日日是Oracle APEX

Oracle APEXを使った作業をしていて、気の付いたところを忘れないようにメモをとります。

2022年6月14日火曜日

Keycloakを認証サーバーとしてAPEXアプリをSAMLで認証する

Keycloakを認証サーバーとして使用して、Oracle APEXのアプリケーションをSAMLにて認証させてみました。現時点で、以下の３つの問題が見つかっています。

- Keycloakが生成する公開鍵証明書をOracle APEXに登録できない。
    - X509v3の公開鍵証明書を再生成する。
- Oracle APEXのSAMLコールバックを呼び出す際にOriginヘッダーがnullになる。
    - ORDS側でOriginヘッダーがnullでも接続を許可する。
- APEXのSAMLコールバックに含まれる電子署名が無効とされる。
    - 現状、対処方法がありません。

最後の問題があるため、現在のところKeycloakによるSAML認証は利用できません。

実施した検証作業を以下に記述します。

以下の環境を想定しています。

- Keycloak 18.0.0
    - ホスト名は**idp.mydomain.dev**とします。環境に合わせて読み替えます。
    - Oracle CloudのAmpere A1インスタンス - Always Freeに実装。
    - 検証は開発モードで実施。
    - 記事 - Ampere A1インスタンスにKeycloakをインストールする
- Oracle APEX 22.1
    - ホスト名は**test.mydomain.dev**とします。環境に合わせて読み替えます。
    - Oracle CloudのVM.Standard.E4.Flex 1 OCPU、4GBに実装（課金されます）。
    - Oracle DatabaseはExpress Edition 21.3を使用。
    - ADB(APEX 21.2) + Customer Managed ORDSでも実装可能（課金されません）。
    - ORDSは22.1.1を使用。ADB+Customer Managed ORDSでは21.4.3。
    - 記事 - APEXを実装したVirutalBoxの仮想マシンをOracle Cloudにインポートする
    - 記事 - ORDSの暗号化にLet's Encryptを使用する
    - 記事 - TomcatとCertbotを使う実装にしています。

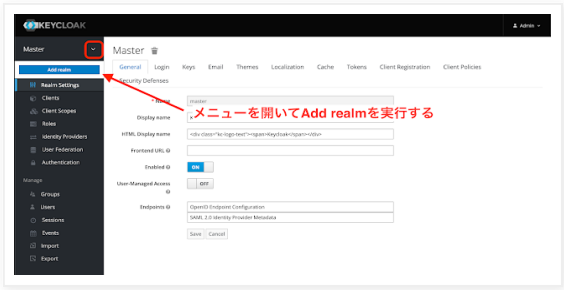
以下より、設定手順を記述します。

レルムの作成

Keycloakのサーバーにて、**レルム（Realm）**を作成します。ユーザー認証にSAMLを使用するAPEX環境は、ここで作成するレルムに**クライアント（Client）**として登録します。

Keycloakの管理サーバーにサインインします。Keycloakをインストールするとレルム**Master**があらかじめ作成されています。

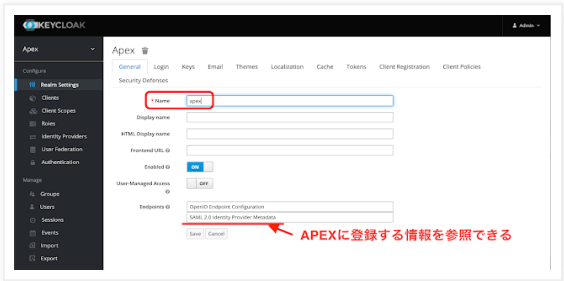
レルムの選択メニューを開き、**Add realm**を実行します。



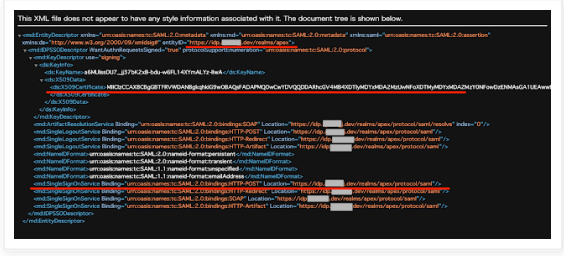
レルムの名前は任意に設定できます。今回の作業では**Name**は**apex**とします。**Enabled**は**ON**にします。

Createをクリックし、レルムを新規に追加します。

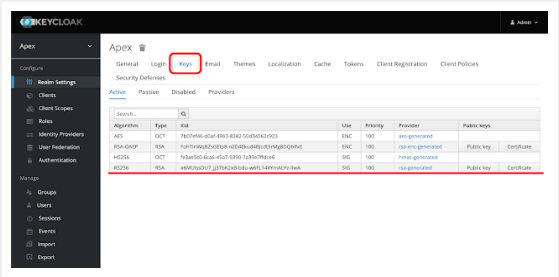
レルム**apex**が作成されます。**Endpoints**に**SAML 2.0 Identity Provider Metadata**のリンクがあります。これをクリックすると、APEX側に登録する情報を参照できます。



entityID、**X509Certificate**、**SingleSignOnService**の値は、後でAPEX側に登録します。
X509Certificateは公開鍵証明書を再作成して置き換えるため、ここで表示されている値から変わります。



Keysタブを開くと、レルムが保持する暗号キーが生成されていることが確認できます。



Oracle APEXはSAMLのIdPの公開鍵証明書がバージョン３であることを期待しています。Keycloakによって生成された公開鍵証明書はバージョン１であるため、Oracle APEXに登録できません。そのため、ワークアラウンドとして公開鍵証明書を再生成します。元々、Keycloakが生成している公開鍵証明書は自己署名証明書なので、秘密キーがあれば再生成できます。

証明書を再生成するためにレルムのデータをエクスポートし、秘密キーを取り出します。Keycloakの認証サーバーは一旦停止します。

Keycloakが動作しているサーバー**idp.mydomain.dev**にsshで接続します。**--dir**にホーム・ディレクトリ、**--realm**としてエクスポートするレルムを指定します。環境変数PATHにkc.shが含まれるディレクトリをあらかじめ追加しておきます。

kc.sh export --dir $HOME --realm apex

```
[keycloak@idp ~]$ kc.sh export --dir $HOME --realm apex
2022-06-10 06:57:19,372 INFO
[org.keycloak.quarkus.runtime.hostname.DefaultHostnameProvider] (main) Hostname
settings: FrontEnd: idp.apexugj.dev, Strict HTTPS: false, Path: <request>, Strict
BackChannel: false, Admin: <request>, Port: -1, Proxied: false
2022-06-10 06:57:19,891 INFO
[中略]
2022-06-10 06:57:22,167 INFO  [io.quarkus] (main) Profile import_export activated.
2022-06-10 06:57:22,167 INFO  [io.quarkus] (main) Installed features: [agroal, cdi,
hibernate-orm, jdbc-h2, jdbc-mariadb, jdbc-mssql, jdbc-mysql, jdbc-oracle, jdbc-
postgresql, keycloak, narayana-jta, reactive-routes, resteasy, resteasy-jackson,
smallrye-context-propagation, smallrye-health, smallrye-metrics, vault, vertx]
2022-06-10 06:57:22,320 INFO  [io.quarkus] (main) Keycloak stopped in 0.150s
[keycloak@idp ~]$
```

レルム名-realm.json、今回の例では**apex-realm.json**というファイルに、レルムのデータがエクスポートされます。管理画面の**Manage**にも**Export**というメニューがありますが、管理画面からのエスクポートでは秘密キーが含まれないようです。そのため、エクスポートはコマンド・ラインで実行する必要があります。

エクスポート・ファイルに含まれる秘密キーのデータを、openssslで扱える形式に変換する簡単なシェル・スクリプトを書きました。これを**format-pem.sh**として作成しておきます。

```
#!/bin/sh


# Usage format-pem.sh filename type
filename=$1
type=$2
```

```
# Start with Header.
if [ "${type}" = "c" ]; then
echo '-----BEGIN CERTIFICATE-----' > ${filename}
elif [ "${type}" = "p1" ]; then
echo '-----BEGIN RSA PRIVATE KEY-----' > ${filename}
fi

# Read single line PEM, then fold at 64 bytes.
echo Paste one line PEM
read l
echo $l | fold -w 64 >> ${filename}

# Append Footer.
if [ "${type}" = "c" ]; then
echo '-----END CERTIFICATE-----' >> ${filename}
elif [ "${type}" = "p1" ]; then
echo '-----END RSA PRIVATE KEY-----' >> ${filename}
fi

# verify using openssl
if [ "${type}" = "c" ]; then
openssl x509 -in ${filename} -text
elif [ "${type}" = "p1" ]; then
openssl rsa -in ${filename} -text
fi

exit;
```

format-pem.sh hosted with ❤ by GitHub　　view raw

エクスポートされたデータを参照します。**apex-realm.json**をエディタで開きます。

org.keycloak.keys.KeyProviderの配列に含まれる、**name**が**rsa-generated**、**keyUse**が**SIG**である**privateKey**のデータをクリップボードにコピーします。

format-pem.shを実行し、クリップボードにコピーした秘密キーの情報を渡します。引数の**private.pem**はopensslで扱える形式にした秘密キーのファイル（PKCS#1形式）、続く**p1**はPKCS#1を指定するオプションです。

sh format-pem.sh private.pem p1

```
[keycloak@idp ~]$ sh format-pem.sh private.pem p1
Paste one line PEM
ここでクリップボードの内容をペーストする。
MIIEowIBAAKCAQEA1kzQCcX0SRCe10iTp3rJYX8sjjAjQ4RWmGN/b0yrc2NgYEwyZsQfFPY7GjDChfnXd2J
+7o27h6q6s1kTL4RdEhToxk8GB+S0OrMvL0f6yCqQbdxNcTQW51LkC6udxpRw+15e8ooqgNxBs7MgEiRcyv
SpRUgblgq+q2hlWm+OMfEOQnaDkKcNLDRhLJK+9CRe5BaL7Z+/kIsMen84knQPpvgwlhBNMjQzRTp
[中略]

    0a:47:25:d9:6b:f9:77:cc:5d:83:34:63:3a:9a:a6:
    fe:7f:fd:5a:f2:63:7c:36:67:94:02:fd:13:3c:e3:
    15:58:e3:6e:d0:e1:fb:eb
writing RSA key
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEA1kzQCcX0SRCe10iTp3rJYX8sjjAjQ4RWmGN/b0yrc2NgYEwy
ZsQfFPY7GjDChfnXd2J+7o27h6q6s1kTL4RdEhToxk8GB+S0OrMvL0f6yCqQbdxN
cTQW51LkC6udxpRw+15e8ooqgNxBs7MgEiRcyvSpRUgblgq+q2hlWm+OMfEOQnaD
[中略]
aBy0ugEJVopg7y4OH7/L1BCCT5HVAYCc+q5+f1axqiMFOdR/hH7Pugit7sUZRUzM
8DaGSNElIlQOBh2nO2ZhREmSNEaA4YzoPHeyy7yaSRCWVbC0hvprhxNV1TpPmU+6
A4VJDQKBgD5zgz39ZNAhvvbGmNdO1VDmZ3J7XKspdbRAp4HtXMkqj2q+MYJj5N7g
SFJH1VSUVrDO7/fCMrRM65I08i0YVYdEbWgc267ylwqAeta+cy1bv0OyqyEb/bVt
HgpHJdlr+XfMXYM0YzqapV5//VryY3w2Z5QC/RM84xVY427Q4fvr
-----END RSA PRIVATE KEY-----
[keycloak@idp ~]$
```

openssslを呼び出して作成した秘密キー・ファイルの内容を印刷しています。エラーが発生しなければ、秘密キーのファイルが適切に生成されています。

作成した秘密キーのファイルより、CSR（証明書署名要求）を生成します。

openssl req -new -key private.pem -out apex.csr

Common Nameとして、レルム名を指定します。今回の例では**apex**です。それ以外は無指定にします。デフォルト値が設定されている場合は、**.** (ピリオド)を入力して空白にします。

生成されるCSRのファイル名として、**apex.csr**を指定しています。

```
[keycloak@idp ~]$ openssl req -new -key private.pem -out apex.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [XX]:.
State or Province Name (full name) []:
Locality Name (eg, city) [Default City]:.
Organization Name (eg, company) [Default Company Ltd]:.
Organizational Unit Name (eg, section) []:
Common Name (eg, your name or your server's hostname) []:apex
```

```
Email Address []:.

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
[keycloak@idp ~]$
```

自己署名証明書を生成します。証明書のバージョンが３となるよう、拡張機能を指定するファイルを作成します。内容としては以下を記載します。ファイル名は**v3.ext**とします。

keyUsage = digitalSignature

以下のコマンドを実行し、自己署名証明書**cert-apex.pem**を作成します。

openssl x509 -req -days 3650 -signkey private.pem -in apex.csr -sha256 -extfile v3.ext -out cert-apex.pem

```
[keycloak@idp ~]$ openssl x509 -req -days 3650 -signkey private.pem -in apex.csr -
sha256 -extfile v3.ext -out cert-apex.pem
Signature ok
subject=CN = apex
Getting Private key
[keycloak@idp ~]$
```

生成された自己署名証明書によってエクスポートされたデータを置き換えるため、ファイルの内容を１行で表示します。trコマンドを使用します。

tr -d '\n\r' < cert-apex.pem

```
[keycloak@idp ~]$ tr -d '\n\r' < cert-apex.pem
-----BEGIN CERTIFICATE-----MIIC/DCCAeSgAwIBAgIUR3t1+jiW0qANoL0CO+vp6N6L/zkwDQYJKo
ZIhvcNAQELBQAwDzENMAsGA1UEAwwEYXBleDAeFw0yMjA2MTAwNzMzMjVaFw0zMjA2MDcwNzMzMjVaMA8
xDTALBgNVBAMMBGFwZXgwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDWTNAJxfRJEJ7XSJOn
eslhfyyOMCNDhFaYY39vTKtzY2BgTDJmxB8U9jsaMMKF+dd3Yn7ujbuHqrqzWRMvhF0SFOjGTwYH5LQ6s
y8vR/rIKpBt3E1xNBbnUuQLq53GlHD7Xl7yiiqA3EGzsyASJFzK9KlFSBuWCr6raGVab44x8Q5CdoOQpw**
********************************** 省略****************************************
E1xNBbnUuQLq53K2EYez8YCk9cmkm/H6/r6aFSteeRMr0/xYX0WjAeZHgDsKpB/LDzKyp9Kc3oAc0XqLY
Rz1x6YnAacpCSsryhfrXh7wlWlgNdDzQlnrO3VFg9fahs56aRSxuNmaJVpiowos+yt+yJJID9LvbxkN0Q
wiF3uwcBIsJAVmC3wbGl0ihKUdD5gmqObQKUa8bIR1rdcFhCdMoicsFFlP6Qu123ZXp+vrpnOXY1BK9--
---END CERTIFICATE-----[keycloak@idp ~]$
```

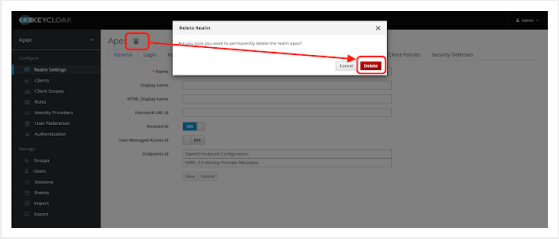
-----BEGIN CERTIFICATE-----から**-----END CERTIFICATE-----**の間のデータを、クリップボードにコピーします。

エクスポート・ファイル**apex-realm.json**を開き、先ほど取り出した秘密キー（privateKeyの項目）のデータに対応する証明書（**certificate**）の部分を置き換えます。

置き換えたファイルをインポートするにあたって、作成済みのレルムapexを削除します。

一旦、Keycloakの認証サーバーを起動し、管理ツールを開きます。

レルム名の右横の**ごみ箱アイコン**を**クリック**します。ダイアログが開いて確認を求められるので、**Delete**をクリックします。



再度、Keycloakの認証サーバーを停止し、レルムのデータをインポートします。

kc.sh import --file apex-realm.json

```
[keycloak@idp ~]$ kc.sh import --file $HOME/apex-realm.json
2022-06-10 07:52:59,350 INFO
[org.keycloak.quarkus.runtime.hostname.DefaultHostnameProvider] (main) Hostname
settings: FrontEnd: idp.apexugj.dev, Strict HTTPS: false, Path: <request>, Strict
BackChannel: false, Admin: <request>, Port: -1, Proxied: false
2022-06-10 07:52:59,978 INFO  [org.infinispan.server.core.transport.EPollAvailable]
(keycloak-cache-init) ISPN005028: Native Epoll transport not available, using NIO
instead: java.lang.UnsatisfiedLinkError: could not load a native library:
netty_transport_native_epoll_aarch_64
2022-06-10 07:53:00,130 WARN  [org.infinispan.CONFIG] (keycloak-cache-init)
ISPN000569: Unable to persist Infinispan internal caches as no global state enabled
[中略]
2022-06-10 07:53:03,458 INFO  [io.quarkus] (main) Profile import_export activated.
2022-06-10 07:53:03,458 INFO  [io.quarkus] (main) Installed features: [agroal, cdi,
hibernate-orm, jdbc-h2, jdbc-mariadb, jdbc-mssql, jdbc-mysql, jdbc-oracle, jdbc-
postgresql, keycloak, narayana-jta, reactive-routes, resteasy, resteasy-jackson,
smallrye-context-propagation, smallrye-health, smallrye-metrics, vault, vertx]
2022-06-10 07:53:03,604 INFO  [io.quarkus] (main) Keycloak stopped in 0.142s
[keycloak@idp ~]$
```

Keycloakの認証サーバーを起動し、インポートされた結果を確認します。

先ほど削除したレルム**apex**がインポートされています。**General**の**Endpoints**の**SAML 2.0 Identity Provider Metadata**のリンクを開いて、**X509Certificate**が置き換えた証明書になっていることを確

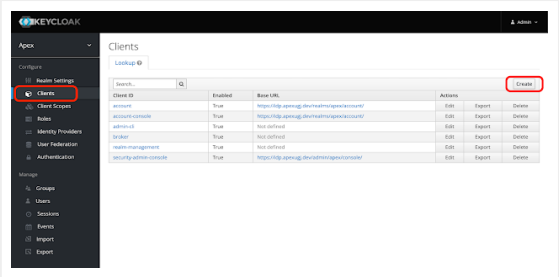
認します。

以上で、レルムapexの作成は完了です。

クライアントの作成

KeycloakのClientとして、SAML認証を行うAPEXインスタンスを登録します。レルムApexを開き、ナビゲーション・メニューよりClientsを開きます。

登録済みのクライアントが一覧されます。右上にあるCreateをクリックし、クライアントを登録します。

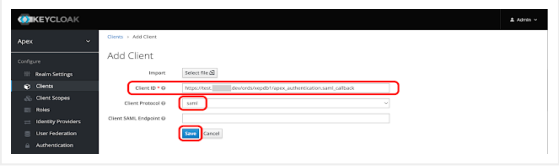


Client IDとして、APEX側のSAMLコールバックのURLを入力します。

簡易URLを使用していない開発ツールなどでf=から始まる部分に、apex_authentication.saml_callbackと記述します。ひとつのAPEXインスタンスにひとつだけ存在するエントリポイントになります。

https://test.mydomain.dev/ords/xepdb1/apex_authentication.saml_callback

Client Protocolとしてsamlを選択し、Saveを実行します。



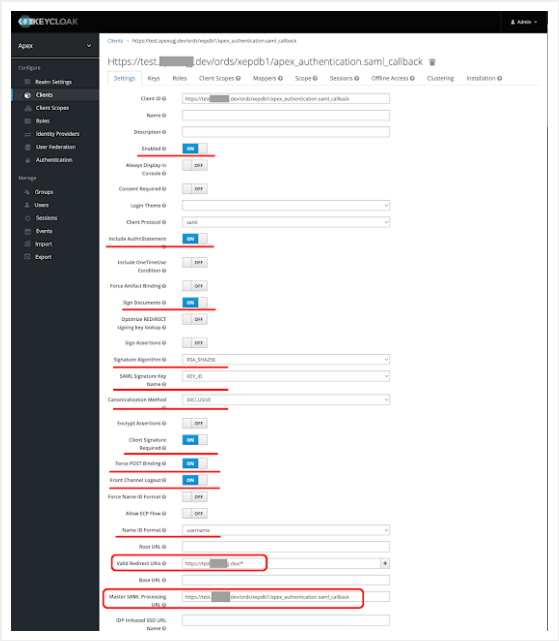
Client IDとしてSAMLコールバックのURLを指定するのは、APEXに登録する発行者属性（Issuer）のデフォルトがSAMLコールバックになるためです。KeycloakはClient IDの認識をIssuer属性から行うようです。

クライアントの詳細画面が開きます。大体はデフォルトから変更しません。

Validate Redirct URIsとして、APEXが稼働しているホストへのリダイレクトを許可します。

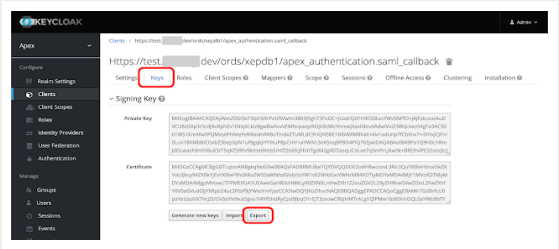
https://test.mydomain.dev/*

また、**Master SAML Processing URL**には、**Client ID**と同じ値を設定します。ここで一旦、**Save**を実行します。



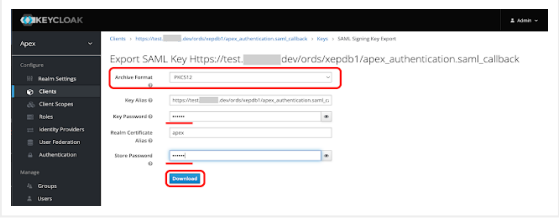
作成したクライアントに、署名に使用する暗号キーが生成されています。**Keys**タブを開きます。

ここで生成されている**公開鍵証明書（Certificate）**もバージョン１です。そのため、**Export**を実行し、バージョン３の証明書を再生成したのちImportし直します。



Archive Formatはopensslで扱いやすい**PKCS12**を選択します。Javaのkeytoolが得意な方はJKSでも良いかもしれません。**Key Password**、**Store Password**に**パスワード**となる文字列を入力し、**Download**を実行します。

keystore.p12というファイルがダウンロードされます。



ダウンロードされたPKCS12のファイルの内容を表示します。

openssl pkcs12 -in keystore.p12 -nodes

```
% openssl pkcs12 -in keystore.p12 -nodes
Enter Import Password: パスワードを入力する
MAC verified OK
Bag Attributes
    localKeyID: 10 69 9B E0 32 C5 F7 AB C6 16 5C A7 83 B2 39 DC CF 6A 82 02
    friendlyName:
https://test.mydomain.dev/ords/xepdb1/apex_authentication.saml_callback
Key Attributes: <No Attributes>
-----BEGIN PRIVATE KEY-----
MIIEvAIBADANBgkqhkiG9w0BAQEFAASCBKYwggSiAgEAAoIBAQDJa+xnQa33FNKW
mHlGs+iF+1XCWbfzr0nmCHvcWg4L6NTF39B3UcgqXfSW6jtaxI09876MAkWyRzO6
wC4NUJTzMNenetJx0kC9GmFm/W02KpUuQjPyDAHBe+9UQx+yulq/K1E6Nv3MxyGG
[ BEGIN PRIVATE KEYからEND PRIVATE KEYまでをファイルpkey.pemに取り出す]
CJ2lJqVYxH61FXXzgSHOT6kmw11O4w59K2cVEkECgYBrt45iQaM4QUZXwJcMfqtR
onDJkN+YsxRo87Y61lHwVDXcNToHe7DtAaVs6C3P6FBxzR2/lfd9f9v7pMvo/P5B
1qD4dtndzPPBn55qLR/vNzShmrdDVmSd7ZduXZDAkcea3Jxao4IX8R0AfJtL3Ie6
pRxKu/Y5Ye/VYlotiIzXsQ==
-----END PRIVATE KEY-----
Bag Attributes
    localKeyID: 10 69 9B E0 32 C5 F7 AB C6 16 5C A7 83 B2 39 DC CF 6A 82 02
    friendlyName:
https://test.apexugj.dev/ords/xepdb1/apex_authentication.saml_callback
subject=/CN=https://test.mydomain.dev/ords/xepdb1/apex_authentication.saml_callback
issuer=/CN=https://test.mydomain.dev/ords/xepdb1/apex_authentication.saml_callback
-----BEGIN CERTIFICATE-----
MIIDGzCCAgMCBgGBTLqtvzANBgkqhkiG9w0BAQsFADBRMU8wTQYDVQQDDEZodHRw
czovL3Rlc3QuYXBleHVnai5kZXYvb3Jkcy94ZXBkYjEvYXBleF9hdXRoZW50aWNh
dGlvbi5zYW1sX2NhbGxiYWNrMB4XDTIyMDYxMDA4MjY1MVoXDTMyMDYxMDA4Mjgz
MVowUTFPME0GA1UEAwxGaHR0cHM6Ly90ZXN0LmFwZXh1Z2ouZGV2L29yZHMveGVw
ZGIxL2FwZXhfYXV0aGVudGljYXRpb24uc2FtbF9jYWxsYmFjazCCASIwDQYJKoZI
[ これは置き換え対象の証明書なので不要 ]
Suai+a9htVNrUyFWWoNvu5SivLmOx+2d9SZNaZo43n6vAAcx9lEIN/XTlDyXsXb7
80YFHXNEze++wCwTvrscYdn5GEdhG5cUvTwPerB/gwxtiGZk7Y3YHpnfE1xK5/id
GdmIIo44D3p4HMqBwrv/x9E++bDBxPrGcBUdLAJ0v6ACr0VMEldt3MjNjxld22Mf
LorCSlFHUyL1aXk47VHjLJUaXj70JLxvktrtci31cQ==
-----END CERTIFICATE-----
Bag Attributes
    friendlyName: apex
subject=/CN=apex
issuer=/CN=apex
-----BEGIN CERTIFICATE-----
MIIC/DCCAeSgAwIBAgIUR3t1+jiW0qANoL0CO+vp6N6L/zkwDQYJKoZIhvcNAQEL
BQAwDzENMAsGA1UEAwwEYXBleDAeFw0yMjA2MTAwNzMzMjVaFw0zMjA2MDcwNzMz
MjVaMA8xDTALBgNVBAMMBGFwZXgwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEK
AoIBAQDWTNAJxfRJEJ7XSJOneslhfyyOMCNDhFaYY39vTKtzY2BgTDJmxB8U9jsa
MMKF+dd3Yn7ujbuHqrqzWRMvhF0SFOjGTwYH5LQ6sy8vR/rIKpBt3E1xNBbnUuQL
[ レルムの公開鍵証明書 - cert-apex.pemとして取り出す。 ]
AkHmVpsBkfKFji9TWpFbfuOJuQtTCK2EYez8YCk9cmkm/H6/r6aFSteeRMr0/xYX
0WjAeZHgDsKpB/LDzKyp9Kc3oAc0XqLYRz1x6YnAacpCSsryhfrXh7wlWlgNdDzQ
lnrO3VFg9fahs56aRSxuNmaJVpiowos+yt+yJJID9LvbxkN0QwiF3uwcBIsJAVmC
3wbGl0ihKUdD5gmqObQKUa8bIR1rdcFhCdMoicsFFlP6Qu123ZXp+vrpnOXY1BK9
-----END CERTIFICATE-----
%
```

秘密キーのデータおよび２つの公開鍵証明書が含まれます。**BEGIN PRIVATE KEY**から**END PRIVATE KEY**の間のデータは、別のファイル**pkey.pem**を作成し内容をコピーします。

先ほど作成したレルムの証明書**cert-apex.pem**が残っていれば、公開鍵証明書のデータを取り出して保存しておく必要はありません。そうでない場合は、friendlyNameやCNがレルム名（この場合

はapex）である証明書のデータを別ファイルcert-apex.pemを作成し、内容をコピーしておきます。

friendlyNameがSAMLコールバックである公開鍵証明書は置き換え対象なので、取り出して保存する必要はありません。

最初にファイルkeystore.p12から取り出した秘密キーのファイル・フォーマットを、PKCS#1に変換します。

以下を実行し、pkey.pemからprivate.pemを生成します。

openssl rsa -in pkey.pem -out private.pem

以降の作業は、レルムの公開鍵証明書を再生成したときの手順と同じです。

最初にCSRを生成します。ファイル名は**test.csr**になります。

openssl req -new -key private.pem -out test.csr

元々の証明書のCommon NameはSAMLコールバックのURLですが、CSRの生成を実行するOSによっては64文字を超えると怒られるため、ホスト名のみをCommon Nameに指定しています。

```
% openssl req -new -key private.pem -out test.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) []:
State or Province Name (full name) []:
Locality Name (eg, city) []:
Organization Name (eg, company) []:
Organizational Unit Name (eg, section) []:
Common Name (eg, fully qualified host name) []:test.mydomain.dev
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
ynakakoshi@NsM1MacBookPro k %
```

公開鍵証明書を生成します。ファイル名は**cert-test.pem**になります。**v3.ext**の内容は以下とします。

keyUsage = keyEncipherment

ただし、SAML Assertionはデフォルトでは暗号化しません。そのため、この証明書に紐づいた秘密キーは使用されないはずです。

openssl x509 -req -days 3650 -signkey private.pem -in test.csr -sha256 -extfile v3.ext -out cert-test.pem

```
% openssl x509 -req -days 3650 -signkey private.pem -in test.csr -sha256 -extfile
v3.ext -out cert-test.pem
Signature ok
subject=/CN=test.mydomain.dev
Getting Private key
%
```

インポートに使用するPKCS12形式のファイルを生成します。今回生成した**cert-test.pem**を含めて、keystore.p12から取り出したデータを連結します。ファイル名は**keystore.pem**となります。

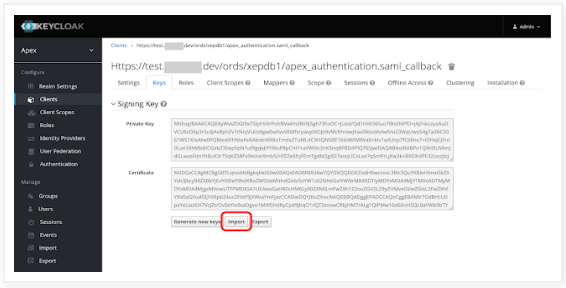
cat pkey.pem cert-test.pem cert-apex.pem > keystore.pem

keystore.pemをPKCS12形式に変換します。PKCS12形式のファイル名は**import.p12**となります。**-name**に**test**と指定することで、friendlyNameを設定しています。KeycloakでPKCS12のファイルをインポートする際に、**Key Alias**として指定します。

openssl pkcs12 -export -in keystore.pem -name test -caname apex -out import.p12

```
% cat pkey.pem cert-test.pem cert-apex.pem > keystore.pem
% openssl pkcs12 -export -in keystore.pem -name test -caname apex -out import.p12
Enter Export Password:*********
Verifying - Enter Export Password:********
%
```

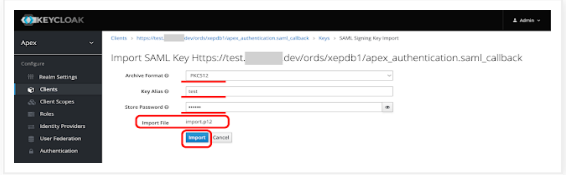
Keycloakの管理画面に戻り、**Keys**タブの画面より**Import**を実行します。



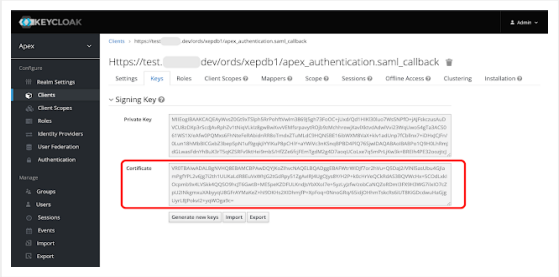
Importを行う画面が開きます。

Archive Formatに**PKCS12**を選択します。**Key Alias**にはopenssl pkcs12コマンドを実行したときに-nameに与えた**test**を設定します。**Store Password**は実行時に入力した**Export Password**を設定します。

Import Fileに**import.p12**を選択し、**Import**を実行します。



Importが成功したら、**Certificate**が再生成したcert-test.pemと同じ内容であることを確認します。



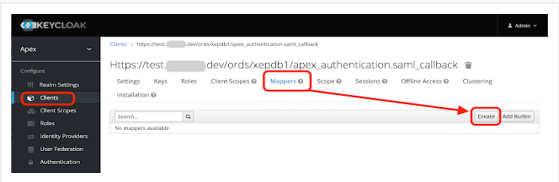
KeycloakのレルムにAPEXのインスタンスをクライアントとして登録する作業は、以上で完了です。

Keycloakでの最低限の設定

Keycloakでユーザー認証を行うにあたって必要な、最小限の設定を行います。

ナビゲーション・メニューからClientsを開き、選択したクライアントのMappersタブを開きます。

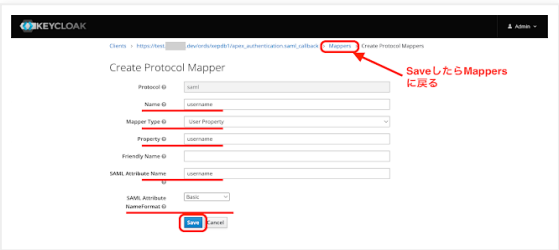
Createを実行します。



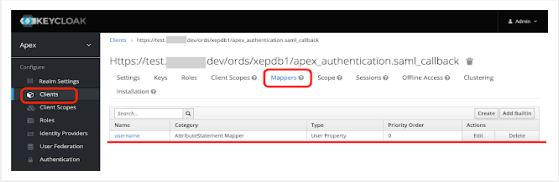
クライアントが扱う属性usernameに、SAML属性のusernameを割り当てます。

Nameに**username**、**Mapper Type**に**User Property**、**Property**に**username**、**SAML Attribute Name**に**username**、**SAML Attribute NameFormat**に**Basic**を選選択します。

以上で**Save**を実行します。Saveした後に、ナビゲーションのリンクより**Mappers**に戻ります。

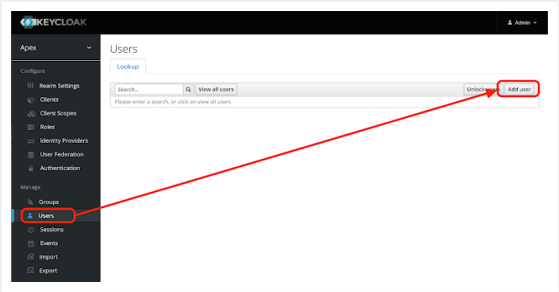


Mappersの画面にて、**username**が登録されていることを確認します。

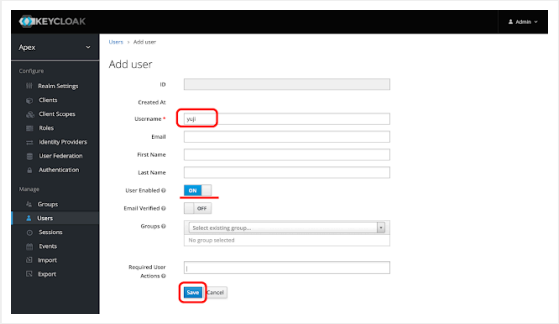


続いて、サインインに使用するユーザーを作成します。

Manageの**Users**を開き、**Add user**を実行します。



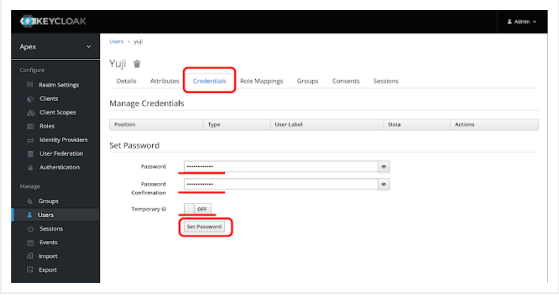
Usernameを入力します。**User Enabled**は**ON**にして、**Save**を実行します。



ユーザーが作成されたら**Credentials**タブを開き、パスワードを設定します。

TemporaryがONだと初回ログイン時にパスワードの変更を求められます。今回はテストに使うユーザーなので、**OFF**にします。

Set Passwordを実行し、パスワードを設定します。

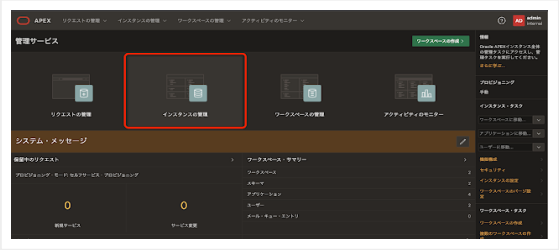


以上で、テストに使用するためのKeycloak側の最低限の設定ができました。

APEXでのSAML認証スキームの設定

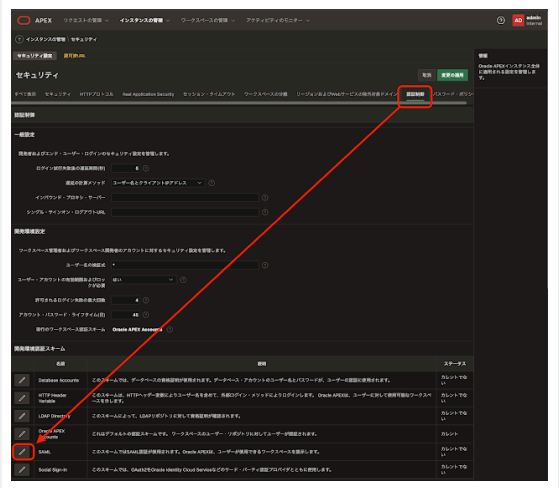
APEXの**管理サービス**にサインインし、SAML認証の設定を行います。

インスタンスの管理を開きます。



インスタンスの設定の**セキュリティ**を開きます。

認証制御タブを選択し、**開発環境認証スキーム**に含まれる**SAML**を開きます。



内部およびワークスペース・アプリケーション用のSAML: APEX属性の**アプリケーションのSAMLの有効化**を**ON**にします。

発行者（Issuer）はデフォルトにします。そのため**SAMLのコールバックURLが発行者**になります。

証明書はクライアントを登録したときに作成したファイル**cert-test.pem**の内容を貼り付けます。
秘密鍵はcert-test.pemを作成するときに使った、PKCS#1形式のファイル**private.pem**の内容を貼り付けます。

内部およびワークスペース・アプリケーション用のSAML: アイデンティティ・プロバイダ属性の**発行者**は、**SAML 2.0 Identitify Provider Metadata**に含まれる**entityID**を設定します。Keycloakでは、Keycloakの認証サーバーのホストに**/realms/レルム名**を付加したURLになります。

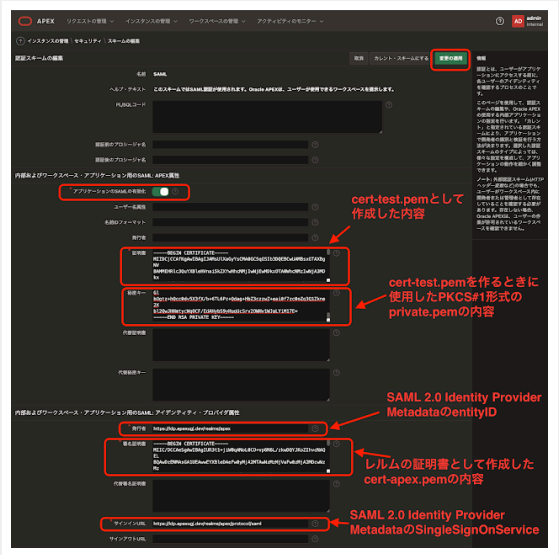
https://idp.mydomain.dev/realms/apex

署名証明書として、レルムを作成したときに再生成した証明書**cert-apex.pem**の内容を貼り付けます。

サインインURLには、**SAML 2.0 Identitify Provider Metadata**に含まれる**SingleSignOnService**のURLを設定します。Keycloakでは、Keycloakの認証サーバーのホストに/realms/レルム名/protocol/samlを付加したURLになります。

https://idp.mydomain.dev/realms/apex/protocol/saml

以上を設定し、**変更の適用**を行います。



以上で、APEX側のSAML認証を使用する設定は完了です。

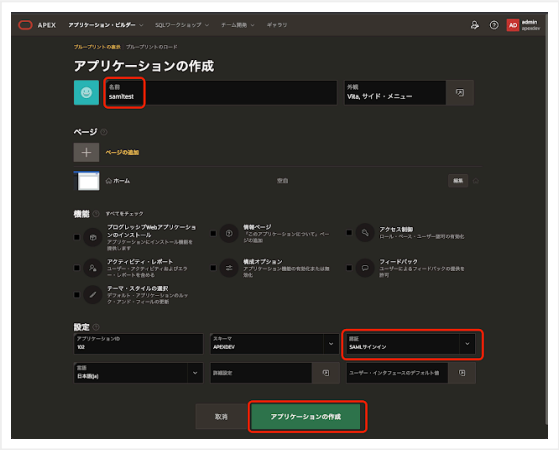
確認に使用するAPEXアプリケーションの作成

認証スキームとしてSAML認証を設定しただけの、空のアプリケーションを作成します。

ワークスペースにサインインし、**アプリケーション作成ウィザード**を起動します。

名前を**samltest**、**設定**の**認証**として**SAMLサインイン**を選択します。

以上で**アプリケーションの作成**を実行します。



テスト用のアプリケーションはこれで完成です。

Oracle REST Data ServicesのCORS設定

テスト用に作成したアプリケーションを実行すると、以下のエラーが発生します。

マニュアルのこちらに記載があるように、ORDSでクロス・オリジン・リソース共有を行うには、明示的な許可が必要です。そのため、パラメータ**security.externalSessionTrustedOrigins**に設定を追加します。

この機能はORDSが受け取るOrignヘッダーの内容に基づいて、接続を許可しているようです。そのため、本来であればこのパラメータに設定する値は、Keycloakの認証サーバーのURLである以下の値が正しいはずです。

https://idp.mydomain.dev

しかし、Keycloakにてサインインしたページから呼び出される、SAMLコールバックへのHTTPリクエストのOriginヘッダーはnullになっています。

ワークアラウンドとして**security.externalSessionTrustedOrigins**に**null**を設定します。おそらくKeycloak側に何らかの設定を追加して、Originヘッダーに値が入るようにする必要があると思われます。

ORDS 22.1以降では、以下のコマンドで設定します。ordsコマンドの位置や構成ディレクトリの位置は、それぞれのインストールによって変わります。

/usr/local/bin/ords --config /etc/ords/config config set security.externalSessionTrustedOrigins null

```
[oracle@apex ~]$ /usr/local/bin/ords --config /etc/ords/config config set
security.externalSessionTrustedOrigins null

ORDS: Release 22.1 Production on Fri Jun 10 11:03:48 2022

Copyright (c) 2010, 2022, Oracle.

Configuration:
  /etc/ords/config/

The global setting named: security.externalSessionTrustedOrigins was set to: null
[oracle@apex ~]$
```

ORDS 21.xまででであれば、実行するコマンドは以下になります。

java -jar ords.war set-property security.externalSessionTrustedOrigins null

または、構成ファイルの**defaults.xml**に以下の記述を追加します。

<entry key="security.externalSessionTrustedOrigins">null</entry>

設定変更を反映するには、ORDSを再起動する必要があります。

SAMLサインインの確認

作成したAPEXアプリケーションに接続し、SAMLによるサインインを確認します。

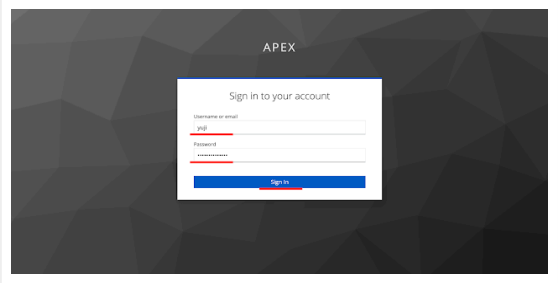
https://ホスト名/ords/PDB名/r/ワークスペース名/samltest/home

今回の例では、以下のURLにアクセスします。

https://test.mydomain.dev/ords/xepdb1/r/apexdev/samltest/home

Keycloakでのサインイン画面が表示されます。

ユーザー名とパスワードを入力し、Sign-Inを実行します。



サインインには成功しますが、最後の最後でエラーが発生します。呼び出されたSAMLコールバックで発生しています。

この現象については、おそらく不具合と思われます。ワークアラウンドは無く、開発からの修正待ちです。

コンピュート・インスタンス上に構成したAPEXでの作業を記載しましたが、Autonomous DatabaseとCustomer Managed ORDSを組み合わせた構成でも、同じ作業が可能です。結果も同様に、上記の不具合がShow Stopperになっています。

以上になります。

完

Yuji N. 時刻: 17:09

共有

ウェブ バージョンを表示

自己紹介

Yuji N.

日本オラクル株式会社に勤務していて、Oracle APEXのGroundbreaker Advocateを拝命しました。
こちらの記事につきましては、免責事項の参照をお願いいたします。

詳細プロフィールを表示