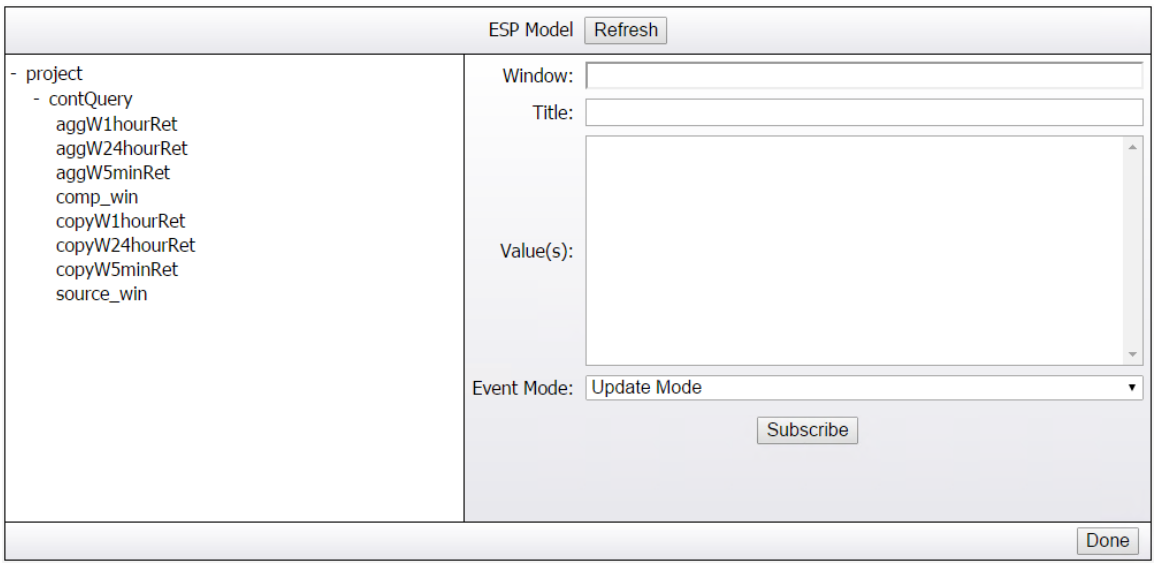
As you can see there are many steps required to establish authentication between ESP clients and servers. Most steps are straight-forward exercises that require minimal time to execute, but it may take some time to digest the purpose of each step. Review the SAS Event Stream Processing User's Guide for additional options and information. Additional references can be found at the end of this document.

Streamviewer

Earlier it was noted that authentication does not exist for SAS ESP Studio. However, authentication is enabled for SAS ESP Streamviewer. Although there is no predetermined setting to enter the OAuth token, a pop-up window is displayed when connecting to the HTTP pubsub port. The following screen shot shows the window presented for entering the OAuth token.



The text value of the token should be entered here or the URL that returns the token. Once the token is entered, the subsequent window allows the user to subscribe to existing windows in a model.



Wrapping Up

In this document we touched on the key items required to enable authentication between an ESP client and server. Since CF UAA is the tool of choice for generating OAuth tokens, it is worthwhile spending time gaining a better understanding of the capabilities of the software. In this example we manually entered key components in the **uaa.yml** configuration file. A more likely scenario is using the UAA command line to manage users and client IDs.