Introduction

In this article we will have a look at how to use the Alfresco Identity Services for centralized authentication and single sign on (SSO) with the Alfresco Digital Business Platform (DBP).

Alfresco DBP consists of Alfresco Content Services (ACS), Alfresco Process Services (APS), Alfresco Governance Services (AGS), and a number of modules and development frameworks, such as Alfresco Application Development Framework (ADF).

The Alfresco Identity Service has been available to use since ACS 6.0, APS 1.9, and ADF 2.4 product releases. In this article we will be using the following product versions: Identity Service 1.2, ACS 6.2.1, APS 1.11 and ADF 3.8.

This means that the applications that we are working with, such as ACS, APS, and ADF clients don't have to deal with login forms and authentication. Once a user is logged into the Alfresco Identity Service they don't have to login again to access ACS, APS, or any ADF application.

This also applies to logout, which means that once a user is logged out of Alfresco Identity Service they are also automatically logged out of all other applications.

Alfresco Identity Service is implemented on top of JBoss Keycloak, which is both an Identity Provider (IdP) and a token issuer for OAuth 2 tokens. Keycloak deals with authentication, safety password storage, SSO, two factor authentication etc. Keycloak supports protocols such as OpenID Connect and SAML. Keycloak can store the user data in a variety of places, such as LDAP, Active Directory, and RDBMS.

Alfresco Identity Service is basically a wrapper around the JBoss Keycloak service.

Source Code for this article

The source code for this article can be found here: https://github.com/gravitonian/dbp-and-identity-

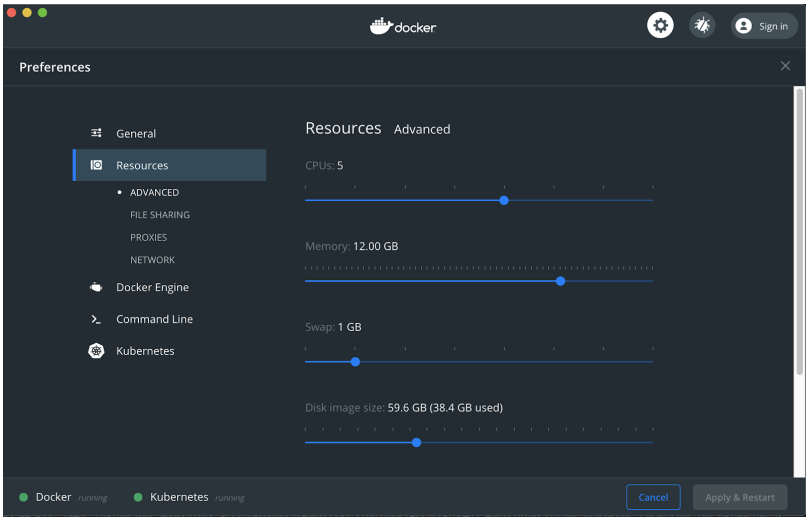
```
remote: Enumerating objects: 31, done.
remote: Counting objects: 100% (31/31), done.
remote: Compressing objects: 100% (22/22), done.
remote: Total 31 (delta 4), reused 31 (delta 4), pack-reused 0
Unpacking objects: 100% (31/31), done.

$ cd my-dbp-identity-service/
my-dbp-identity-service mbergljung$ ls -la
total 16
drwxr-xr-x   7 mbergljung  staff   224 20 May 14:08 .
drwxr-xr-x  11 mbergljung  staff   352 20 May 14:08 ..
drwxr-xr-x  12 mbergljung  staff   384 20 May 14:08 .git
-rw-r--r--   1 mbergljung  staff   278 20 May 14:08 .gitignore
-rw-r--r--   1 mbergljung  staff  1230 20 May 14:08 README.md
drwxr-xr-x   7 mbergljung  staff   224 20 May 14:08 docker-compose-dbp
drwxr-xr-x   4 mbergljung  staff   128 20 May 14:08 docker-compose-identity
```

Remove the github project info so it's not associated with my Github account:

```
my-dbp-identity-service mbergljung$ rm -rf .git
```

Docker resources settings

Docker will need quite a bit of memory available to run this solution, I have the following settings in Docker:



Note that 12GB RAM is available.

Using multiple Docker Compose files for Deployment

We will do all application deployments with Docker Compose. The Identity Service needs to be available before the ACS service starts. This is because ACS will call the Identity Service during the initialization of its 'Authentication' subsystem. Docker Compose does not provide a way to wait for a service to be ready, like you can do in a Kubernetes deployment. Because of this we will use two different Docker Compose files, one for the Alfresco Identity Service, which will be deployed first, and another one for the Alfresco DBP.

The Alfresco Identity Service also needs to be configured with an Alfresco Realm, client, etc before we start the DBP. It's possible to import a realm during deployment of the Identity Service image, but in this article we are creating the Alfresco realm from scratch so we can get a better understanding on what's going on.

Custom built PostgreSQL Docker Image

To get the solution to work on Windows we need to build a custom Docker Image for the PostgreSQL server. This image will have an **init.sql** script that will create the three databases that we need for ACS, APS, and Identity Service (i.e. Keycloak):

```
CREATE USER alfresco WITH PASSWORD 'alfresco';
CREATE DATABASE keycloak;
```

This way we can share the Postgres server between all components in the DBP deployment and solve the problem with running on Windows.

If we were running solely on Linux based systems, including Mac, then we could use a Docker Compose service definition like this for PostgreSQL:

```
services:
   postgres:
       image: postgres:11.4
       mem_limit: 512m
       environment:
           - POSTGRES_PASSWORD=alfresco
           - POSTGRES_USER=alfresco
           - POSTGRES_DB=alfresco
       command: postgres -c max_connections=300 -c log_min_messages=LOG
       ports:
           - 5433:5432
       volumes:
           - ./data/postgres:/var/lib/postgresql/data
           - ./docker-postgresql-multiple-databases:/docker-entrypoint-initdb.d
```

However, when you try and run this on for example a Windows 10 system it will generate errors when it tries to do the local Docker Volume mappings.

We solve this by using the following Docker Compose service definition:

```
   postgres:
       container_name: postgres_custom
       build:
           context: ./postgres
           dockerfile: Dockerfile
       image: postgres_custom:development
       mem_limit: 512m
       command: postgres -c max_connections=300 -c log_min_messages=LOG
       ports:
           - 5433:5432
       volumes:
           - pgdata:/var/lib/postgresql/data
```

Two things to note here: first that we build a custom Docker Image and the **Dockerfile** for this is located in the **/postgres** relative directory and looks lile this:

```
FROM postgres:11.4
COPY init.sql /docker-entrypoint-initdb.d/
```

The Dockerfile copies the **init.sql** file into a directory that Postgres will check when it starts. This SQL script will be run the first time the server starts and there is no database available (i.e. there is no data yet).

Secondly we use a named docker volume (i.e. pgdata) instead of a local directory mapping, which does not work on Windows. The named volume is defined at the end of the Docker Compose file:

```
volumes:
 pgdata:
```

Deploying and Running Alfresco Identity Service (Keycloak)

The Alfresco Identity Service Docker image is based on the JBoss Keycloak Docker image. We will build a Docker Compose file that starts up Alfresco Identity Service 1.2 and the PostgreSQL database 11.4.

> If you wanted to have a quick look at keycloak you can kick it off like this:
>
> ```
> $ docker run -p “8080:8080” -e KEYCLOAK_USER=admin -e
> KEYCLOAK_PASSWORD=secret jboss/keycloak
> ```
>
> This uses a built in H2 database, which we actually don’t want to use as we lose all data as soon as the container is removed...

Identity Service Docker Compose file

The Docker Compose file that we will be using to deploy the Identity Service looks as follows (see **my-dbp-identity-service/docker-compose-identity/docker-compose.yml**):

```
version: "2"

services:
```

```
            context: ./postgres
            dockerfile: Dockerfile
        image: postgres_custom:development
        mem_limit: 512m
        command: postgres -c max_connections=300 -c log_min_messages=LOG
        ports:
            - 5433:5432
        volumes:
            - pgdata:/var/lib/postgresql/data

    auth:
        image: quay.io/alfresco/alfresco-identity-service:1.2
        environment:
          KEYCLOAK_USER: admin
          KEYCLOAK_PASSWORD: admin
          # KEYCLOAK_IMPORT: alfresco-realm.json
          DB_VENDOR: postgres
          DB_ADDR: "postgres:5432"
          DB_DATABASE: keycloak
          DB_USER: alfresco
          DB_PASSWORD: alfresco
          JAVA_OPTS: "-Djboss.socket.binding.port-offset=808 -Djava.net.preferIPv4Stack=true
-Djava.net.preferIPv4Addresses=true"
        ports:
          - 8888:8888
        depends_on:
          - postgres

#    ldap:
#        image: greggigon/apacheds
#        environment:
#          - BOOTSTRAP_FILE=/bootstrap/demo.ldif
#        restart: always
#        ports:
#          - 10389:10389
#        volumes:
#          - ./data/ldap/data:/data
#          - ./ldap/bootstrap:/bootstrap

# We need a named Docker Volume instead of local mapping, which does not work on Windows
volumes:
 pgdata:
```

The file starts with the definition of a PostgreSQL database service called `postgres`. For more information about this custom built Postgres Docker Image see previous section. We use an external database as we want our Identity Service configuration to persist if the Identity Service container is stopped and removed.

The database files are persisted outside the container via the named volume mapping. You can find the location by doing the following:

```
C:\Users\gravi>docker volume ls
DRIVER              VOLUME NAME
local               docker-compose-identity_pgdata
...

C:\Users\gravi>docker volume inspect docker-compose-identity_pgdata
[
    {
        "CreatedAt": "2020-06-02T14:09:58Z",
        "Driver": "local",
        "Labels": null,
        "Mountpoint": "/var/lib/docker/volumes/docker-compose-
identity_pgdata/_data",
        "Name": "docker-compose-identity_pgdata",
        "Options": null,
        "Scope": "local"
    }
]
```

The named volume is located in the Docker linux VM on Windows.

The Postgres container will automatically create the Keycloak database via the DB init script copied into the `postgres` container during build time.

The second service, called `auth`, defines the Identity Service (i.e. keycloak). It is configured with the following environment variables (for more info about available configuration variables see the keycloak image docs):

Variable	Value	Description
KEYCLOAK_USER	admin	This is the initial Administrator username that will be created the first time keycloak is started. You can use this user to login

		store the configuration of the Identity Service.
DB_ADDR	`postgres:5432`	The host where the PostgreSQL database server is running. In this case it points to the internal docker network hostname `postgres` (the same as the service name in a Docker Compose file). Plus the port that the PostgreSQL server is listening on. **Note**. this is the internal port, if you map an external port to something else, then you still need to use 5432.
DB_DATABASE	`keycloak`	The name of the keycloak database in the PostgreSQL server. This database is created the first time PostgreSQL starts via an init script called **docker-postgresql-multiple-databases/create-multiple-postgresql-databases.sh** (see source code)
DB_USER	`alfresco`	The user that should be used to login to PostgreSQL and access the `keycloak` database.
DB_PASSWORD	`alfresco`	PostgreSQL user password.
JAVA_OPTS	`"-Djboss.socket.binding.port-offset=808 -Djava.net.preferIPv4Stack=true -Djava.net.preferIPv4Addresses=true"`	In order to not clash with the Alfresco NGinx proxy, which runs on port 8080 and is used in the DBP deployment, we change the external port to 8888. Because we changed the external port to 8888 we also need to change the internal port to the same (see **ports** for explanation). The internal port is set to 8080 by default and if we offset it with 808 we get 8888. The other two variables get around an error during the start up.
ports	`8888:8888`	This is the external port mapping. The keycloak server will start up on port 8888 in the internal Docker network and be exposed on port 8888 externally. At the moment we cannot configure a different external port as then we cannot configure Keycloak Auth Server URL properly from for example APS. More on this when we configure APS.

The last service, called `ldap`, is commented out and we will look at LDAP integration at the end of the article.

The volume definitions and the postgres Docker image build are supported by the following directory structure in the project (see source code):

```
docker-compose-identity mbergljung$ tree
.
├── data (will be created when ldap server is added)
│   └── ldap       (ldap directory data - users)
├── docker-compose.yml
├── postgres
│   ├── Dockerfile (Used to build custom postgres Docker image)
│   └── init.sql          (SQL init script that will create ACS,APS,Keycloak DBs)
└── ldap           (we will talk about the LDAP integration at the end)
    └── bootstrap
        └── demo.ldif
```

Running the Identity Service and Database with Docker Compose

Some of the Alfresco Docker images we will be using, such as the Identity Service Image, are Enterprise images that are hosted in the Quay Docker Image Repository. If you don't have access to them, then you need to sign up for an ACS Trial to get temporary access. However, most of the images are available on Docker Hub, which doesn't require a login.

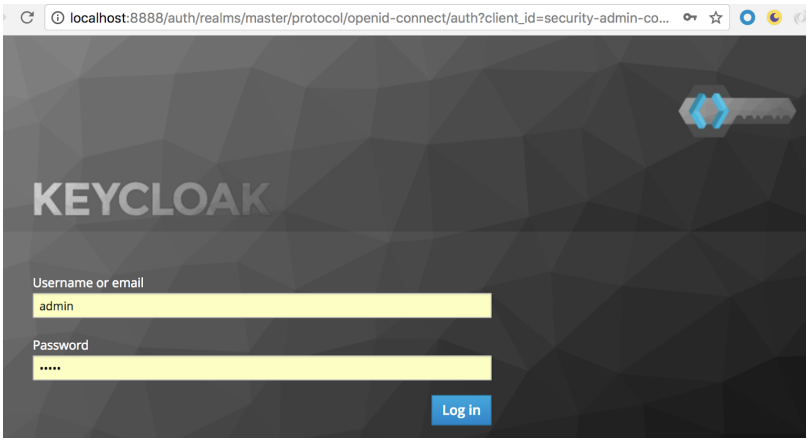
To set up access to Alfresco enterprise docker images do a Docker Login as follows with the information from your "*Start Your Trial Now*" email:

```
$ docker login quay.io -u="alfresco+acs_v6_trial"

Enter the following password when prompted (Copy and Paste):
<YOUR SECRET/PASSWORD FROM THE EMAIL>
```

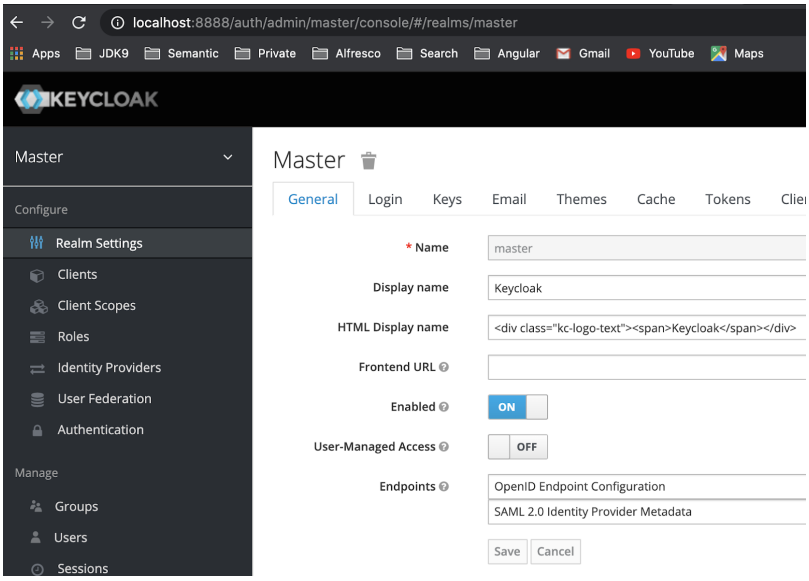
Now, to start the Identity Service and the Postgres Service, including building the custom PostgreSQL image with all the databases, just step into the **my-dbp-identity-service/docker-compose-identity** directory and execute (note the `--build` directive, which will build the custom postgres

```
Building postgres
Step 1/2 : FROM postgres:11.4
 ---> 53912975086f
Step 2/2 : COPY init.sql /docker-entrypoint-initdb.d/
 ---> 43d403b379b0
Successfully built 43d403b379b0
Successfully tagged postgres_custom:development
Creating postgres_custom                ... done
Creating docker-compose-identity_auth_1 ... done
Attaching to postgres_custom, docker-compose-identity_auth_1
…
auth_1      | 08:44:17,330 INFO  [org.jboss.as] (Controller Boot Thread)
WFLYSRV0025: Keycloak 8.0.1 (WildFly Core 10.0.3.Final) started in 37179ms -
Started 684 of 989 services (701 services are lazy, passive or on-demand)
```

Now access Alfresco Identity Service (i.e. Keycloak) at http://localhost:8888/auth/admin/master/console, login using **admin/admin**:



You should see the following screen with the Master realm info after successful login:



We are now ready to start configuring keycloak for use with ACS, APS, and ADF.

Configuring Alfresco Identity Service (keycloak)

Alfresco Identity Service is built on Keycloak and uses the OAuth 2.0 standard. OAuth is directly related to OpenID Connect (OIDC), which is an authentication layer built on top of OAuth 2.0.

OAuth Background Information

Before we dig into configuring the Identity Service it's worth getting some background information on what OAuth is and how it relates to so-called Single-Page Web Applications (SPA). This is relevant to

challenges when it comes to authentication:

- The security characteristics of these apps are significantly different from traditional server-based web applications.
- Many authorization servers and identity providers do not support CORS requests.
- Full page browser redirects away from the Single-page app become particularly invasive to the user experience.

For these applications, such as Alfresco ADF based Single-page apps, Alfresco Identity Services supports the OAuth 2.0 *Implicit Grant flow*. The implicit flow is described in the OAuth 2.0 Specification.

This flow is called implicit flow because the authentication is implicit from a redirect when the user has successfully logged in. After the user has logged in the authorization server returns a redirect to the client containing the access token. Be aware that this flow should not use refresh tokens as these can be stolen from the browser with huge security consequences.
When working with implicit flow it is crucial to always use HTTPS so a man in the middle attack is not intercepting the access token.

The primary benefit of Implicit Grant flow is that it allows the app to get tokens from the Identity Service without performing a backend server credential exchange. This allows the app to login the user, maintain a session, and get tokens to other ReST APIs all within the client JavaScript code. There are a few important security considerations to take into account when using the implicit flow, specifically around client and user impersonation.

The OAuth2 specification declares that the Implicit Grant flow has been devised to support user-agent applications, meaning JavaScript applications, such as ADF apps, executing within a browser. The defining characteristic of such applications is that JavaScript code is used for accessing server resources (typically via a ReST API) and for updating the application user experience accordingly.

Here we can think of scenarios in our ADF based applications, such as Alfresco Digital Workspace (ADW) and the ones you write yourself for your Alfresco project: when you for example navigate around in the Document Library component, only the central panel changes to display the new folder selection, while the rest of the page remains unmodified. This characteristic is in contrast with traditional redirect-based Web apps, where every user interaction results in a full page postback and a full page rendering of the new server response.

Applications that take the JavaScript based approach to its extreme are called Single-page web apps, or SPAs. The idea is that these applications only serve an initial HTML page and associated JavaScript, with all subsequent interactions being driven by ReST API calls performed via JavaScript.

Redirect-based applications typically secure their requests via cookies, however, that approach does not work as well for JavaScript applications. Cookies only work against the domain they have been generated for, while JavaScript calls might be directed toward other domains.

A growing trend for JavaScript applications is to have no backend at all, relying 100% on third party ReST APIs to implement their business function. This is pretty much how our ADF based apps work.

Currently, the preferred method of protecting calls to a ReST API is to use the OAuth2 *bearer token* approach, where every call is accompanied by an OAuth2 access token. The ReST API examines the incoming access token and, if it finds in it the necessary scopes, it grants access to the requested operation. The Implicit Grant flow provides a convenient mechanism for JavaScript applications to obtain access tokens for a ReST API.

To understand the Implicit Grant flow we need to talk about the following roles involved in the OpenID Connect authentication flow:

- **Resource Owner**: The resource owner is the person or application that owns the data that is to be shared. For instance, the end user accessing Alfresco Content Services (ACS) could be a resource owner. The resource that it owns is its Repository content (i.e. folders and files).
- **Resource Server**: The resource server is the server hosting the protected resources. For instance, an ACS server is a resource server. It is also capable of dealing with access tokens in protected resource requests.
- **Client**: The client application is the application requesting access to the protected resources on behalf of the resource owner. A client application could be an ADF application requesting access to an end user's ACS repository content.
- **Authorization Server**: The authorization server is the server authorizing the client app to access the protected resources on behalf of the resource owner. It issues access tokens to the client. In our case this is the Alfresco Identity Service, which is built on Keycloak.

Tokens are random strings generated by the authorization server (i.e. Identity Service/keycloak) and are issued when the client (e.g. ADF, APS) requests them.

There are 2 types of tokens:

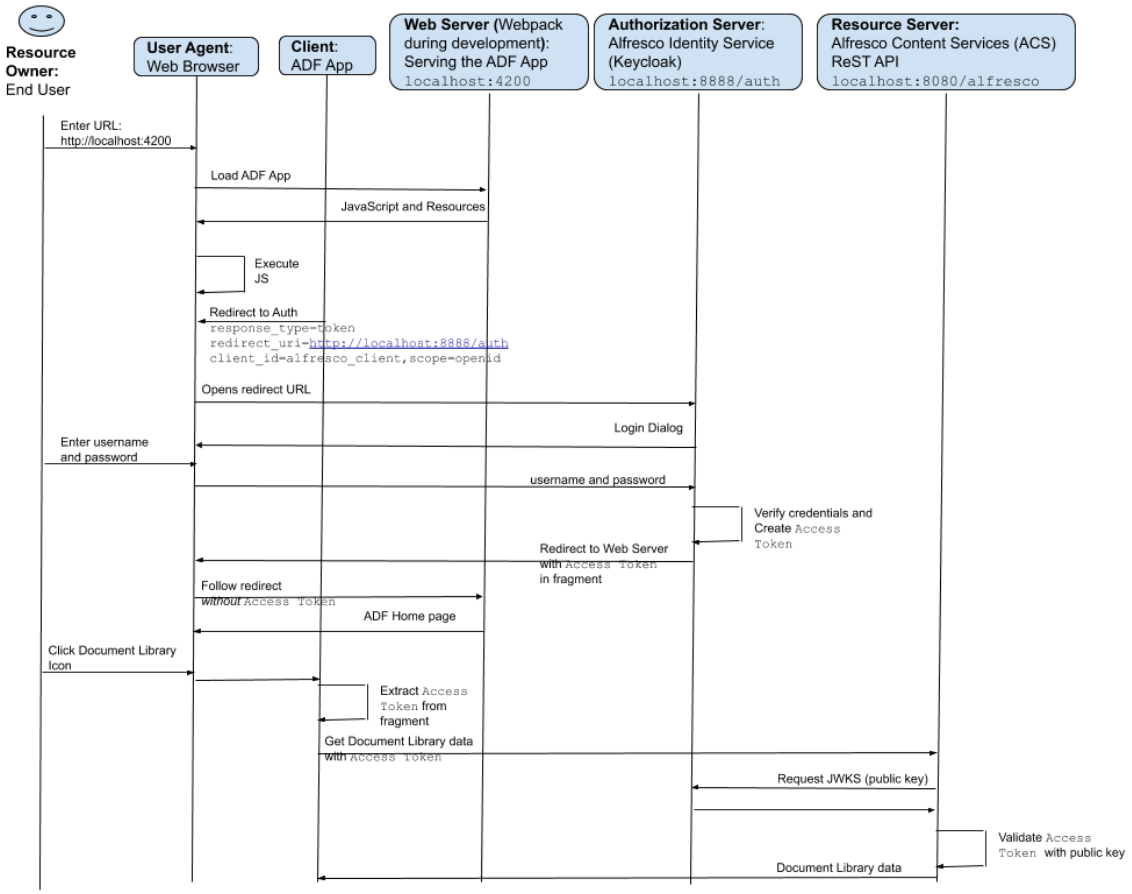
- **Access Token**: this is the most important one as it allows the user data (folders, files, processes) to be accessed by a third-party client application (e.g. ADF). This token is sent by the client as a parameter or as a header in the request to the resource server. It has a limited lifetime, which is defined by the authorization server. It must be kept confidential as much as possible but we will see that this is not always possible, especially when the client is a web browser that sends requests to the resource server via Javascript.
- **Refresh Token**: this token is issued with the access token but unlike the latter, it is not sent in each request from the client to the resource server. It merely serves to be sent to the authorization server for renewing the access token when it has expired. For security reasons, it is not always possible to obtain this token.

By default, access tokens are opaque text strings that serve only to allow the resource server (e.g.

(JSON) formatted structured alternative. These tokens, known as JSON Web Tokens (JWT), will provide for the ReSTful API world a security token comparable to SAML assertions in SOAP Web Services.

The access token represents the credentials to access protected resources. So your application (i.e. ADF App) does not need to store the resource owner credentials, but instead just the access token. The whole idea with OAuth2 is to create an authentication and authorization system that is not reliant on a username/password combination being broadcast and to instead rely upon a token exchange mechanism built upon digital signature technology.

The following picture illustrates *Implicit Grant flow* in the Alfresco DBP world:



By looking at the flows in the above picture we can see that the client (i.e. the ADF App) needs to know about the location of the authorization server and what redirection URL to use. Each resource server (i.e. ACS and APS) needs to also know the location of the authorization server so it can verify the access token being used to access a protected resource. The authorization server needs to be hooked up to a database or a directory with user credentials data.

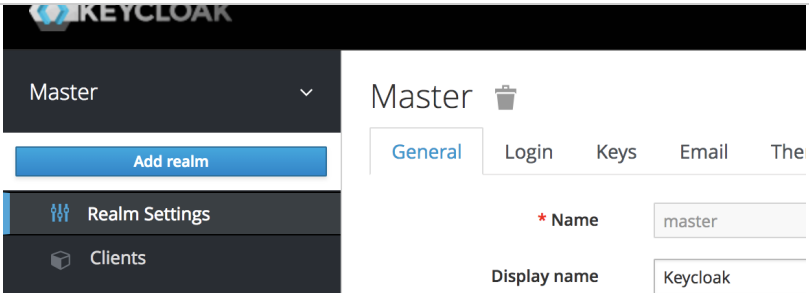
It is also common to create a separate security realm for the solution that you are implementing authentication for. There is only the master realm by default with the admin user, and we don't want to use this realm for our DBP solution's users, groups, roles etc. The master realm is a place for super admins to create and manage the realms in your system.

Creating a DBP Realm

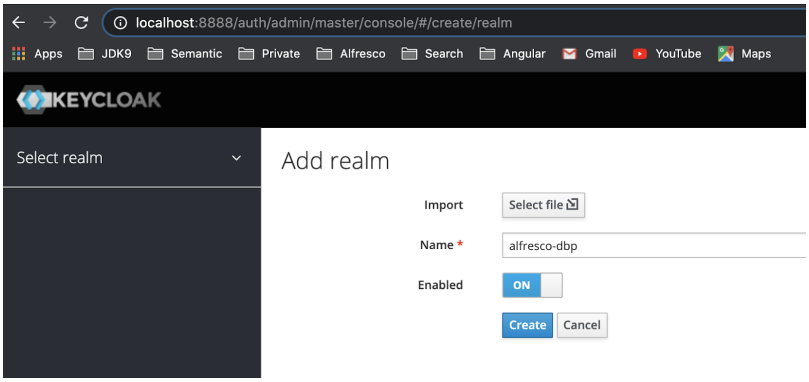
If you have followed along you should now be able to access the Keycloak Admin Console with your admin account by visiting the following URL: http://localhost:8888/auth/admin. By default, you'll be inside the Master realm.

OK, but what is a realm, really? It's just a domain in which you apply specific security policies. The Master realm is the parent of any realm you create. For my purpose, I want to create a new realm, which will be a new security domain specifically for the Alfresco Digital Business Platform (DBP) solution.

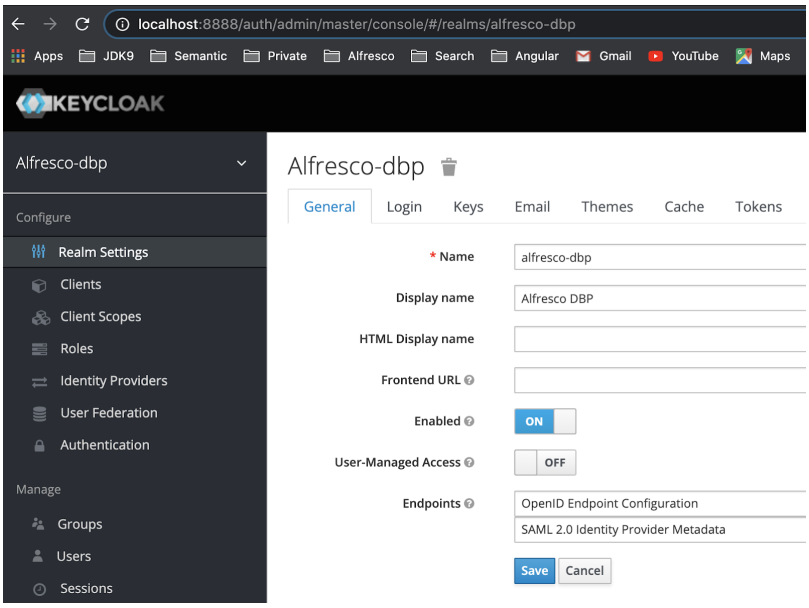
On the top left of the Admin Console, click the little arrow next to Master. It'll appear a drop-down menu where Keycloak shows all your realms and let you select any of them:



Click **Add realm** and then fill in the name of the new realm, in this case we call it **alfresco-dbp** (note. avoid space in the realm name as it can cause problems):



Then click **Create** button, which will then lead to this screen:



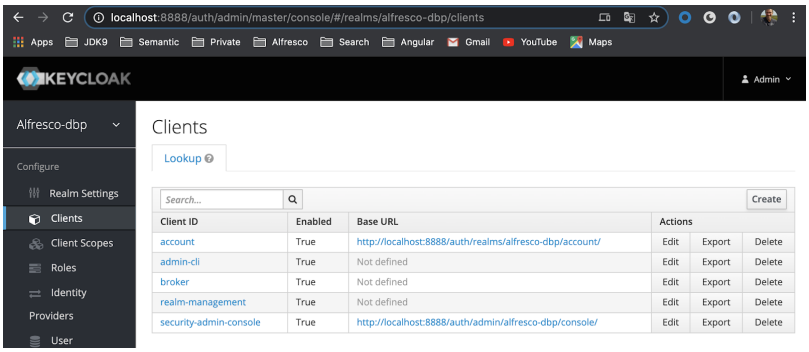
Fill in the *Display name* and click the **Save** button.

For the moment you can keep all the default configurations for the realm. We can get a quick summary of the new realm by hitting the following URL:
http://localhost:8888/auth/realms/alfresco-dbp:

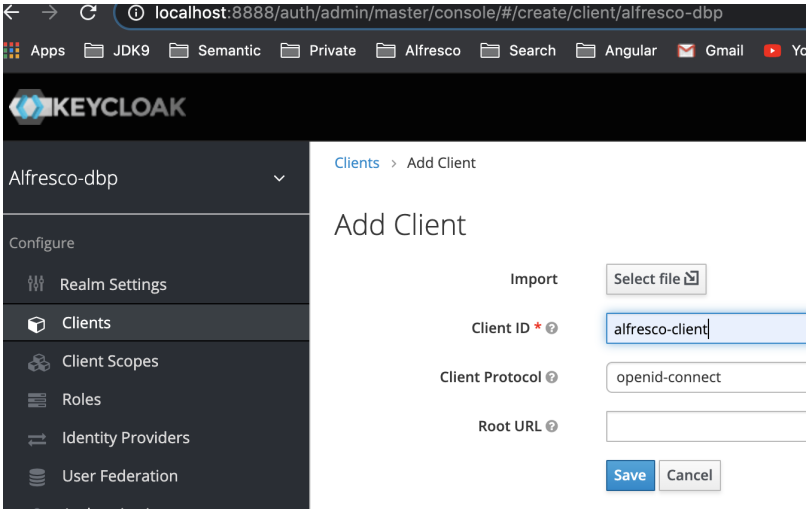

```
{
	realm: "alfresco-dbp",
	public_key:
	"MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAoD4Ulo90861IBumQOzFID/uQGGwuaDnS4on3q2GY9k5BXa0SnxBqmTzcW9neLf6jUqQ0d0b2PskTFfmTrT
	token-service: "http://localhost:8888/auth/realms/alfresco-dbp/protocol/openid-connect",
	account-service: "http://localhost:8888/auth/realms/alfresco-dbp/account",
	tokens-not-before: 0
}
```


Here we can see that it will use the OpenID Connect protocol.

To create a new client, go to **Clients:**

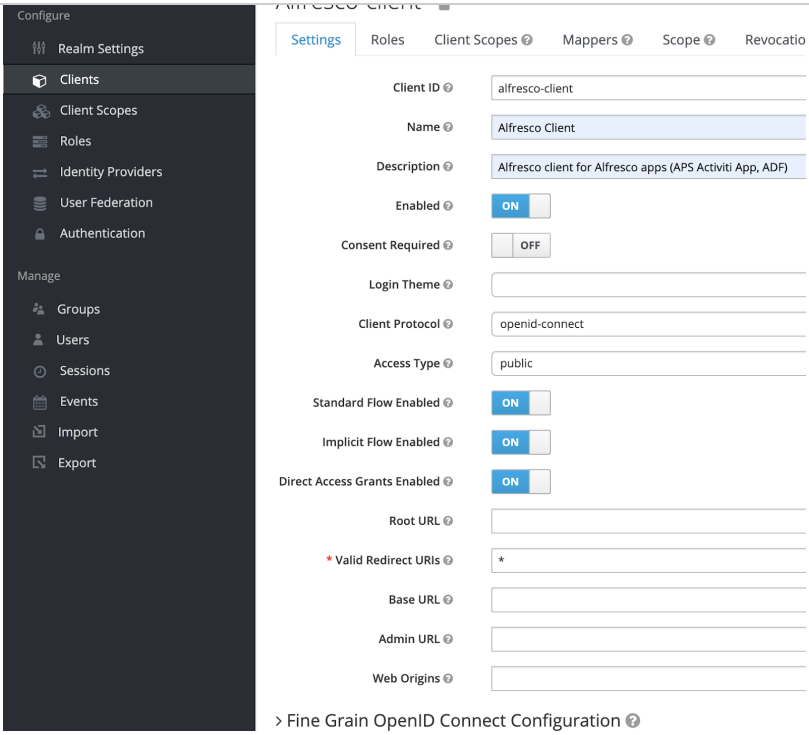


Then click the **Create** button to the right:



Set the Client ID to **alfresco-client**. We will use one Client for ADF apps, ACS, and APS. We want to use the OpenID Connect (OIDC) protocol, which is selected by default. Click **Save** to create this new client.

Once you create the client, a new page opens with further fields to configure it:



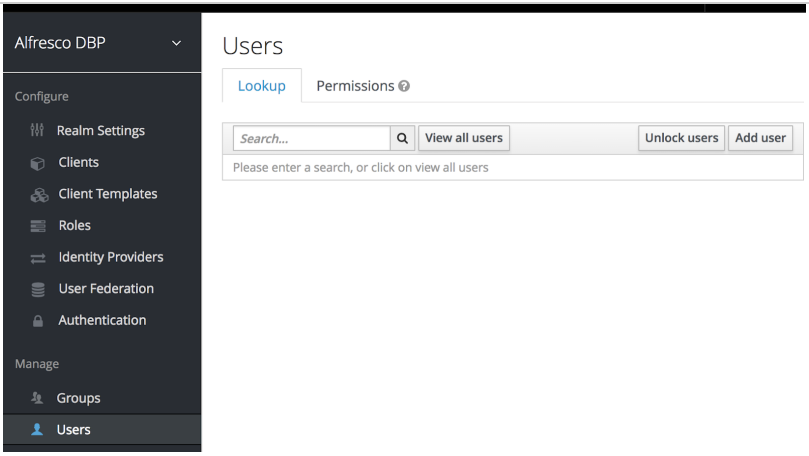
Make sure you set the following options in this way:

- `Enabled`: **ON** - allows turning on and off the client for requesting authentication.
- `Access Type`: **public** - The Access Type option defines the type of the client. As explained in the documentation, “Public access type is for client-side clients that need to perform a browser login”, that is exactly what we want.
- `Standard Flow Enabled`: **ON** - is used to activate the Authorization Code Flow as defined in the OpenID Connect (OIDC) standard. It’s the recommended protocol to use for authenticating and authorising browser-based applications. With this flow the Keycloak server returns *an authorization code, not an authentication token*, to the application. The client then exchanges the `code` for an access token and a refresh token after the browser is redirected back to the application. **This flow is used by the APS Activiti App UI**.
- `Implicit Flow Enabled`: **ON** - is used to activate the Implicit Grant flow where an access token is sent immediately after successful authentication with Keycloak. **This flow is used by ADF applications.**
- `Direct Access Grant Enabled`: **ON** - is used to activate the OAuth2 Password Grant flow, which is used by command line tools like **curl when testing ACS ReST** API. It can also be used by **ADF applications**.
- `Valid Redirect URIs`: ***** - configure it to the base URI for your Alfresco components, for example:
  - Any URL: *****
  - A local development environment: **http://localhost:***
  - A cloud or product test env: **http://mydomain:80**

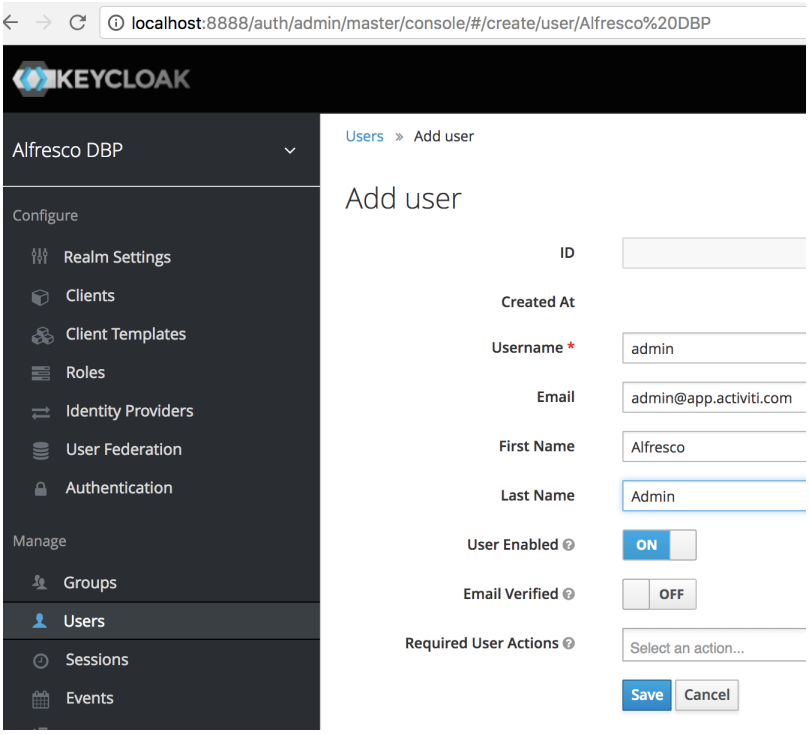
Click the **Save** button when finished with the Client configuration.

Create a User to Validate the Environment

To validate our environment (and for APS Admin capabilities), it’s recommended to create a local user in Keycloak. Click on the **Users** menu item to the left:



Then click the **Add user** button:

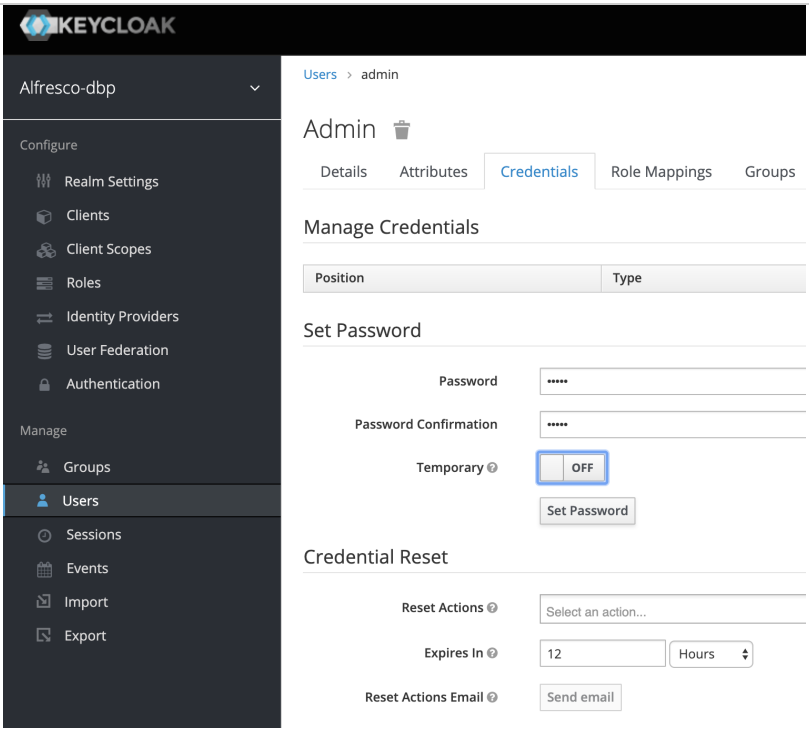


Fill in the following fields:

- `Username`: **admin** - this username need to match the Administrator username in ACS *(i.e. the user we create here need to exist in ACS)*
- `Email`: **admin@app.activiti.com** - this email address need to match the username in APS *(i.e. the user we create here need to exist in APS)*
- `First Name` and `Last Name`: set to whatever you like

Click the **Save** button to add the user to the Alfresco DBP realm.

Go to the **Credentials** tab and insert a new password (I am using **admin** as password):



For this blog post I don't want to require the user to change the password on their first login, so I'll toggle the **Temporary** option to **OFF**. After that, click **Set password**.

> In a production environment, it would be a good idea enabling this option when you define a password on behalf of some user.

We are now ready to start configuration of the DBP components (i.e. ACS, APS, ADF).

Deploying and Running Alfresco DBP

To deploy the Alfresco DBP we will use another Docker Compose file. It will deploy ACS 6.2.1, APS 1.11, PostgreSQL database, and other supporting services.

DBP Docker Compose file

The Docker Compose file that we will be using for the DBP looks as follows (see **my-dbp-identity-service/docker-compose-dbp/docker-compose.yml**):

```
version: "2"

services:

#########################
###   Alfresco Content Services (ACS)
#########################################################################################
  alfresco:
      image: alfresco/alfresco-content-repository:6.2.1
      environment:
          JAVA_OPTS: "
              -Ddb.driver=org.postgresql.Driver
              -Ddb.username=alfresco
              -Ddb.password=alfresco
              -Ddb.url=jdbc:postgresql://postgres:5432/acs
              -Dsolr.host=solr6
              -Dsolr.port=8983
              -Dsolr.secureComms=none
              -Dsolr.base.url=/solr
              -Dindex.subsystem.name=solr6
              -Dshare.host=127.0.0.1
              -Dshare.port=8080
              -Dalfresco.host=localhost
              -Dalfresco.port=8080
              -Daos.baseUrlOverwrite=http://localhost:8080/alfresco/aos
              -Dmessaging.broker.url=\"failover:(nio://activemq:61616)?
timeout=3000&jms.useCompression=true\"
```

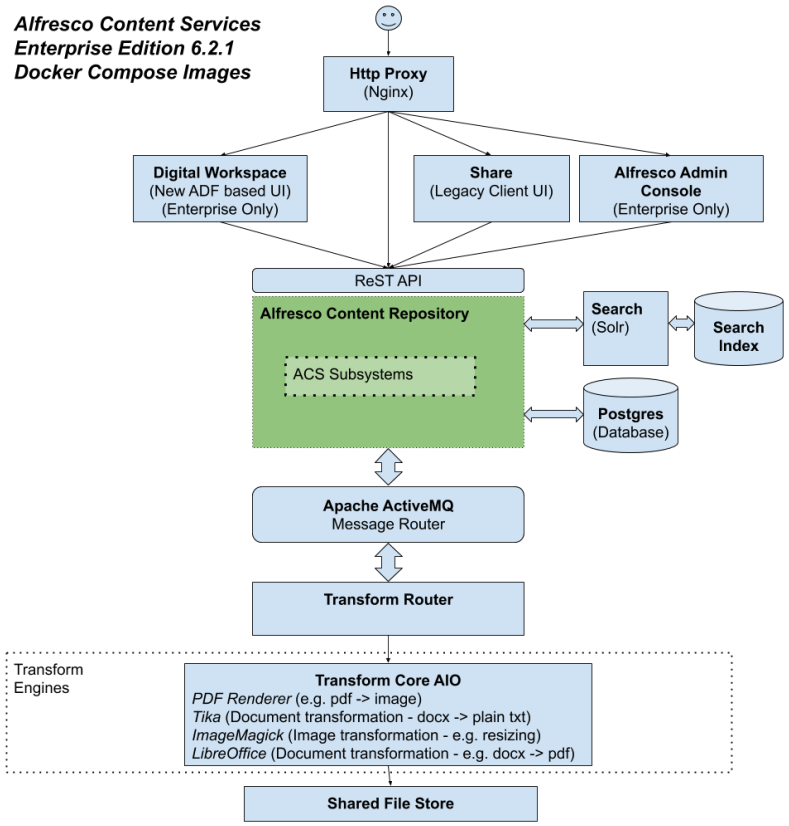
```
                -Dtransform.service.url=http://transform-router:8095
                -DlocalTransform.core-aio.url=http://transform-core-aio:8090/
                -Dalfresco-pdf-renderer.url=http://transform-core-aio:8090/
                -Djodconverter.url=http://transform-core-aio:8090/
                -Dimg.url=http://transform-core-aio:8090/
                -Dtika.url=http://transform-core-aio:8090/
                -Dtransform.misc.url=http://transform-core-aio:8090/
                -Dsfs.url=http://shared-file-store:8099/
                -Dcsrf.filter.enabled=false
                -Ddsync.service.uris=http://localhost:9090/alfresco
                -Xms1500m
                -Xmx1500m
                "
        volumes:
          - ./data/acs:/usr/local/tomcat/alf_data/
#           - ./acs/alfresco-global.properties:/usr/local/tomcat/shared/classes/alfresco-
global.properties
#           - ./acs/log4j.properties:/usr/local/tomcat/webapps/alfresco/WEB-INF/classes/log4j.properties
          - ./acs/license:/usr/local/tomcat/shared/classes/alfresco/extension/license
        depends_on:
          - solr6
        ports:
          - 8082:8080 # Need a separate port mapping for this so we can talk directly from inside APS

  transform-router:
        image: quay.io/alfresco/alfresco-transform-router:1.2.0
        environment:
          JAVA_OPTS: " -Xms256m -Xmx512m"
          ACTIVEMQ_URL: "nio://activemq:61616"
          CORE_AIO_URL : "http://transform-core-aio:8090"
          FILE_STORE_URL: "http://shared-file-store:8099/alfresco/api/-
default-/private/sfs/versions/1/file"
        ports:
            - 8095:8095
        links:
          - activemq

  transform-core-aio:
        image: alfresco/alfresco-transform-core-aio:2.2.1
        environment:
            JAVA_OPTS: " -Xms256m -Xmx1536m"
            ACTIVEMQ_URL: "nio://activemq:61616"
            FILE_STORE_URL: "http://shared-file-store:8099/alfresco/api/-
default-/private/sfs/versions/1/file"
        ports:
            - 8090:8090
        links:
        - activemq

  shared-file-store:
        image: alfresco/alfresco-shared-file-store:0.7.0
        environment:
            JAVA_OPTS: " -Xms256m -Xmx512m"
            scheduler.content.age.millis: 86400000
            scheduler.cleanup.interval: 86400000
        ports:
            - 8099:8099
        volumes:
            - shared-file-store-volume:/tmp/Alfresco/sfs

  share:
        image: alfresco/alfresco-share:6.2.1
        environment:
            REPO_HOST: "alfresco"
            REPO_PORT: "8080"
            JAVA_OPTS: "
                -Xms500m
                -Xmx500m
                -Dalfresco.host=localhost
                -Dalfresco.port=8080
                -Dalfresco.context=alfresco
                -Dalfresco.protocol=http
                "

  solr6:
        image: alfresco/alfresco-search-services:1.4.2
        environment:
            #Solr needs to know how to register itself with Alfresco
            - SOLR_ALFRESCO_HOST=alfresco
            - SOLR_ALFRESCO_PORT=8080
            #Alfresco needs to know how to call solr
            - SOLR_SOLR_HOST=solr6
            - SOLR_SOLR_PORT=8983
            #Create the default alfresco and archive cores
            - SOLR_CREATE_ALFRESCO_DEFAULTS=alfresco,archive
            #HTTP by default
```

```
    image: alfresco/alfresco-activemq:5.15.8
    ports:
      - 8161:8161 # Web Console
      - 5672:5672 # AMQP
      - 61616:61616 # OpenWire
      - 61613:61613 # STOMP

  digital-workspace:
    image: quay.io/alfresco/alfresco-digital-workspace:1.5.0
    mem_limit: 128m
    environment:
      BASE_PATH: ./

  proxy:
    image: alfresco/alfresco-acs-nginx:3.0.1
    depends_on:
      - alfresco
      - digital-workspace
    ports:
      - 8080:8080
    links:
      - digital-workspace
      - alfresco
      - share

  sync-service:
    image: quay.io/alfresco/service-sync:3.3.2
    environment:
      JAVA_OPTS : "
      -Dsql.db.driver=org.postgresql.Driver
      -Dsql.db.url=jdbc:postgresql://postgres:5432/acs
      -Dsql.db.username=alfresco
      -Dsql.db.password=alfresco
      -Dmessaging.broker.host=activemq
      -Drepo.hostname=alfresco
      -Drepo.port=8080
      -Ddw.server.applicationConnectors[0].type=http
      -Xms1000m -Xmx1000m
      "
    ports:
      - 9090:9090

#########################
###   Alfresco Process Services (APS)
#########################################################################################

  process:
    image: alfresco/process-services:1.11.0
    environment:
      CATALINA_OPTS: "-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=5005"
      ACTIVITI_DATASOURCE_USERNAME: alfresco
      ACTIVITI_DATASOURCE_PASSWORD: alfresco
      ACTIVITI_DATASOURCE_DRIVER: org.postgresql.Driver
      ACTIVITI_HIBERNATE_DIALECT: org.hibernate.dialect.PostgreSQLDialect
      ACTIVITI_DATASOURCE_URL: 'jdbc:postgresql://postgres:5432/aps?characterEncoding=UTF-8'
      ACTIVITI_CSRF_DISABLED: 'true'
      ACTIVITI_CORS_ENABLED: 'true'
      ACTIVITI_ES_SERVER_TYPE: client
      ACTIVITI_ES_DISCOVERY_HOSTS: elasticsearch:9300
      ACTIVITI_ES_CLUSTER_NAME: elasticsearch
    volumes:
      - ./aps/enterprise-license:/root/.activiti/enterprise-license/:ro
      - ./aps/transform.lic:/usr/share/tomcat/lib/transform.lic
      - ./aps/activiti-app.properties:/usr/share/tomcat/lib/activiti-app.properties
#       - ./aps/activiti-identity-service.properties:/usr/local/tomcat/lib/activiti-identity-
service.properties
#       - ./aps/activiti-ldap.properties:/usr/local/tomcat/lib/activiti-ldap.properties
      - ./aps/log4j.properties:/usr/share/tomcat/webapps/activiti-app/WEB-
INF/classes/log4j.properties
      - ./data/aps:/usr/local/data/
    ports:
      - 9080:8080 # Browser port
      - 5006:5005 # Remote Debug
    depends_on:
      - elasticsearch

  elasticsearch:
    image: elasticsearch:1.7.3

# Docker Volumes for ACS
volumes:
   shared-file-store-volume:
    driver_opts:
      type: tmpfs
      device: tmpfs
```

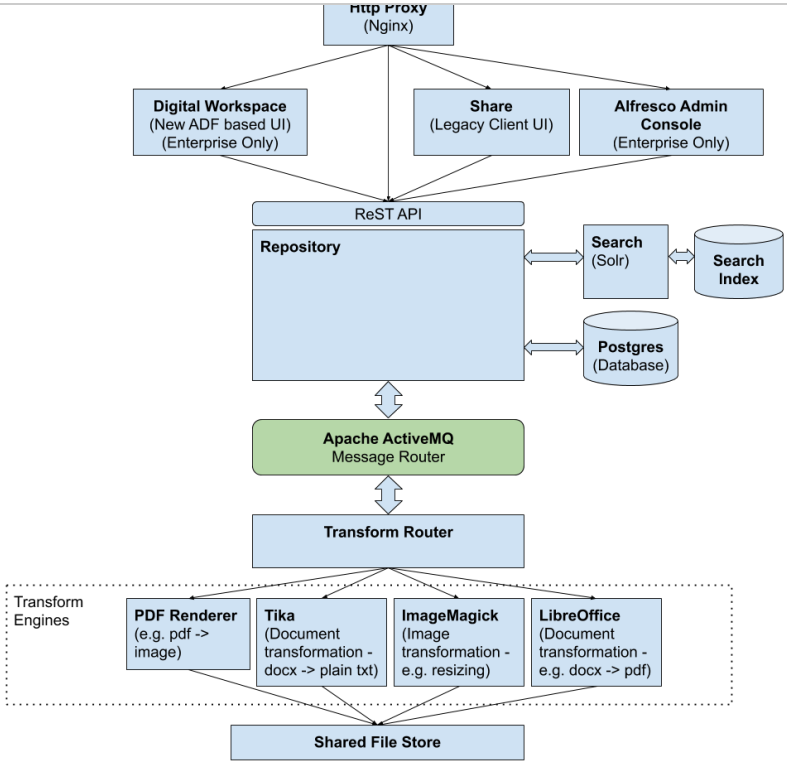
The Alfresco Repository and the Alfresco Process services will use databases created when the Identity Service Docker Compose file was deployed. We can reach the `postgres` service deployed with that Docker Compose file directly with `postgres:5432`. The reason this works is because we define the default Docker Network for the DBP Docker Compose file as an external network called `docker-compose-identity_default`, which is the network created by the Identity Service Docker Compose file.

In this DBP Deployment we use the new ACS 6.2.1 release, which comes with some nice improvements for the Alfresco Transform Service. We no longer need to deploy separate services/containers for each Transform Engine (i.e. imagemagic, alfresco-pdf-renderer, libreoffice, tika etc), instead we can deploy one service to handle all. This service is called `transform-core-aio`, where aio stands for All-In-One.

Here is an overview of the ACS 6.2.1 architecture:



To compare, the following picture give you an overview of the 6.2.0 ACS architecture:



After defining the ACS and Transformation related services we define the APS related services, which are `process` and `elasticsearch`. You can see that the Alfresco Digital Workspace UI client is also included (`digital-workspace`), so we can see later how that client looks like, it is the new Alfresco UI based on ADF and as time goes it will replace more and more of the Share functionality, currently it just supports content.

Now, there are a lot of volumes configured for these services. They are supported by the following directories and files in the **base-adbp-project/docker-compose-dbp** directory (see source code):

```
.
├── acs (Contains custom configuration for ACS)
│   ├── license (put enterprise ACS license here)
│   ├── alfresco-global.properties (main config file)
│   └── log4j.properties
├── aps (Contains custom configuration for APS)
│   ├── enterprise-license             (put enterprise APS license here)
│   ├── activiti-app.properties              (main config file)
│   ├── activiti-identity-service.properties (Identity Service/Keycloak)
│   ├── activiti-ldap.properties             (LDAP auth config)
│   ├── log4j.properties
│   └── transform.lic
├── data (Created when you deploy and run)
│   ├── acs  (repository content files)
│   ├── aps  (process content)
│   └── solr (index files)
└── docker-compose.yml
```

As you can see, there are no ACS or APS licenses available in the example project. You have to sign up for trials to get them.

For ACS you will not actually get a license file but instead a login to Quay. Some of the Alfresco images we will be using are Enterprise images that are hosted in the Quay Docker Image Repository. If you don't have access to them, then you need to sign up for an ACS Trial to get temporary access. However, most of the images are available on Docker Hub, which doesn't require a login.

To set up access to Alfresco enterprise docker images do a Docker Login as follows with the information from your "Start Your Trial Now" email:

```
$ docker login quay.io -u="alfresco+acs_v6_trial"

Enter the following password when prompted (Copy and Paste):
<YOUR SECRET/PASSWORD FROM THE EMAIL>
```

When you sign up for an APS trial you will get an email where you can download a license file (**activiti.lic**). Put this file in the **my-dbp-identity-service/docker-compose-dbp/aps/enterprise-license** directory.

```
docker-compose-dbp mbergljung$ docker-compose up
Creating docker-compose-dbp_digital-workspace_1 ... done
Creating docker-compose-dbp_activemq_1          ... done
Creating docker-compose-dbp_sync-service_1      ... done
Creating docker-compose-dbp_elasticsearch_1     ... done
Creating docker-compose-dbp_share_1             ... done
Creating docker-compose-dbp_solr6_1             ... done
Creating docker-compose-dbp_shared-file-store_1 ... done
Creating docker-compose-dbp_process_1            ... done
Creating docker-compose-dbp_alfresco_1           ... done
Creating docker-compose-dbp_transform-router_1   ... done
Creating docker-compose-dbp_transform-core-aio_1 ... done
Creating docker-compose-dbp_proxy_1              ... done
...
```

You should see APS version and license info in the logs:

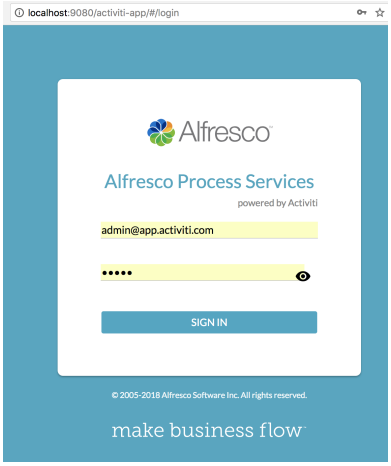
```
process_1               | --------------ProcessEngineConfiguration initialized --------------------------
process_1               | edition = Alfresco Process Services (powered by Activiti)
process_1               | version = 1.11.0
process_1               | ------------------------------------------------------------------------------
process_1               |
process_1               | 10:30:45 [localhost-startStop-1] INFO  com.activiti.license.LicenseHolder  - Using
custom license loader to load license
process_1               | 10:30:47 [localhost-startStop-1] INFO  com.activiti.license.LicenseHolder  - Trying to
load license from user home /root/.activiti/enterprise-license/activiti.lic
process_1               | 10:30:47 [localhost-startStop-1] INFO
 org.activiti.engine.impl.cfg.ProcessEngineConfigurationImpl  - License found:
process_1               |
process_1               | ----------------------------------------
process_1               | Holder:AlfrescoInternalSupport
process_1               | Product Key:1.0ent
process_1               | Good After date:20130713
process_1               | Good Before date:20210816
process_1               | Multi tenant?:false
process_1               | Default tenant name:alfresco.com
process_1               | ----------------------------------------
...
```

Then you should see ACS license and version info:

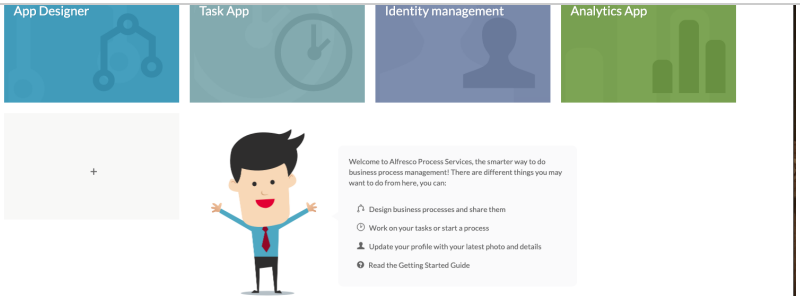
```
alfresco_1              | 2020-05-21 10:32:53,378  INFO  [service.descriptor.DescriptorService]
[localhost-startStop-1] Alfresco license: Mode ENTERPRISE, cluster:enabled granted to Trial User
limited to 30 days expiring Sat Jun 20 00:00:00 UTC 2020 (30 days remaining).
alfresco_1              | 2020-05-21 10:32:53,379  INFO  [service.descriptor.DescriptorService]
[localhost-startStop-1] Server Mode :UNKNOWN
alfresco_1              | 2020-05-21 10:32:53,387  INFO  [service.descriptor.DescriptorService]
[localhost-startStop-1] Alfresco Content Services started (Enterprise). Current version: 6.2.0 (2
r68ee468c-b10) schema 13,001. Originally installed version: 6.2.0 (2 r68ee468c-b10) schema 13,001.
...
alfresco_1              | 21-May-2020 10:33:48.849 INFO [main]
org.apache.coyote.AbstractProtocol.start Starting ProtocolHandler ["http-nio-8080"]
alfresco_1              | 21-May-2020 10:33:48.934 INFO [main]
org.apache.coyote.AbstractProtocol.start Starting ProtocolHandler ["ajp-nio-8009"]
alfresco_1              | 21-May-2020 10:33:49.063 INFO [main]
org.apache.catalina.startup.Catalina.start Server startup in 460722 ms
```

Wait for a couple of minutes until it has all started, it can take quite a bit of time the first time you do this as all the Docker Images need to be downloaded from different places.

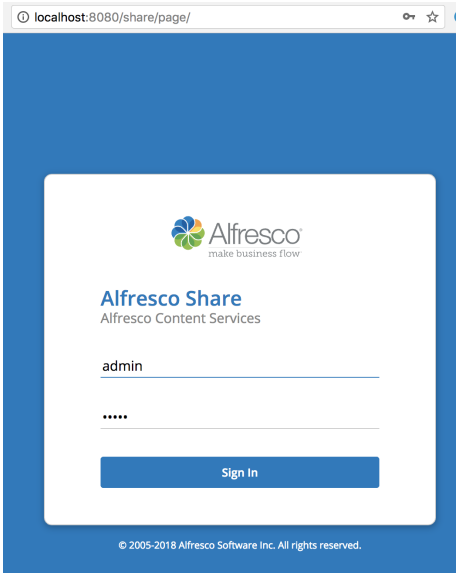
Then verify that it's all working by accessing all the apps. Start with APS via http://localhost:9080/activiti-app and **admin@app.activiti.com/admin** credentials:



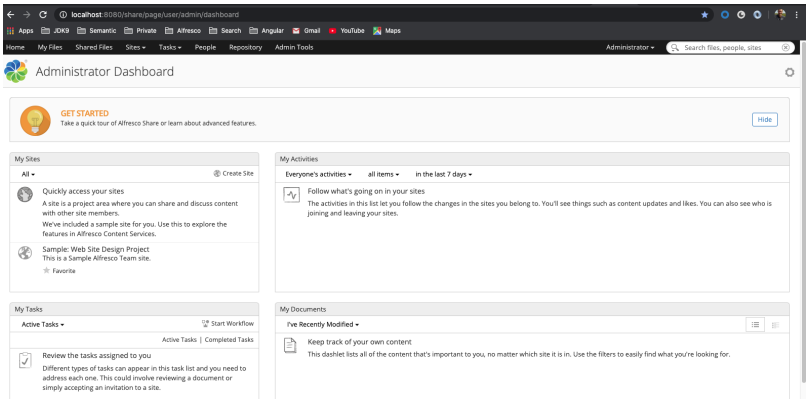
After successful login you should see:



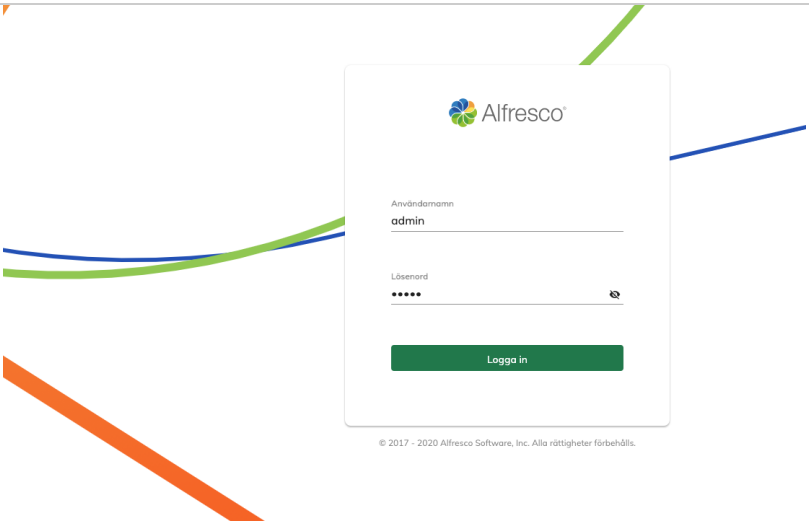
Then try ACS at http://localhost:8080/share with **admin/admin**:



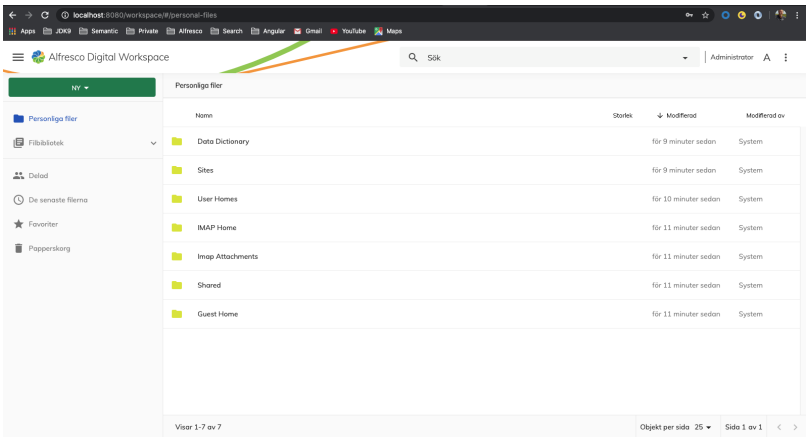
You should see something like this:



Let's also try the new content UI called Alfresco Digital Workspace at http://localhost:8080/workspace, login with **admin/admin**:



After a successful login we should see the following screen:



Configure APS to use the Identity Service

The standard APS User Interface (i.e. **/activiti-app**) can participate in OpenID Connect authentication. It will act as the OAuth2 Client.

Identity Service Configuration

Open up the **docker-compose-dbp/aps/activiti-identity-service.properties** file in the source code project and make the following changes:

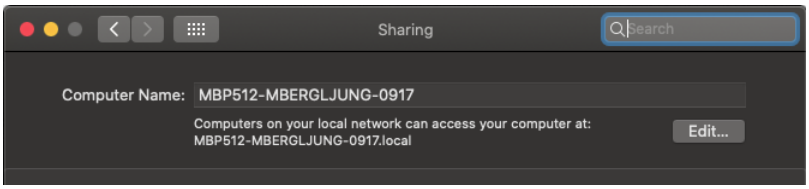
```
keycloak.enabled=true
keycloak.ssl-required=none
keycloak.resource=alfresco-client
keycloak.realm=alfresco-dbp
keycloak.auth-server-url=http://<Host/domain name that works inside and outside APS
container>:8888/auth
# Example URL on article author's Mac
#keycloak.auth-server-url=http://mbp512-mbergljung-0917.local:8888/auth
# Example URL on article author's Windows 10
#keycloak.auth-server-url=http://host.docker.internal:8888/auth
keycloak.principal-attribute=email
keycloak.public-client=true
keycloak.always-refresh-token=true
keycloak.autodetect-bearer-only=true
keycloak.token-store=cookie
keycloak.enable-basic-auth=true
```

If you used the same Alfresco realm name as the article (i.e. `alfresco-dbp`) and the same Alfresco client name (i.e. `alfresco-client`), then you only have to change the `keycloak.auth-server-url` value. This URL needs to have a host/domain name that will work from both inside the APS container and from the outside of the APS container (e.g. from the Browser). See below for more info on how to find this name.

Property Name	Value	Description
keycloak.enabled	**true**	`true` = turns on SSO via Keycloak. `false` = APS will not be involved in OAuth2 authentication.
keycloak.realm	**alfresco-dbp**	Which keycloak security realm should be used (i.e. where are the Alfresco users etc?)
keycloak.auth-server-url	**http://mbp512-mbergljung-0917.local:8888/auth**	Where is the Keycloak server running? Used to construct two URLs: 1) For the browser redirect to keycloak server - this will use the external port for the identity service container (i.e. 8888) 2) For the APS backend communication with keycloak when verifying access tokens. - this will use the internal port for the identity service container (i.e. 8888) Which means that this URL has to work both inside and outside the APS container. It does that by using the Mac *hostname (in lowercase).local* and the external port 8888 (the internal port is the same so works both externally and internally). And this port cannot clash with the Alfresco nginx proxy container used in the DBP deployment. On Windows you can use a special hostname called *host.docker.internal* to get around the problems. This URL is easier to get working properly when the whole solution is deployed with proper DNS names set up, such as for example **identity.acme.com**. Then you just use the DNS name for all URL configurations, whether it's from inside a container or from the outside.
keycloak.ssl-required	none	If set to `all`, then it ensures that all communication to and from the Keycloak server is over HTTPS.
keycloak.resource	**alfresco-client**	The OAuth2 client ID. Each application has a client-id that is used to identify the application. We use the same client ID for all Alfresco clients (i.e. APS web client, ADF web client).
keycloak.principal-attribute	email	The primary user id in APS is email address. So we use the email field in the keycloak user account to authenticate. This is the OpenID Connection ID Token attribute to populate the `UserPrincipal` name with. If the token attribute is `null`, then it defaults to `sub`. Possible values are `sub`, `preferred_username`, `email`, `name`, `nickname`, `given_name`, `family_name`.
keycloak.public-client	true	If set to `true`, then the APS Activiti App web client will not send credentials for the client to Keycloak.
keycloak.credentials.secret		The secret key for this client if the access type is not set to `public`.
keycloak.always-refresh-token	true	If `true`, then APS will refresh the token in every request.
keycloak.autodetect-bearer-only	true	This should be set to `true` if APS serves both a web application (e.g. Activiti App)

		unauthenticated ReST clients instead as they would not understand a redirect to the login page. Keycloak auto-detects ReST clients based on typical headers like `X-Requested-With`, `SOAPAction` or `Accept`. The default value is `false`.
`keycloak.token-store`	`cookie`	Possible values are `session` and `cookie`. Default is `session`, which means that APS stores token info in HTTP Session. Alternative `cookie` means storage of info in cookie.
`keycloak.enable-basic-auth`	`true`	This tells the adapter to also support basic authentication.

To find the Mac hostname that you should use go to **Apple menu > System Preferences**, then click **Sharing**, you should see something like this:



In this case APS is both an OAuth2 Client and an OAuth2 Resource Server.

The APS container needs to have this properties file mapped into it for it to be picked up. Uncomment the following line in **my-dbp-identity-service/docker-compose-dbp/docker-compose.yml** file for the `process` service:

```
#       - ./aps/activiti-identity-service.properties:/usr/local/tomcat/lib/activiti-identity-service.properties
```

Restart the APS container

To test the new Identity Service configuration we need to restart the APS container as follows from a different console window:

```
docker-compose-dbp mbergljung$ docker-compose stop process
Stopping docker-compose-dbp_process_1 ... done
docker-compose-dbp mbergljung$ docker-compose rm process
Going to remove docker-compose-dbp_process_1
Are you sure? [yN] y
Removing docker-compose-dbp_process_1 ... done
docker-compose-dbp mbergljung$ docker-compose up -d --no-deps process
Creating docker-compose-dbp_process_1 ... done
```

Note. you need to fully remove the container for these changes to take effect.

In the original console window where you started the DBP you should see the logs for the APS starting up again. Look for logs such as:

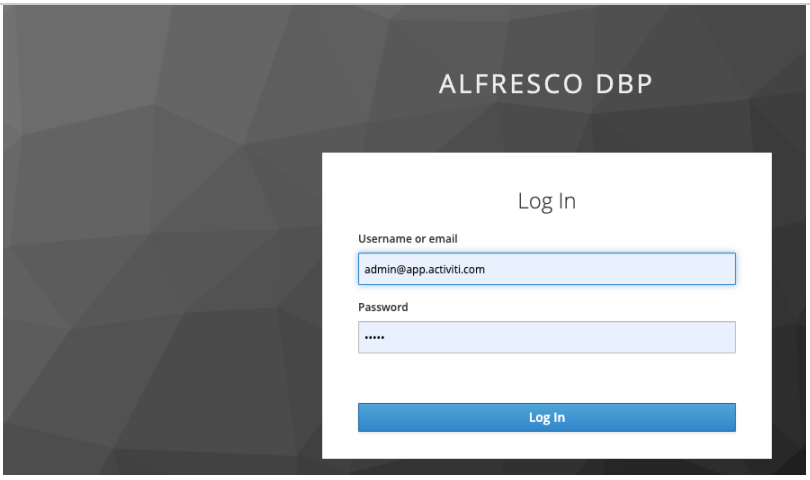
```
…
process_1                | 09:10:17 [localhost-startStop-1] INFO
 com.activiti.conf.SecurityConfiguration  - Auth - set Identity service as primary authentication
system
…
process_1                | 22-May-2020 09:10:54.906 INFO [main]
org.apache.coyote.AbstractProtocol.start Starting ProtocolHandler ["http-nio-8080"]
process_1                | 22-May-2020 09:10:54.959 INFO [main]
org.apache.coyote.AbstractProtocol.start Starting ProtocolHandler ["ajp-nio-8009"]
process_1                | 22-May-2020 09:10:54.988 INFO [main]
org.apache.catalina.startup.Catalina.start Server startup in 87310 ms
```

When we now go to http://localhost:9080/activiti-app we will be redirected to the Keycloak server for authentication with a URL looking something like:

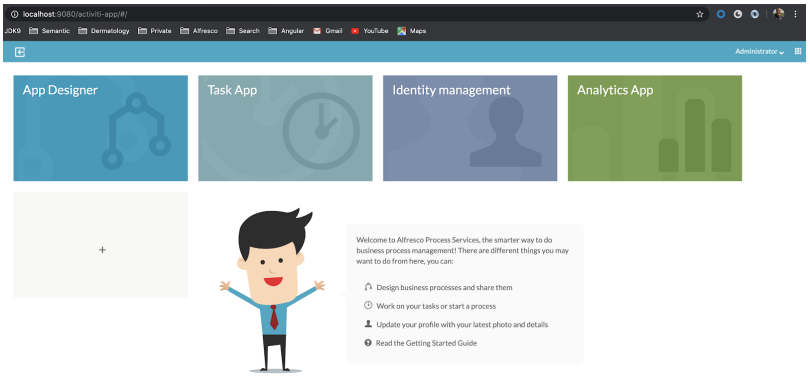
http://mbp512-mbergljung-0917.local:8888/auth/realms/alfresco-dbp/protocol/openid-connect/auth?response_type=code&client_id=alfresco-client&redirect_uri=http%3A%2F%2Flocalhost%3A9080%2Factiviti-app%2Fsso%2Flogin&state=629333c2-a405-4a77-bdef-d27f1d4c8c01&login=true&scope=openid

Note the `response_type=code`, which means that this uses the OAuth2 Authorization Code Grant flow.

The Keycloak login screen should now be visible as follows:



In this case we use the email **admin@app.activiti.com** as username and password **admin**. Just like we used when creating the user in the **alfresco-dbp** realm. Clicking the **Log in** button should successfully log us in and we should see the APS dashboard (and the APS login screen should never be visible):



We can logout from APS by clicking **Sign out** in the upper right corner menu. That should take us back to the Keycloak login screen.

Setting up an Alfresco Repository Endpoint in APS

The APS service can be configured to talk to the ACS service in two ways. The first way is via *Alfresco Repositories* configurations. These configurations are used by Alfresco Specific tasks. You can also configure ReST endpoints that target ACS, which can then be used by the ReST call task.

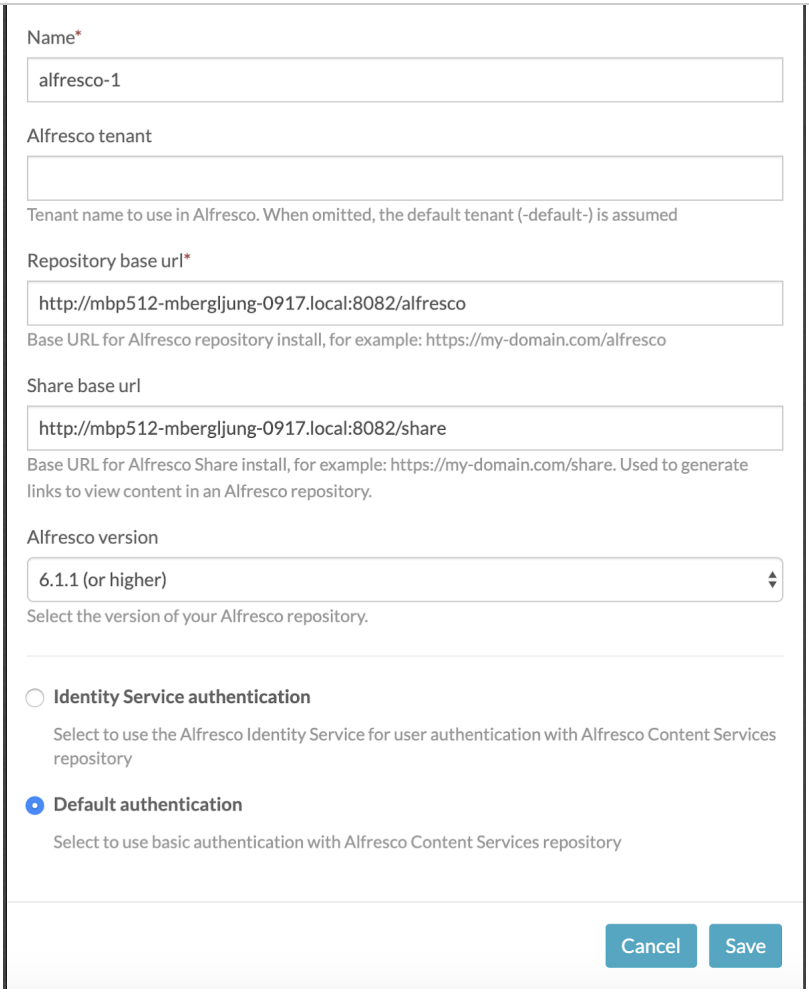
This might at first seem straightforward, just use **http://localhost:8080/alfresco**. But it's a bit tricky as communication takes place from inside the APS container to the ACS container, in a similar way to how we talk to the Identity Service from APS.

So we need to set up the URLs in a special way. And the ACS service needs to have an external port mapping set up, which it doesn't in the ACS Trial downloaded Docker Compose file.

The modified ACS Service definition in the Docker Compose file has the following port mapping:

```
Alfresco:
    ...
    ports:
      - 8082:8080
```

To set up an Alfresco Repository endpoint in APS navigate to **Identity management -> Tenants -> Alfresco Repositories**, then click the **+** to add a new one:



Note how we use a similar URL to the one we used for the Identity Service, it's just the port and URL path that is different:

http://mbp512-mbergljung-0917.local:8082/alfresco

You should try this URL in the Browser to make sure it works.

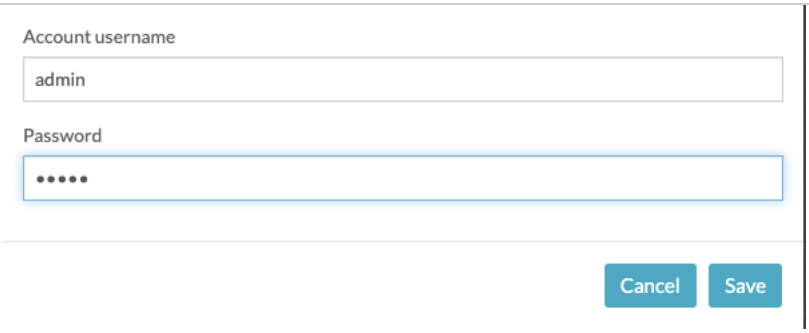
During development on Windows you might want to use the **host.docker.internal** hostname.

Also, select **Default authentication** so we can use Basic Auth with username/password.

Note. The reason we don't just use the internal ACS port 8080 is that this clashes with the Alfresco NGinx proxy public port mapping, which is 8080. So if we used http://mbp512-mbergljung-0917.local:8080/alfresco, APS would try to talk to the Alfresco Nginx Proxy, and this would not work. It does however work to use the internal default Docker network: http://alfresco:8080/alfresco where the hostname is the same as the Docker Compose file service name.

If your ACS service is available under a proper domain name, such as **acs.acme.com**, then everything becomes much more easier (clearer) as you would just use the domain name inside the container and outside the container.

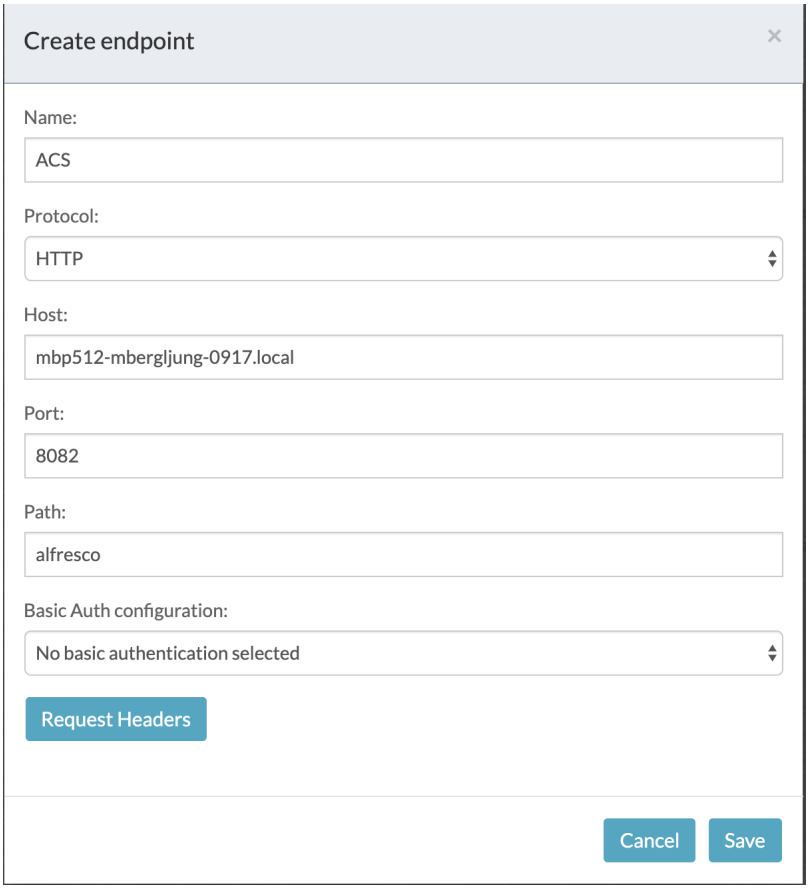
To verify that the configuration actually works we can set up authentication with username and password for the Alfresco Repository configuration. Navigate to **Identity management -> Personal -> click on alfresco-1 link** at the bottom of the screen:



In this case I use the **admin/admin** credentials as those are available by default in ACS. If everything works OK, then when you click **Save** the dialog should just close. If something is not configured correctly with the URL, then you will get an error and the dialog will not close.

Setting up an ACS ReST Endpoint in APS

Setting up a ReST endpoint that points to ACS is done in a similar way to how you configure Alfresco Repositories above:



For more info about configuring these ACS URLs see the previous section about configuring Alfresco Repositories.

Configure ACS to use the Identity Service

Now, let's see if we can configure ACS to use the Alfresco Identity Service (i.e. keycloak). Open up the **my-dbp-identity-service/docker-compose-dbp/acs/alfresco-global.properties** file and configure keycloak integration as follows:

```
# Authentication Subsystem Chain
authentication.chain=identity-service1:identity-service,alfrescoNtlm1:alfrescoNtlm

# Alfresco Identity Service configuration
```

```
#identity-service.auth-server-url=http://mbp512-mbergljung-0917.local:8888/auth
# Example URL on article author's Windows 10
#identity-service.auth-server-url=http://host.docker.internal:8888/auth
identity-service.realm=alfresco-dbp
identity-service.resource=alfresco-client
```

If you used the same Alfresco realm name as the article (i.e. `alfresco-dbp`) and the same Alfresco client name (i.e. `alfresco-client`), then you only have to change the `identity-service.auth-server-url` value. This URL needs to have a host/domain name that will work from both inside the ACS container and from the outside of the ACS container (e.g. from the Browser).

Note that we add the `identity-service` subsystem to the authentication chain, if we don't do that, then the Identity Service will not be active and none of the related properties will take effect.

The ACS container needs to have this properties file mapped into it for it to be picked up. Uncomment the following line in **my-dbp-identity-service/docker-compose-dbp/docker-compose.yml** file for the `alfresco` service:

```
#       - ./acs/alfresco-global.properties:/usr/local/tomcat/shared/classes/alfresco-global.properties
```

Restart the ACS container

To test the Identity Service configuration restart the ACS container as follows:

```
docker-compose-dbp mbergljung$ docker-compose stop alfresco
Stopping docker-compose-dbp_alfresco_1 ... done
docker-compose-dbp mbergljung$ docker-compose rm alfresco
Going to remove docker-compose-dbp_alfresco_1
Are you sure? [yN] y
Removing docker-compose-dbp_alfresco_1 ... done
docker-compose-dbp mbergljung$ docker-compose up -d --no-deps alfresco

Creating docker-compose-dbp_alfresco_1 ... done
```

Note. you need to fully remove the container for these changes to take effect.

In the original console window where you started the DBP you should see the logs for the ACS starting up again. Look for logs such as:

```
…
alfresco_1               | 2020-05-22 09:42:05,847  INFO
 [management.subsystems.ChildApplicationContextFactory] [localhost-startStop-1] Starting
'Authentication' subsystem, ID: [Authentication, managed, identity-service1]
alfresco_1               | 2020-05-22 09:42:06,970  INFO
 [management.subsystems.ChildApplicationContextFactory] [localhost-startStop-1] Startup of
'Authentication' subsystem, ID: [Authentication, managed, identity-service1] complete
a
…
alfresco_1               | 22-May-2020 09:43:16.480 INFO [main]
org.apache.coyote.AbstractProtocol.start Starting ProtocolHandler ["http-nio-8080"]
alfresco_1               | 22-May-2020 09:43:16.517 INFO [main]
org.apache.coyote.AbstractProtocol.start Starting ProtocolHandler ["ajp-nio-8009"]
alfresco_1               | 22-May-2020 09:43:16.539 INFO [main]
org.apache.catalina.startup.Catalina.start Server startup in 117866 ms
```

There is no ACS web application that we can use to verify that ACS keycloak configuration is correct (If you use SAML instead of OAuth you can get SSO with Share too, see https://docs.alfresco.com/saml/concepts/saml-overview.html and https://github.com/Alfresco/alfresco-identity-service/blob/1.0.0/docs/config/ping-federate-example.md). So we have to call an ACS ReST API instead to test it.

Request an access token first via `curl` (or with Postman, note that you would have to change the hostname to whatever you are using), note the parameter `grant_type=password`, which means that this uses the OAuth2 Password Grant flow:

```
$ curl -d 'client_id=alfresco-client' -d 'username=admin' -d 'password=admin' -d
'grant_type=password' 'http://mbp512-mbergljung-0917.local:8888/auth/realms/alfresco-
dbp/protocol/openid-connect/token' | python -m json.tool
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  2556  100  2481  100    75  10693    323 --:--:-- --:--:-- --:--:-- 10969
{
    "access_token":
"eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJBZlBSWnNaLTJjQ3BnblhXanAwcmpmV2tZc2djNUlBT0NZN0NzRkd6X0E4In0.eyJqdGkiOiJhYWI0NWNiMy
-LISYwMcp6r3tzNHtwB4U9ln33z8Dhwc0NYxj3jwPzXSQVK_45fSk0eg-
Bifz9Mz8HZmRPOpfpvmL28jK7TrsS9AhMyBpAupb9eZCV39szFI61T9qafcm8fnLL8vGXpmxkgCb5IrO-kyL-
xhymc9p9JDOwdXf7WNMICFH10zTXPrhTz78JKbtNzue2VGTXPJ-nazvcOJg1NdgOnsG-
m80sV1aUMmgBOOK5RBLKUZrkWFTH_jjG2URW7Ka1TUrdrSax8PrsFTv3h34ryTDjwAi-
_Jwc10wsfEEt8A2MhBdIIKkHQzJJHQ4GA6a5VXH1Krfe5a-141zsARxg",
    "expires_in": 300,
    "refresh_expires_in": 1800,
    "refresh_token":
"eyJhbGciOiJIUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJmYjUxNWNkNi1kNmU3LTQwZDctYTJjYy05Y2VlOGFkOTY2NTQifQ.eyJqdGkiOiJhMTlmOTQ3MS1iZGVmLTQ
```

Then test an ACS ReST API with the access token (note that you would have to change the hostname to whatever you are using, and the token of course):

```
$ curl http://mbp512-mbergljung-0917.local:8080/alfresco/api/-default-/public/alfresco/versions/1/nodes/-root-/children -H "Authorization: bearer eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJBZlBSWnNaLTJjQ3BnblhXanAwcmpmV2tZc2djNUlBT0NZN0NzRkd6X0E4In0.eyJqdGkiOiJhYWI0NWNiMy0-LISYwMcp6r3tzNHtwB4U9ln33z8Dhwc0NYxj3jwPzXSQVK_45fSk0eg-Bifz9Mz8HZmRPOpfpvmL28jK7TrsS9AhMyBpAupb9eZCV39szFI61T9qafcm8fnLL8vGXpmxkgCb5IrO-kyL-xhymc9p9JDOwdXf7WNMICFH10zTXPrhTz78JKbtNzue2VGTXPJ-nazvcOJg1NdgOnsG-m80sV1aUMmgBOOK5RBLKUZrkWFTH_jjG2URW7Ka1TUrdrSax8PrsFTv3h34ryTDjwAi-_Jwc10wsfEEt8A2MhBdIIKkHQzJJHQ4GA6a5VXH1Krfe5a-141zsARxg" | python -m json.tool
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  2758    0  2758    0     0   3940      0 --:--:-- --:--:-- --:--:--  3934
{
    "list": {
        "pagination": {
            "count": 7,
            "hasMoreItems": false,
            "totalItems": 7,
            "skipCount": 0,
            "maxItems": 100
        },
        "entries": [
            {
                "entry": {
                    "createdAt": "2020-05-21T10:31:58.831+0000",
                    "isFolder": true,
                    "isFile": false,
                    "createdByUser": {
                        "id": "System",
                        "displayName": "System"
                    },
                    "modifiedAt": "2020-05-21T10:32:57.038+0000",
                    "modifiedByUser": {
                        "id": "System",
                        "displayName": "System"
                    },
                    "name": "Data Dictionary",
                    "id": "fcd204f5-4c2a-495d-9c29-0f27db329613",
                    "nodeType": "cm:folder",
                    "parentId": "16e0cdb4-1786-4eff-9b6e-955e53a2e1d7"
                }
            },
            {
                "entry": {
                    "createdAt": "2020-05-21T10:32:00.326+0000",
                    "isFolder": true,
                    "isFile": false,
                    "createdByUser": {
                        "id": "System",
                        "displayName": "System"
                    },
                    "modifiedAt": "2020-05-21T10:32:00.326+0000",
                    "modifiedByUser": {
                        "id": "System",
                        "displayName": "System"
                    },
                    "name": "Guest Home",
                    "id": "a9263162-2761-4bf4-9792-c4692a1d5533",
                    "nodeType": "cm:folder",
                    "parentId": "16e0cdb4-1786-4eff-9b6e-955e53a2e1d7"
                }
            },
            {
                "entry": {
                    "createdAt": "2020-05-21T10:32:00.732+0000",
                    "isFolder": true,
                    "isFile": false,
                    "createdByUser": {
                        "id": "System",
                        "displayName": "System"
                    },
                    "modifiedAt": "2020-05-21T10:32:00.732+0000",
                    "modifiedByUser": {
                        "id": "System",
                        "displayName": "System"
                    },
                    "name": "Imap Attachments",
                    "id": "91503c28-b082-4e86-8542-4fe3e5171995",
                    "nodeType": "cm:folder",
                    "parentId": "16e0cdb4-1786-4eff-9b6e-955e53a2e1d7"
                }
            },
            {
                "entry": {
```

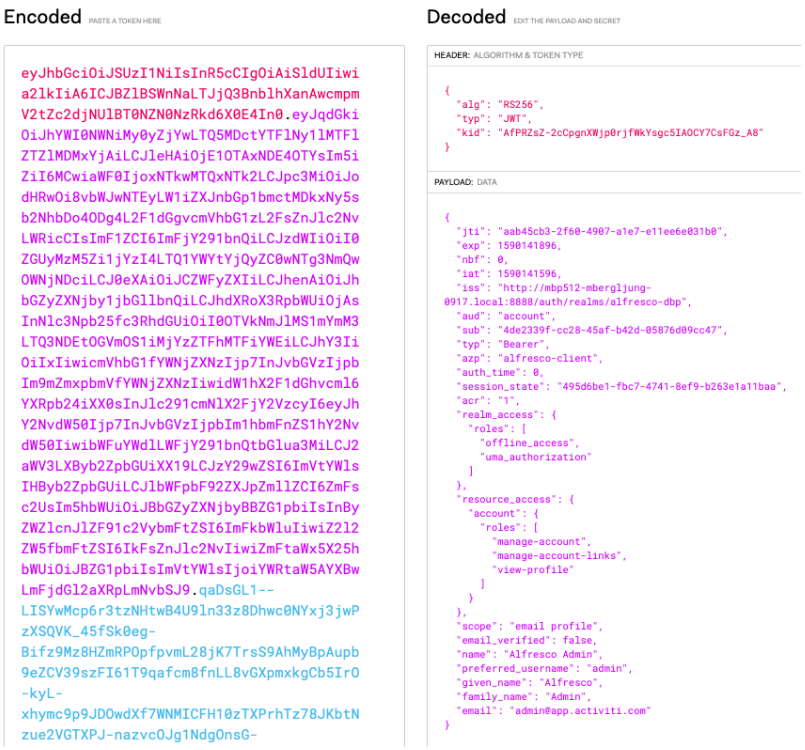
```
                    },
                    "modifiedAt": "2020-05-21T10:32:00.776+0000",
                    "modifiedByUser": {
                        "id": "System",
                        "displayName": "System"
                    },
                    "name": "IMAP Home",
                    "id": "5d9af67e-020b-49dc-a4dc-737a8535e458",
                    "nodeType": "cm:folder",
                    "parentId": "16e0cdb4-1786-4eff-9b6e-955e53a2e1d7"
                }
            },
            {
                "entry": {
                    "createdAt": "2020-05-21T10:32:00.569+0000",
                    "isFolder": true,
                    "isFile": false,
                    "createdByUser": {
                        "id": "System",
                        "displayName": "System"
                    },
                    "modifiedAt": "2020-05-21T10:32:00.569+0000",
                    "modifiedByUser": {
                        "id": "System",
                        "displayName": "System"
                    },
                    "name": "Shared",
                    "id": "21210e31-ad8d-4473-a796-974b732de266",
                    "nodeType": "cm:folder",
                    "parentId": "16e0cdb4-1786-4eff-9b6e-955e53a2e1d7"
                }
            },
            {
                "entry": {
                    "createdAt": "2020-05-21T10:32:17.268+0000",
                    "isFolder": true,
                    "isFile": false,
                    "createdByUser": {
                        "id": "System",
                        "displayName": "System"
                    },
                    "modifiedAt": "2020-05-21T10:32:51.500+0000",
                    "modifiedByUser": {
                        "id": "System",
                        "displayName": "System"
                    },
                    "name": "Sites",
                    "id": "b4717a16-b18e-4f40-a87f-481933ad19b8",
                    "nodeType": "st:sites",
                    "parentId": "16e0cdb4-1786-4eff-9b6e-955e53a2e1d7"
                }
            },
            {
                "entry": {
                    "createdAt": "2020-05-21T10:32:00.530+0000",
                    "isFolder": true,
                    "isFile": false,
                    "createdByUser": {
                        "id": "System",
                        "displayName": "System"
                    },
                    "modifiedAt": "2020-05-21T10:32:00.530+0000",
                    "modifiedByUser": {
                        "id": "System",
                        "displayName": "System"
                    },
                    "name": "User Homes",
                    "id": "cb3ad1ed-e941-4678-a0e4-713d8505cd28",
                    "nodeType": "cm:folder",
                    "parentId": "16e0cdb4-1786-4eff-9b6e-955e53a2e1d7"
                }
            }
        ]
    }
}
```


Everything is working, time to move on to the ADF applications and how to configure them to integrate with the Alfresco Identity Service.

Decoding Access Tokens

OpenID Connect uses the JWT access token from OAuth2, which is a token that is used to access authorized resources. It contains information about the issuer (the authorization server - i.e. Alfresco Identity Service), audiences for whom the access token is for (i.e. the user) and a scope list which are

A JWT is encoded in base64, so it can easily be decoded on the https://jwt.io/ page. If we took above access token and decoded it we would get the following:



As we can see, this access token is for the *Administrator* user.

Configure and Run an ADF Application

I'm going to use the ADF Project generator. I already got `node` and `npm` installed:

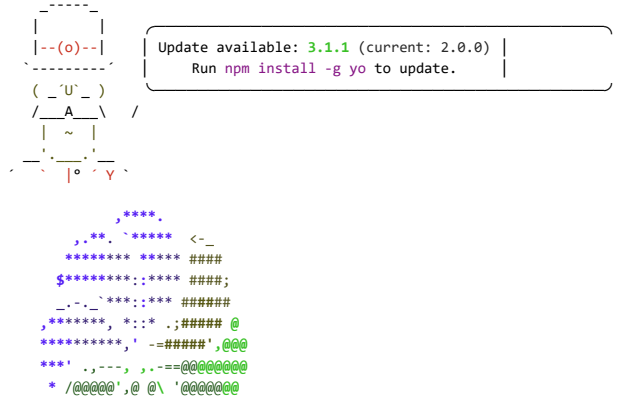
```
$ node -v
v10.12.0
$ npm -v
6.4.1
```

Make sure you have the newest ADF app generator as the Identity Service integration only works with ADF version 2.4.0 or newer:

$ sudo npm uninstall -g generator-alfresco-adf-app
$ npm install -g generator-alfresco-adf-app
...
+ generator-alfresco-adf-app@3.8.0
...

Now, generate an ADF application, standing for example in the **my-dbp-identity-service** directory:

```
my-dbp-identity-service mbergljung$ yo alfresco-adf-app
```



```
     ADF Angular app generator for Alfresco
           Version 3.8.0

? Your project name adf-app-client
? Application blueprint Content and Process Services with APS 1
? Would you like to install dependencies now? No
adf-cli-acs-aps-template
? What's your name: Martin Bergljung
? Your email (optional): xxxxxxxxxxx
? Your website (optional):
? Which license do you want to use? MIT
   create .editorconfig
   create .npmignore
   create .travis.yml
   create Dockerfile
   create README.md
   create angular.json
   create docker.md
   create karma.conf.js
   create nginx.conf
   create package.json
   create protractor-ci.conf.js
   create protractor.conf.js
   create proxy.conf.js
   create tsconfig.json
   create tslint.json
   create e2e/app.e2e-spec.ts
   create e2e/app.po.ts
   create e2e/tsconfig.e2e.json
   create src/app.config.json
   create src/custom-style.scss
   create src/favicon-96x96.png
   create src/index.html
   create src/main.ts
   create src/polyfills.ts
   create src/test.ts
   create src/tsconfig.app.json
   create src/tsconfig.spec.json
   create src/typings.d.ts
   create resources/i18n/de.json
   create resources/i18n/en.json
   create resources/i18n/es.json
   create resources/i18n/fr.json
   create resources/i18n/it.json
   create resources/i18n/ja.json
   create resources/i18n/nb.json
   create resources/i18n/nl.json
   create resources/i18n/pt-BR.json
   create resources/i18n/ru.json
   create resources/i18n/zh-CN.json
   create src/app/app.component.html
   create src/app/app.component.scss
   create src/app/app.component.spec.ts
   create src/app/app.component.ts
   create src/app/app.module.ts
   create src/app/app.routes.ts
   create src/app/stencils.module.ts
   create src/assets/.gitkeep
   create src/environments/environment.prod.ts
   create src/environments/environment.ts
   create src/app/app-layout/app-layout.component.css
   create src/app/app-layout/app-layout.component.html
   create src/app/app-layout/app-layout.component.spec.ts
   create src/app/app-layout/app-layout.component.ts
   create src/app/apps/apps.component.css
   create src/app/apps/apps.component.html
   create src/app/apps/apps.component.ts
   create src/app/documentlist/documentlist.component.css
   create src/app/documentlist/documentlist.component.html
   create src/app/documentlist/documentlist.component.spec.ts
   create src/app/documentlist/documentlist.component.ts
   create src/app/file-view/blob-view.component.ts
   create src/app/file-view/bob-view.component.html
   create src/app/file-view/file-view.component.html
   create src/app/file-view/file-view.component.scss
   create src/app/file-view/file-view.component.ts
   create src/app/home/home.component.css
   create src/app/home/home.component.html
   create src/app/home/home.component.spec.ts
   create src/app/home/home.component.ts
   create src/app/login/login.component.css
   create src/app/login/login.component.html
   create src/app/login/login.component.spec.ts
   create src/app/login/login.component.ts
   create src/app/services/preview.service.ts
   create src/app/start-process/start-process.component.html
```

```
   create src/app/task-details/task-details.component.spec.ts
   create src/app/task-details/task-details.component.ts
   create src/app/tasks/tasks.component.css
   create src/app/tasks/tasks.component.html
   create src/app/tasks/tasks.component.spec.ts
   create src/app/tasks/tasks.component.ts
   create LICENSE
```

The ACS and APS URLs need to be configured in the newly generated ADF app as the default ones don't match. Open up the **my-dbp-identity-service/adf-app-client/proxy.conf.js** file and change it to the following:

```
{
 "/alfresco": {
   "target": "http://localhost:8082",
   "secure": false,
   "changeOrigin": true
 },
 "/activiti-app": {
   "target": "http://localhost:9080",
   "secure": false,
   "changeOrigin": true
 }
}
```

We also need to do some configuration to enable Identity Service authentication, open up the **my-dbp-identity-service/adf-app-client/src/app.config.json** file and configure OAuth2 as follows, you need to change the **host** URL in the oauth2 config to match your environment:

```
{
   "$schema": "../node_modules/@alfresco/adf-core/app.config.schema.json",
   "ecmHost": "http://{hostname}{:port}",
   "bpmHost": "http://{hostname}{:port}",
   "providers" : "ALL",
   "authType" :"OAUTH",
   "oauth2": {
     "host": "http://mbp512-mbergljung-0917.local:8888/auth/realms/alfresco-dbp",
     "clientId": "alfresco-client",
     "scope": "openid",
     "secret": "",
     "implicitFlow": true,
     "silentLogin": false,
     "redirectUri": "/",
     "redirectUriLogout": "/logout"
   },
   "application": {
       "name": "Alfresco ADF Application"
   },
   "languages": [
...
```

The following table explains the **oauth2** of the properties:

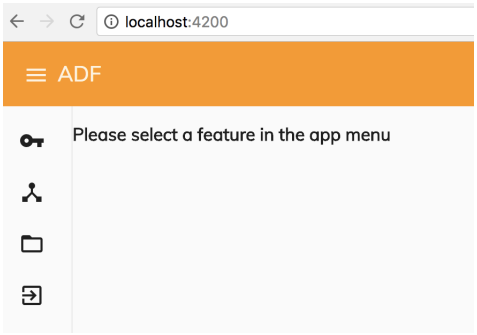
Property	Value	Description
host	**http://mbp512-mbergljung-0917.local:8888/auth/realms/alfresco-dbp**	The URL to the realm (i.e. alfresco-dpb) we created in the Alfresco Identity Service (i.e. keycloak). Here you might be tempted to use `localhost` as ADF does not run inside a container. However, that will not work, you will get errors like the following on the server side: *"Token validation failed: Invalid token issuer. Expected 'http://mbp512-mbergljung-0917.local:8888/auth/realms/alfresco-dbp', but was 'http://localhost:8888/auth/realms/alfresco-dbp'"*
clientId	**alfresco-client**	The OAuth2 client ID. Each application has a client-id that is used to identify the application. We use the same client ID for all Alfresco clients (i.e. APS web client, ADF web client).
scope	**openid**	OpenID Connect (OIDC) scopes are used by an application during authentication to authorize access to a user's details, like name and picture. Specifying `openid` indicates that the application intends to use OIDC to verify the user's identity.

		In this case we don't use a secret so any application that knows a username/password can request tokens.
`implicitFlow`	`true`	This means that the ADF application will use the OAuth2 Implicit Grant flow and store the JWT in Local Storage between ReST calls. Login will be via Identity Service Login dialog. If you set this property to `false`, then it will use the OAuth2 Password Grant flow. This means that the username and password will be entered via the ADF Login component.
`silentLogin`	`false`	When this is set to `false` an extra dialog (with a **Login SSO** button) will pop up where you have to click a button to execute SSO login. If set to `true`, then this extra dialog is bypassed and the application will redirect automatically to the authorization server (i.e. Identity Service) when the user is not logged-in.
`redirectUri`	`/`	Where to redirect after a successful login via the Identity Service. By using a single slash (/) you will get a redirect URL to the home page for the ADF app: http://localhost:4200/

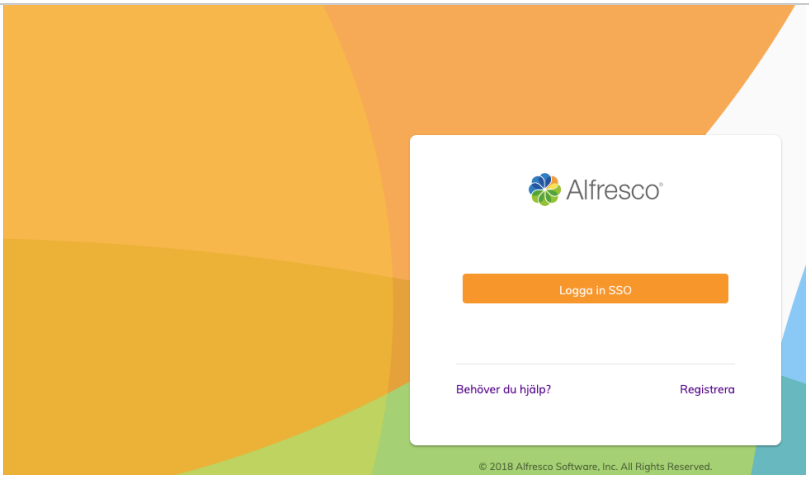
Now, install all packages and then start the ADF App as follows:

```
my-dbp-identity-service mbergljung$ cd adf-app-client/
adf-app-client mbergljung$ npm install
...
adf-app-client mbergljung$ npm start
…
** Angular Live Development Server is listening on localhost:4200, open your browser on
http://localhost:4200/ **

Date: 2020-05-22T10:17:24.693Z
Hash: 798cef4c104db00eb250
Time: 22192ms
chunk {main} main.js, main.js.map (main) 94.7 kB [initial] [rendered]
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 449 kB [initial] [rendered]
chunk {runtime} runtime.js, runtime.js.map (runtime) 6.08 kB [entry] [rendered]
chunk {scripts} scripts.js, scripts.js.map (scripts) 1.08 MB  [rendered]
chunk {styles} styles.js, styles.js.map (styles) 1.01 MB [initial] [rendered]
chunk {vendor} vendor.js, vendor.js.map (vendor) 14.2 MB [initial] [rendered]

WARNING in ./node_modules/minimatch/minimatch.js
Module not found: Error: Can't resolve 'path' in '/Users/mbergljung/IDEAProjects/case-management/my-dbp-
identity-service/adf-app-client/node_modules/minimatch'
i 「wdm」: Compiled with warnings.
```

And then accessed the UI on URL http://localhost:4200.

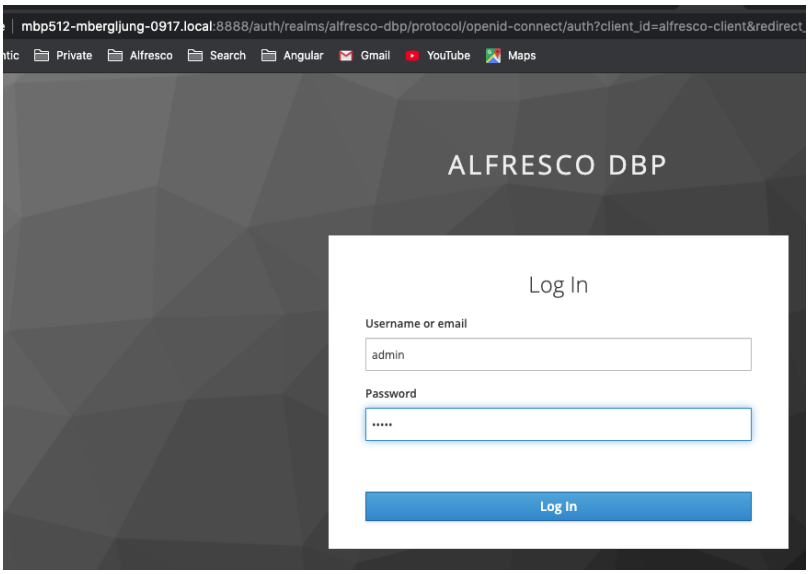


Now click on the key icon to the left and you should be presented with a SSO login dialog:



Click on the **Login SSO** button.

It should redirect you to Keycloak for authentication:

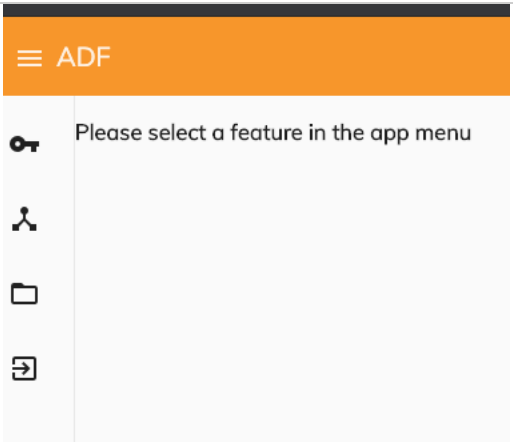


The keycloak URL looks something like this:

http://mbp512-mbergljung-0917.local:8888/auth/realms/alfresco-dbp/protocol/openid-connect/auth?client_id=alfresco-client&redirect_uri=http%3A%2F%2Flocalhost%3A4200%2F&scope=openid&response_type=id_token%20token&nonce=NwhAJmIHowW

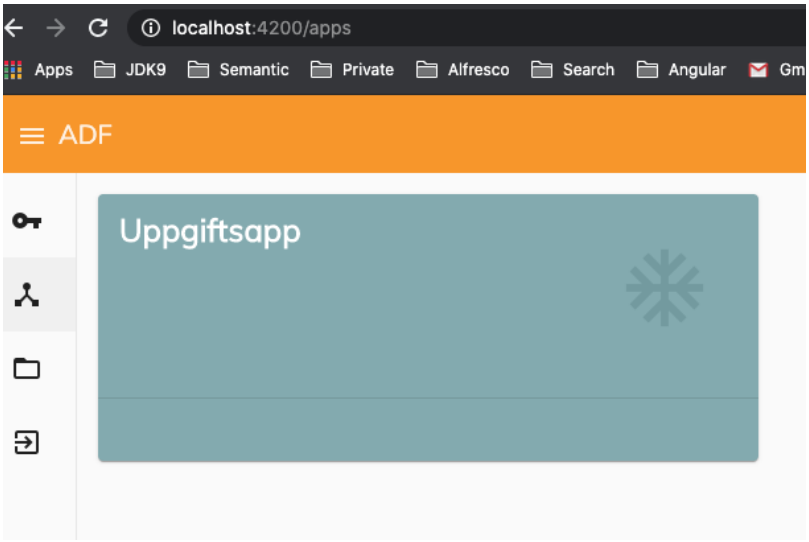
Note the parameter `response_type=id_token`, which means that this app uses the OAuth2 Implicit Grant flow.

And after successfully logging in (with **admin**/**admin**) you should be redirected back to the ADF app as per the `redirect_uri` in the above URL. You should now see the following ADF app UI:

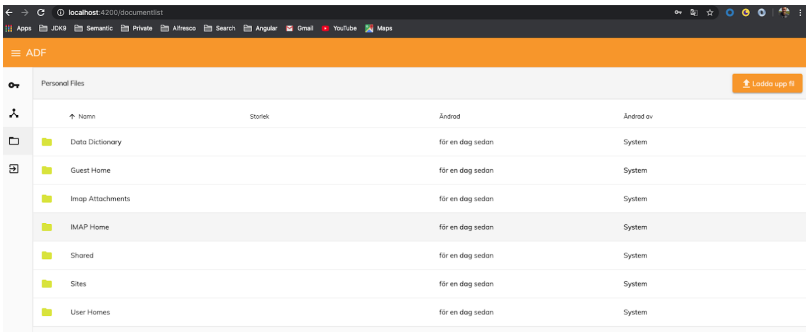


Now click on both Process Apps and Document List in the left navigation menu, you should not be prompted to login.

Process Apps:



Document List:



Test also that you can logout by clicking the bottom icon in the left navigation menu.

Using an External LDAP Server

So far we created the user directly in Alfresco Identity Service (i.e. Keycloak). You will most likely not create and store your users in the keycloak database. Most solutions will use an enterprise LDAP server, such as MS Active Directory or OpenLDAP. We will look at how to use an external LDAP server by installing Apache DS locally and then populate it with a couple of users that we will then use for authentication.

```
. . .
ldap:
 image: greggigon/apacheds
 environment:
   - BOOTSTRAP_FILE=/bootstrap/demo.ldif
 restart: always
 ports:
   - 10389:10389
 volumes:
   - ./data/ldap/data:/data
   - ./ldap/bootstrap:/bootstrap
```

Make sure you have cloned the source code project as it has an LDIF file located at **my-dbp-identity-service/docker-compose-identity/ldap/bootstrap/demo.ldif** that is used to bootstrap the users.

Restart everything (i.e Ctrl+C and then docker-compose up):

```
...
auth_1      | 08:31:35,271 INFO  [org.jboss.as] (Controller Boot Thread) WFLYSRV0051: Admin console
listening on http://127.0.0.1:10798
auth_1      | 08:31:35,271 INFO  [org.jboss.as] (Controller Boot Thread) WFLYSRV0025: Keycloak 8.0.1
(WildFly Core 10.0.3.Final) started in 26206ms - Started 684 of 989 services (701 services are lazy,
passive or on-demand)
^CGracefully stopping... (press Ctrl+C again to force)
Stopping docker-compose-identity_auth_1     ... done
Stopping docker-compose-identity_postgres_1 ... done
MBP512-MBERGLJUNG-0917:docker-compose-identity mbergljung$ docker-compose up
Starting docker-compose-identity_postgres_1 ... done
Creating docker-compose-identity_ldap_1     ... done
Starting docker-compose-identity_auth_1     ... done
Attaching to docker-compose-identity_postgres_1, docker-compose-identity_ldap_1, docker-compose-
identity_auth_1
ldap_1      | Starting ApacheDS - default…
...
```

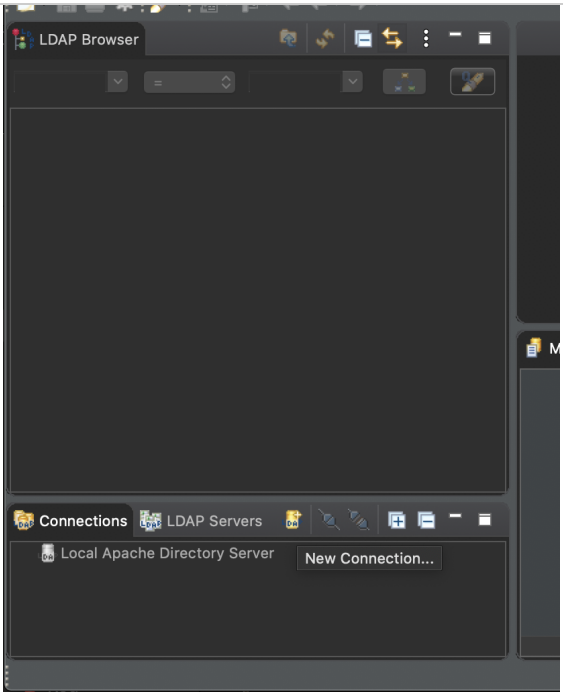
You should see something like this in the logs:

```
...
ldap_1      | Bootstraping Apache DS with Data from /bootstrap/demo.ldif
ldap_1      | adding new entry "ou=People,dc=example,dc=com"
ldap_1      | adding new entry "ou=RealmRoles,dc=example,dc=com"
ldap_1      | adding new entry "ou=FinanceRoles,dc=example,dc=com"
ldap_1      | adding new entry "uid=jbrown,ou=People,dc=example,dc=com"
ldap_1      | adding new entry "uid=bwilson,ou=People,dc=example,dc=com"
ldap_1      | adding new entry "cn=ldap-user,ou=RealmRoles,dc=example,dc=com"
ldap_1      | adding new entry "cn=ldap-admin,ou=RealmRoles,dc=example,dc=com"
ldap_1      | adding new entry "cn=accountant,ou=FinanceRoles,dc=example,dc=com"
...
```

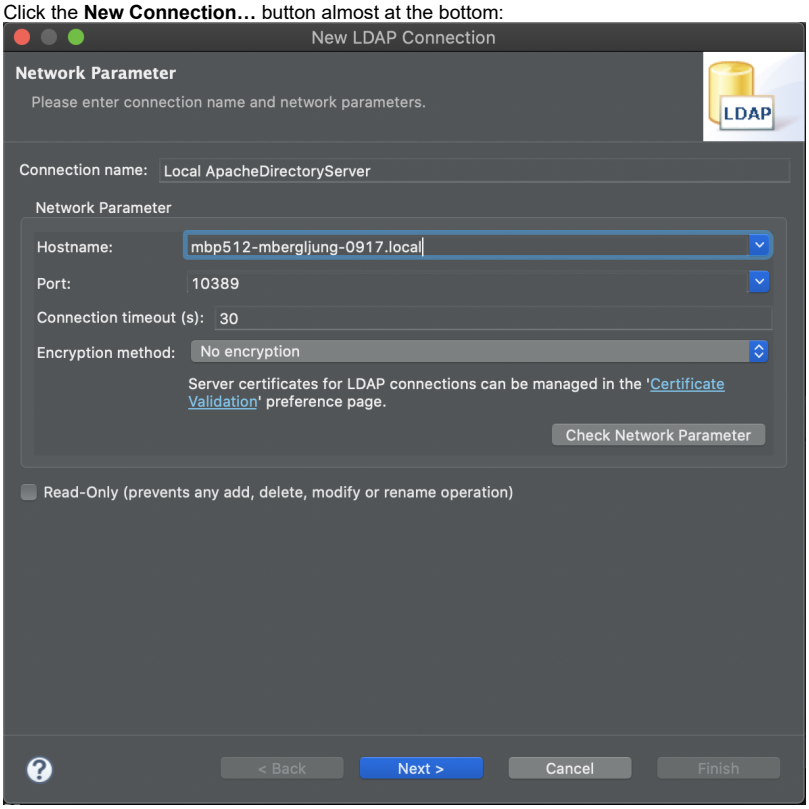
Installing Apache Directory Studio

If you want to you can install Apache Directory Studio and have a look at the directory from there:

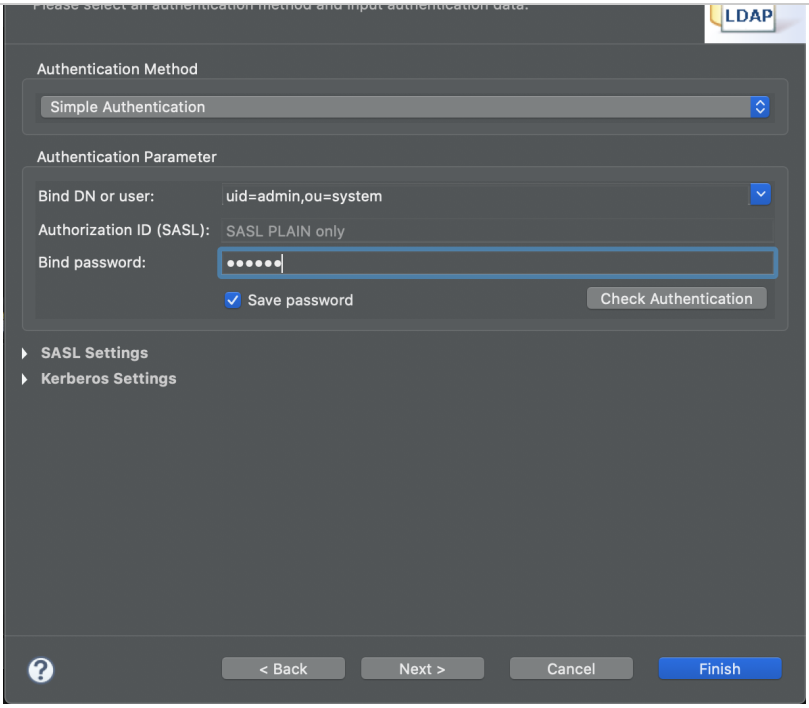
Create a connection:



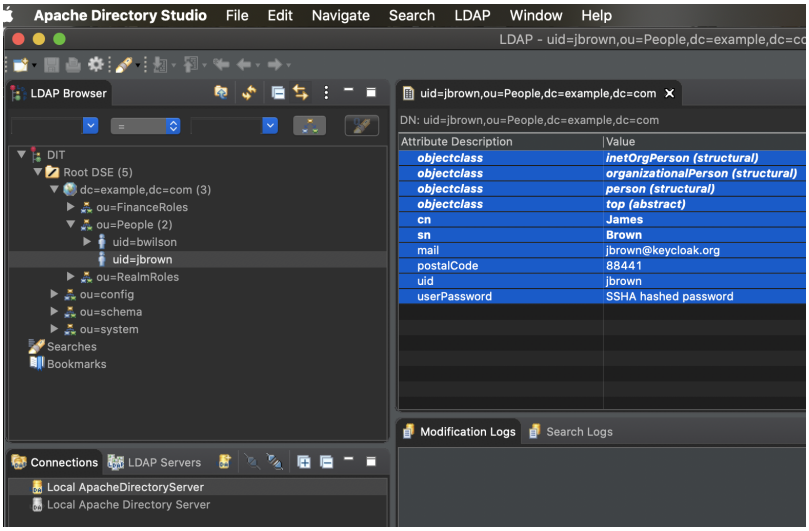
Click the **New Connection…** button almost at the bottom:



Fill in hostname and port, then check that they work by clicking on the **Check Network Parameter** button. Then click **Next>** button:



Fill in admin credentials (i.e. uid=admin,ou=system and password as secret). Then click the **Check Authentication** button to verify. Then click **Finish** button:



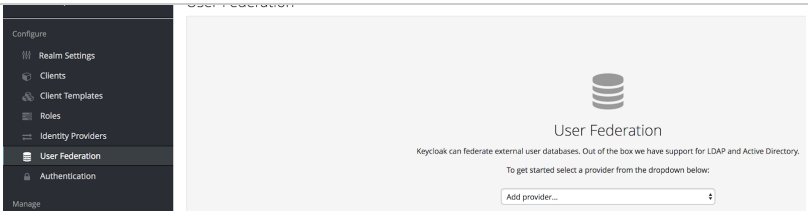
We can see the two users that will be imported.

Configure the Identity Service to use the Directory Server

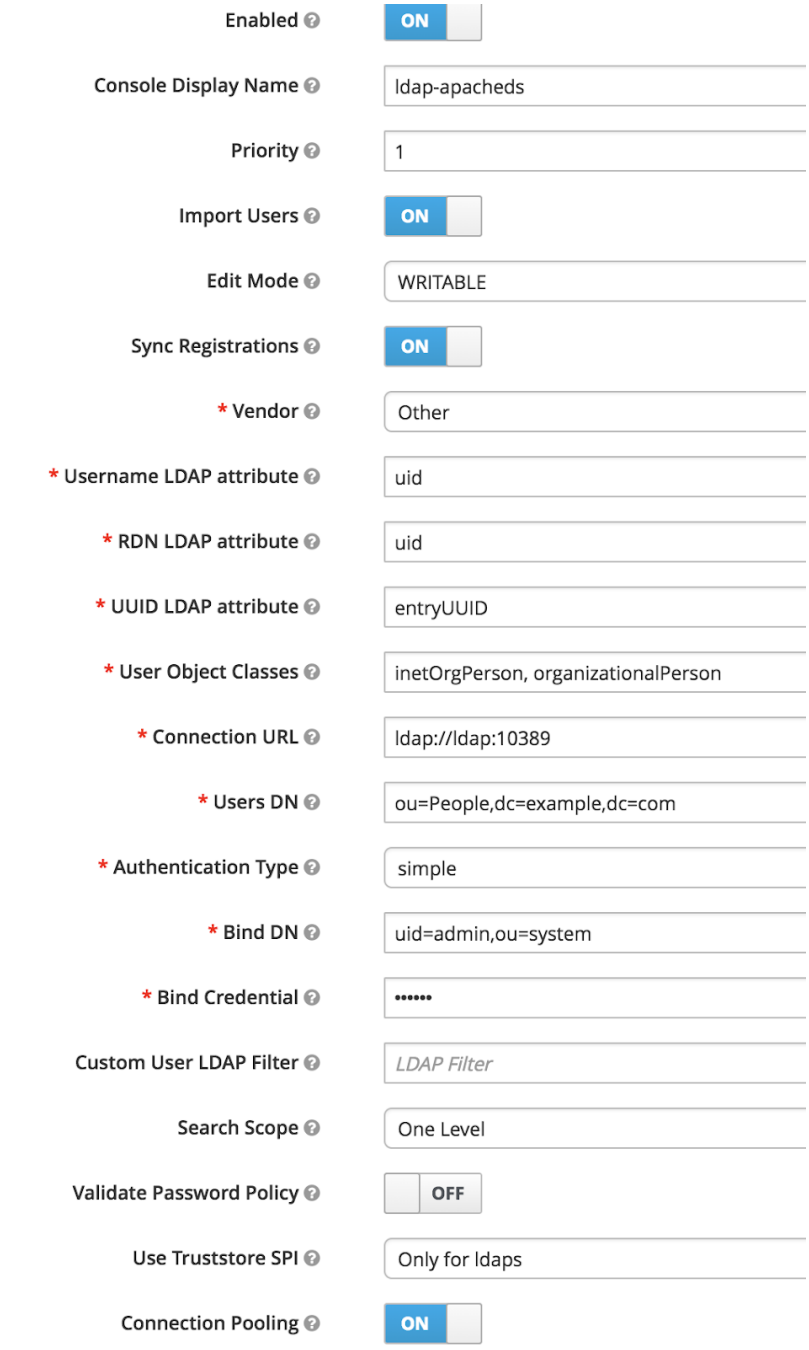
So we now got the directory server running and it is populated with a couple of users. Time to integrate it with the Identity Service (Keycloak) so we can authenticate with these new users. Keycloak can federate external user databases. Out of the box there's support for LDAP and Active Directory.

Go to the Admin Console in Keycloak via http://localhost:8888/auth/admin/master/console. Login with **admin/admin**.

In the realm **alfresco-dbp** that we created earlier on click on the **Users Federation** menu option to get you to the User Federation page:



When you get to this page, there is an **Add Provider...** select box. You should see **ldap** within this list. Selecting **ldap** will bring you to the ldap configuration page. We will add here the **ApacheDS LDAP** settings:



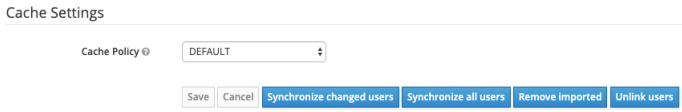
Here are the fields that you need to configure:

- **Console Display Name**: `ldap-apacheds` - good to have a name corresponding to the directory server that we use
- **Priority**: `1` - use this directory server first, if you have more directory servers configured
- **Console Display Name**: `WRITEABLE` - Data will be synced back to apacheds when needed (**Import Users** mean that users will be imported into keycloak from apacheds)
- **Sync Registrations**: `ON` - newly created users will be synced back to apacheds

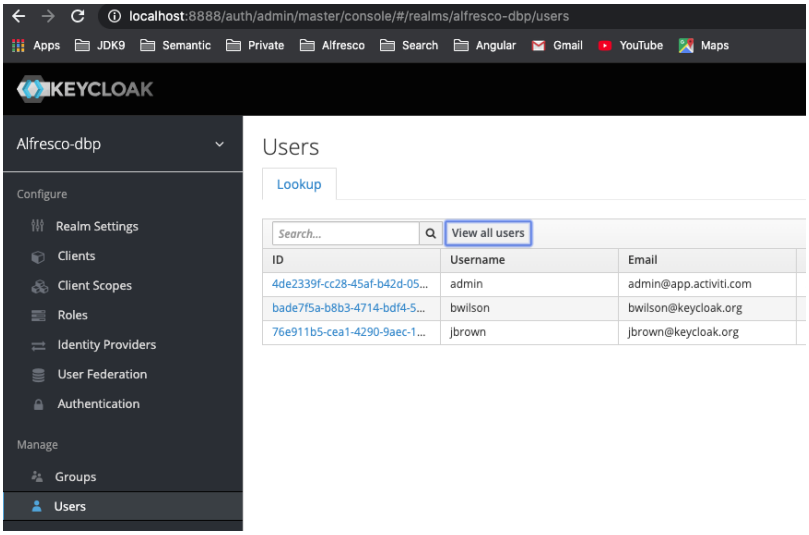
directory
- **Bind DN**: `uid=admin,ou=system` - this is the user that keycloak will use when talking to apacheds
- **Bind Credentials**: `secret` - this is the password that keycloak will use when talking to apacheds. Make sure to click the **Test authentication** button to verify that the authentication with the admin user works

Click the **Save** button at the bottom of the page to store the settings permanently.

At the bottom of the page you will also see a button that can be used to immediately import the LDAP users. Click the **Synchronize all users** button:



You can now check the users by clicking **Users** in the left navigation menu and then click **View all users**:

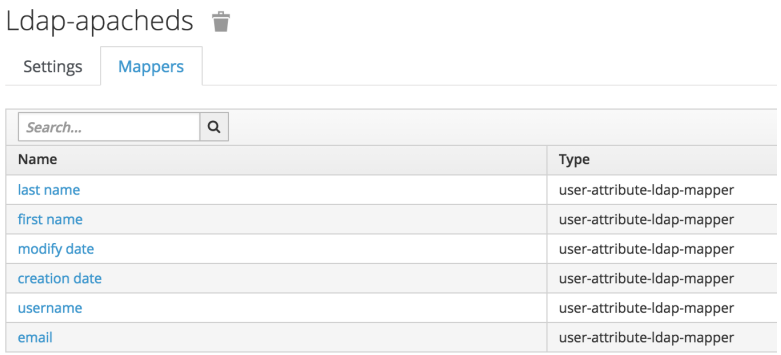


Note the two new imported LDAP users *jbrown* and *bwilson*. If we don't manually import the users then they will be imported later on based on a preconfigured schedule.

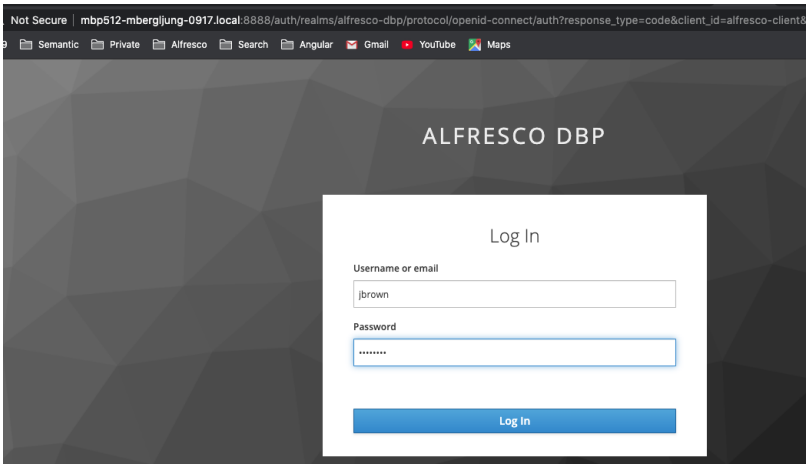
Mappers

Out of the box, Keycloak is configured to import only username, email, first and last name, but you are free to configure mappers and add more attributes or delete default ones. It supports password validation via LDAP/AD protocols and different user metadata synchronization modes.

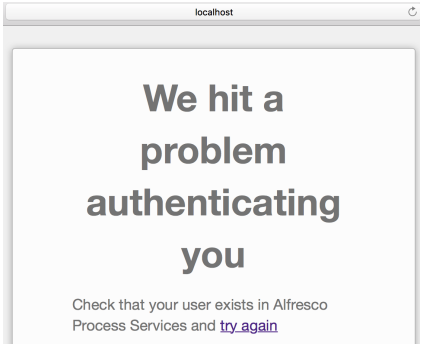
Here is the full list of preconfigured attributes for our schema:



Configure APS so it knows about the users in the Directory



You will see an error message as follows when trying to login with a directory user:



Configure LDAP Sync as follows in **my-dbp-identity-service/docker-compose-dbp/aps/activiti-ldap.properties**:

```
# Uncomment properties to enable LDAP Sync
# LDAP Connection
ldap.authentication.java.naming.provider.url=ldap://ldap:10389
ldap.synchronization.java.naming.security.principal=uid=admin,ou=system
ldap.synchronization.java.naming.security.credentials=secret
# LDAP Sync
ldap.synchronization.full.enabled=true
ldap.synchronization.full.cronExpression=0 0 0 * * ?
ldap.synchronization.userSearchBase=ou=People,dc=example,dc=com
ldap.synchronization.personQuery=(objectclass\=inetOrgPerson)
# Need to set proper group search base, otherwise we get exceptions
ldap.synchronization.groupSearchBase=ou=RealmRoles,dc=example,dc=com
ldap.synchronization.groupQuery=(objectclass\=groupOfNames)
```

You shouldn't have to change anything in the above configuration.

This file already exist in your cloned project, you just need to mount it into the container by uncommenting the following volume mapping for the `process` service (in my-dbp-identity-service/docker-compose-dbp/docker-compose.yml):

```
...
process:
   image: alfresco/process-services:1.11.0
...
     - ./aps/activiti-ldap.properties:/usr/local/tomcat/lib/activiti-ldap.properties
...
```

Now, restart the `process` container as follows:

```
docker-compose-dbp mbergljung$ docker-compose stop process
Stopping docker-compose-dbp_process_1 ... done
docker-compose-dbp mbergljung$ docker-compose rm process
Going to remove docker-compose-dbp_process_1
Are you sure? [yN] y
Removing docker-compose-dbp_process_1 ... done
docker-compose-dbp mbergljung$ docker-compose up -d --no-deps process
Creating docker-compose-dbp_process_1 ... done
```

The logs should show the LDAP sync progress:

```
process_1               | 10:30:38 [localhost-startStop-1] INFO
 com.activti.idm.ldap.service.LdapSettingsManager  - userQuery = (objectclass=inetOrgPerson)
process_1               | 10:30:38 [localhost-startStop-1] INFO
 com.activti.idm.ldap.service.LdapSettingsManager  - userDifferentialQuery = (&(objectclass=inetOrgPerson)(!
(modifyTimestamp<={0})))
process_1               | 10:30:38 [localhost-startStop-1] INFO
 com.activti.idm.ldap.service.LdapSettingsManager  - userIdAttributeName = uid
process_1               | 10:30:38 [localhost-startStop-1] INFO
 com.activti.idm.ldap.service.LdapSettingsManager  - userFirstNameAttributeName = givenName
process_1               | 10:30:38 [localhost-startStop-1] INFO
 com.activti.idm.ldap.service.LdapSettingsManager  - userLastNameAttributeName = sn
process_1               | 10:30:38 [localhost-startStop-1] INFO
 com.activti.idm.ldap.service.LdapSettingsManager  - userEmailAttributeName = 'mail'
process_1               | 10:30:38 [localhost-startStop-1] INFO
 com.activti.idm.ldap.service.LdapSettingsManager  - userType = 'inetOrgPerson'
process_1               | 10:30:38 [localhost-startStop-1] INFO
 com.activti.idm.ldap.service.LdapSettingsManager  - syncAdditionalUsers = 'true'
process_1               | 10:30:38 [localhost-startStop-1] INFO
 com.activti.idm.ldap.service.LdapSettingsManager  - groupSearchBase = 'ou=RealmRoles,dc=example,dc=com'
process_1               | 10:30:38 [localhost-startStop-1] INFO
 com.activti.idm.ldap.service.LdapSettingsManager  - groupQuery = '(objectclass=groupOfNames)'
process_1               | 10:30:38 [localhost-startStop-1] INFO
 com.activti.idm.ldap.service.LdapSettingsManager  - groupDifferentialQuery = '(&(objectclass=groupOfNames)(!
(modifyTimestamp<={0})))'
process_1               | 10:30:38 [localhost-startStop-1] INFO
 com.activti.idm.ldap.service.LdapSettingsManager  - groupIdAttributeName = 'cn'
process_1               | 10:30:38 [localhost-startStop-1] INFO
 com.activti.idm.ldap.service.LdapSettingsManager  - groupMemberAttributeName = member
process_1               | 10:30:38 [localhost-startStop-1] INFO
 com.activti.idm.ldap.service.LdapSettingsManager  - groupType = groupOfNames
process_1               | 10:30:38 [localhost-startStop-1] INFO
 com.activti.idm.ldap.service.LdapSettingsManager  - distinguishedNameAttributeName = dn
process_1               | 10:30:38 [localhost-startStop-1] INFO
 com.activti.idm.ldap.service.LdapSettingsManager  - createTimestampAttributeName = createTimestamp
process_1               | 10:30:38 [localhost-startStop-1] INFO
 com.activti.idm.ldap.service.LdapSettingsManager  - modifyTimestampAttributeName = 'modifyTimestamp'
process_1               | 10:30:38 [localhost-startStop-1] INFO
 com.activti.idm.ldap.service.LdapSettingsManager  - timeStampFormat = yyyyMMddHHmmss.SSS'Z', locale = (en,GB),
timezone = GMT
…
process_1               | 01:22:23 [localhost-startStop-1] INFO  com.activiti.ActivitiApplication  - Started
ActivitiApplication in 57.977 seconds (JVM running for 61.944)
process_1               | 01:22:23 [activiti-app-rest-Executor-2] INFO
 com.activiti.api.idm.AbstractExternalIdmSourceSyncService  - No initial LDAP sync info found. Executing full
synchronization.
process_1               | 01:22:23 [activiti-app-rest-Executor-2] INFO
 com.activiti.api.idm.AbstractExternalIdmSourceSyncService  - Starting full LDAP synchronization
process_1               | 01:22:23 [activiti-app-rest-Executor-2] INFO
 com.activiti.api.idm.AbstractExternalIdmSourceSyncService  - Starting to process the LDAP users and groups.
process_1               | 01:22:24 [activiti-app-rest-Executor-2] INFO
 com.activti.idm.ldap.service.LdapGroupContextMapper  - Found 2 members with attribute name member for group
ldap-user.
process_1               | 01:22:24 [activiti-app-rest-Executor-2] INFO
 com.activti.idm.ldap.service.LdapGroupContextMapper  - Found 1 members with attribute name member for group
ldap-admin.
process_1               | 01:22:24 [activiti-app-rest-Executor-2] INFO
 com.activiti.api.idm.AbstractExternalIdmSourceSyncService  - Found 2 groups and 2 users in LDAP
process_1               | 01:22:24 [activiti-app-rest-Executor-2] INFO
 com.activiti.api.idm.AbstractExternalIdmSourceSyncService  - Finished processing 2 users
process_1               | 01:22:25 [activiti-app-rest-Executor-2] INFO
 com.activiti.api.idm.AbstractExternalIdmSourceSyncService  - Finished processing 2 groups and its members.
process_1               | 01:22:25 [activiti-app-rest-Executor-2] INFO
 com.activiti.api.idm.AbstractExternalIdmSourceSyncService  - All groups and users processed. Checking for
deleted users
process_1               | 01:22:25 [activiti-app-rest-Executor-2] INFO
 com.activiti.api.idm.AbstractExternalIdmSourceSyncService  - All groups and users processed. Checking for
deleted groups
process_1               | 01:22:25 [activiti-app-rest-Executor-2] INFO
 com.activiti.api.idm.AbstractExternalIdmSourceSyncService  - LDAP full sync completed. Writing sync log
process_1               | 01:22:25 [activiti-app-rest-Executor-2] INFO
 com.activiti.api.idm.AbstractExternalIdmSourceSyncService  - initial-ldap-sync completed. Latest LDAP timestamp
: Tue May 26 09:24:25 UTC 2020
...
```

Now verify that it works to login with an LDAP user. Go to http://localhost:9080/activiti-app and use LDAP user **jbrown**/**password** (for this to work **clear browser data** for the last hour):

Configure ACS so it knows about the users in the Directory

In ACS you don't have to configure LDAP user sync before you can test with a directory user. ACS will create missing users on the fly if they don't exist. However, if you want the first name, last name, email etc to be available in ACS, then you still need to configure LDAP sync as normally done in ACS. Here is how to do that.

Uncomment the following properties in **my-dbp-identity-service/docker-compose-dbp/acs/alfresco-global.properties**:

```
# LDAP Sync
ldap.authentication.java.naming.provider.url=ldap://ldap:10389
ldap.authentication.active=false
ldap.synchronization.active=true
ldap.synchronization.java.naming.security.principal=uid=admin,ou=system
ldap.synchronization.java.naming.security.credentials=secret
ldap.synchronization.groupQuery=(objectclass\=groupOfNames)
ldap.synchronization.groupSearchBase=ou=RealmRoles,dc=example,dc=com
ldap.synchronization.personQuery=(objectclass\=inetOrgPerson)
ldap.synchronization.userSearchBase=ou=People,dc=example,dc=com
synchronization.syncWhenMissingPeopleLogIn=true
synchronization.syncOnStartup=true
```

You don't have to change anything in the above configuration.

For these properties to take effect you need to add the **ldap** subsystem to the authentication chain as follows:

```
# Authentication Subsystem Chain
authentication.chain=identity-service1:identity-
service,alfrescoNtlm1:alfrescoNtlm,ldap1:ldap
```

To test the LDAP sync configuration restart the ACS container as follows:

```
docker-compose-dbp mbergljung$ docker-compose stop alfresco
Stopping docker-compose-dbp_alfresco_1 ... done
docker-compose-dbp mbergljung$ docker-compose rm alfresco
Going to remove docker-compose-dbp_alfresco_1
Are you sure? [yN] y
Removing docker-compose-dbp_alfresco_1 ... done
docker-compose-dbp mbergljung$ docker-compose up -d --no-deps alfresco

Creating docker-compose-dbp_alfresco_1 ... done
```

Note. you need to fully remove the container for these changes to take effect.

In the original console window where you started the DBP you should see the logs for the ACS starting up again. Look for logs such as:

```
...
alfresco_1          | 2020-05-27 08:25:44,848  INFO  [management.subsystems.ChildApplicationContextFactory] [localhost-
startStop-1] Starting 'Authentication' subsystem, ID: [Authentication, managed, ldap1]
alfresco_1          | 2020-05-27 08:25:45,094  WARN  [authentication.ldap.LDAPInitialDirContextFactoryImpl] [localhost-
startStop-1] LDAP server supports anonymous bind ldap://ldap:10389
alfresco_1          | 2020-05-27 08:25:45,668  INFO  [management.subsystems.ChildApplicationContextFactory] [localhost-
startStop-1] Startup of 'Authentication' subsystem, ID: [Authentication, managed, ldap1] complete
...
alfresco_1          | 2020-05-27 08:26:00,334  INFO  [management.subsystems.ChildApplicationContextFactory] [localhost-
startStop-1] Starting 'Synchronization' subsystem, ID: [Synchronization, default]
alfresco_1          | 2020-05-27 08:26:00,857  INFO  [security.sync.ChainingUserRegistrySynchronizer] [localhost-startStop-
1] Synchronizing users and groups with user registry 'ldap1'
alfresco_1          | 2020-05-27 08:26:01,611  INFO  [security.sync.ChainingUserRegistrySynchronizer] [localhost-startStop-
1] Retrieving all groups from user registry 'ldap1'
alfresco_1          | 2020-05-27 08:26:01,797  INFO  [security.sync.ChainingUserRegistrySynchronizer] [localhost-startStop-
1] Synchronization,Category=directory,id1=ldap1,id2=1 Group Analysis: Commencing batch of 2 entries
alfresco_1          | 2020-05-27 08:26:01,845  INFO  [security.sync.ChainingUserRegistrySynchronizer] [localhost-startStop-
1] Synchronization,Category=directory,id1=ldap1,id2=1 Group Analysis: Processed 2 entries out of 2. 100% complete. Rate:
41 per second. 0 failures detected.
```

```
startStop-1] Starting 'Search' subsystem, ID: [Search, managed, solr6]
alfresco_1              | 2020-05-27 08:26:02,885  INFO  [management.subsystems.ChildApplicationContextFactory] [localhost-
startStop-1] Startup of 'Search' subsystem, ID: [Search, managed, solr6] complete
alfresco_1              | 2020-05-27 08:26:03,252  INFO  [security.sync.ChainingUserRegistrySynchronizer] [localhost-startStop-
1] Synchronization,Category=directory,id1=ldap1,id2=3 Group Creation and Association Deletion: Processed 2 entries out of
2. 100% complete. Rate: 1 per second. 0 failures detected.
alfresco_1              | 2020-05-27 08:26:03,253  INFO  [security.sync.ChainingUserRegistrySynchronizer] [localhost-startStop-
1] Synchronization,Category=directory,id1=ldap1,id2=3 Group Creation and Association Deletion: Completed batch of 2
entries
alfresco_1              | 2020-05-27 08:26:03,259  INFO  [security.sync.ChainingUserRegistrySynchronizer] [localhost-startStop-
1] Retrieving all users from user registry 'ldap1'
alfresco_1              | 2020-05-27 08:26:03,287  INFO  [security.sync.ChainingUserRegistrySynchronizer] [localhost-startStop-
1] Synchronization,Category=directory,id1=ldap1,id2=6 User Creation and Association: Commencing batch of 2 entries
alfresco_1              | 2020-05-27 08:26:06,719  INFO  [security.sync.ChainingUserRegistrySynchronizer] [localhost-startStop-
1] Synchronization,Category=directory,id1=ldap1,id2=6 User Creation and Association: Processed 2 entries out of 2. 100%
complete. Rate: 0 per second. 0 failures detected.
alfresco_1              | 2020-05-27 08:26:06,720  INFO  [security.sync.ChainingUserRegistrySynchronizer] [localhost-startStop-
1] Synchronization,Category=directory,id1=ldap1,id2=6 User Creation and Association: Completed batch of 2 entries
alfresco_1              | 2020-05-27 08:26:06,725  INFO  [security.sync.ChainingUserRegistrySynchronizer] [localhost-startStop-
1] Finished synchronizing users and groups with user registry 'ldap1'
alfresco_1              | 2020-05-27 08:26:06,725  INFO  [security.sync.ChainingUserRegistrySynchronizer] [localhost-startStop-
1] 2 user(s) and 2 group(s) processed
```

We will test this config by logging into the ADF app as **jbrown** user.

Test ADF App with Directory user

It should now be possible to login with **jbrown**/**password** at http://localhost:4200 and access both process and content components.

To test that it worked upload a file to the **jbrown** users home folder: