Un [**certificat électronique**](https://www.certeurope.fr/produits/certificat-electronique/) a pour objet de certifier une **clé publique**. Pour que celle-ci soit valide, il faut être certain de l’identité de son détenteur : si vous naviguez sur un site web qui vous demande de remplir un formulaire, vous souhaitez vous assurer que l’adresse URL est associée au bon propriétaire, et non à un tiers malveillant qui chercherait à récupérer des données personnelles. Ce certificat permet d’associer une clé publique à une entité (personne ou entreprise) et de garantir qu’elle lui appartient bel et bien. Comme une **carte d’identité digitale**. Pour cela, vous devez faire confiance à une autre entité, légitimement appelée **« tiers de confiance »**, qui s’occupe de certifier une clé publique et de valider l’identité du détenteur. Les **Autorités de Certification** (comme CertEurope) en font partie : elles délivrent des certificats, leur assignent un délai de validité, et peuvent les révoquer en cas de problème (*voir, plus bas, la partie consacrée aux révocations*).

keytool -genkeypair -alias larbotech -keyalg RSA -keysize 4096 -sigalg SHA256withRSA -dname "cn=larbotech.com,ou=larbotechou,dc=larbotech,dc=com" -keypass changeit -startdate "2022/09/11 00:00:00" -validity 365 -storetype PKCS12 -storepass changeit -keystore larbotech.p12

C'est quoi un keystore ? Un magasin de clef ... oui, c'est un container de clef privée/publique et de certificats. Une sorte de fourre-tout dans lequel vous pouvez associer les types d'objets cités précédemment autour d'un alias. Bien entendu, vous pouvez créer autant d'alias que vous voulez.

La commande ci-dessus vous permet de créer autour d'un alias nommé <mykeyalias> dans un keystore nommé <mykeystore.extension>:

* une paire de clef privée/publique de type RSA taille 4096,
* un certificat x509 valide de 1 an lié à la clef publique.

Ici, je n'ai précisé le type de keystore voulu (p12). Généralement, on en trouve de deux types:

* le **JKS** (extension .jks) : [Java Keystore](https://en.wikipedia.org/wiki/Java_KeyStore), propre à Oracle, propriétaire et qui s'utilise très bien avec le langage Java. Par contre, dans d'autres languages, c'est compliqué (je pense aussi d'un point de vue licence).
* le **PKCS12** (extension .p12) : [Public-Key Cryptography Standards 12](https://fr.wikipedia.org/wiki/PKCS12), un format créé par les laboratoires RSA. Ils sont désormais tombés dans le domaine public.

Si on veut certifier notre clé publique par une [CA (Certificate Authority)](https://fr.wikipedia.org/wiki/Autorit%C3%A9_de_certification), quelle soit issue d'un tiers de confiance ou d'une CA interne, on peut utiliser la commande suivante pour créer une [CSR (Certificate Signing Request)](https://fr.wikipedia.org/wiki/Demande_de_signature_de_certificat).

Une fois envoyé à la CA de votre choix, elle va vous fournir en échange un certificat x509 signé par la clé privé de l’organisme CA, bien souvent au format *.crt*. Il vous suffira alors de le placer dans un nouvel alias de votre keystore.

**Host key fingerprint**

The fingerprint is based on the Host’s Public key, usually based on “**/etc/ssh/ssh\_host\_rsa\_key.pub**” Generally its for easy identification/verification of the host you are connecting to.

The fingerprint changed means machine you are trying to connect changed its public key(because of reinstall ssh, reinstall the os). If fingerprint of the machine changed, when connecting to this machine it would give the error “**host key verification failed**” (When connecting to this server, server returns the new RSA key fingerprint which is different with the key stored in known\_hosts file)

# Keystore vs Truststore

## Keystore

* Keystore provide credentials
* It means it stores private keys and the certificates with their public keys
* You only need this if you are a server

## Truststore

* Is use to verify credentials
* It means it stores public key certificates from third parties(which you expect to communicate with)

A CSR contains:

* **Domain Name**: Fully qualified domain name (FQDN) of the server, such as encryptionconsulting.com.
* **Organization Name**: Legal name of the organization such as ‘Encryption Consulting LLC.’
* **Organization Unit**: Division of the organization, such as IT Department, or Marketing.
* **City/Locality**: City, where the organization is located, such as Prosper.
* **State/County/Region**: such as Texas
* **Country**: such as the US
* **Email address**: Email address to contact the organization, such as [info@encryptionconsulting.com](mailto:info@encryptionconsulting.com)
* **Public Key**: The public key which will be associated with the certificate.

Extract public key from csr ; openssl req -in csr.txt -noout -pubkey -out publickey.pem

docker cp <containerId>:/file/path/within/container /host/path/target

docker exec -it 19570b4ddf1e6f7ca563cb40cdad4d24a452ddb4c96757b544c7079dd731a63f /bin/sh

docker cp 463a56f1831a85223c2b5ed:/home/wso2carbon/wso2is-5.11.0/repository/resources/security/wso2carbon.jks .

keytool -genkey -keyalg RSA -alias saml2cert -keystore keystore.jks -storepass password -validity 365 -keysize 2048

**Generate the certificate from wso2carbon.jks.**  
keytool -export -keystore wso2carbon.jks -alias wso2carbon -file wso2is-certificate.cer

Pas de mot de passe défini

Génération de la clé privée : supprimer les lignes supplémentatires -----BEGIN PRIVATE KEY-----

Création de l’image avec le tag :

docker commit 19570b4ddf1e6f7ca wso2-poc

regexp=(<https://localhost:8080/login/oauth2/code/wso2|https://localhost:8080>)

mvn spring-boot:run -Dspring-boot.run.profiles=saml2/oauth2

keytool -import -alias wso2carbon -keystore $JAVA\_HOME/lib/security/cacerts -file src/main/resources/certs/wso2is-certificate.cer

Nous allons utiliser WSO2 identity server afin de mettre en place du (Single Sign-On ou SSO) dans le cadre d’un dévéloppement en local. WSO2 Identity Server est un serveur d'authentification unifié et de gestion des droits développé depuis 2007 notamment par le Dr. Sanjiva Weerawarana, l'un des pères de la vision architecturale WS-\*. WSO2 Identity Server  repose sur la plateforme Carbon, implémentant les spécifications OSGi, commune à tous les produits de WSO2, modulaire, extensible et pouvant nativement être mis à l'échelle. La solution permet de charger les données utilisateurs depuis n'importe quelle source externe, LDAP, Active directory, JDBC, base propre ou propose un serveur Apache Directory Server intégré. Elle met à disposition un système d'authentification unifié via les protocoles OAuth 1.0 & 2.0, OpenID, SAML2 et Kerberos KDC et de propagation d'identité.

La configuration SAML avec [**WSO2 Identity Server**](https://www.chakray.com/fr/ebooks/securite-numerique-gestion-des-identites-et-des-acces/) nécessite la **version 5.11 ou ultérieure pour une utilisation optimale avec la version 2.7 de spring boot.**

Pour commencer, récupérons l’image docker :

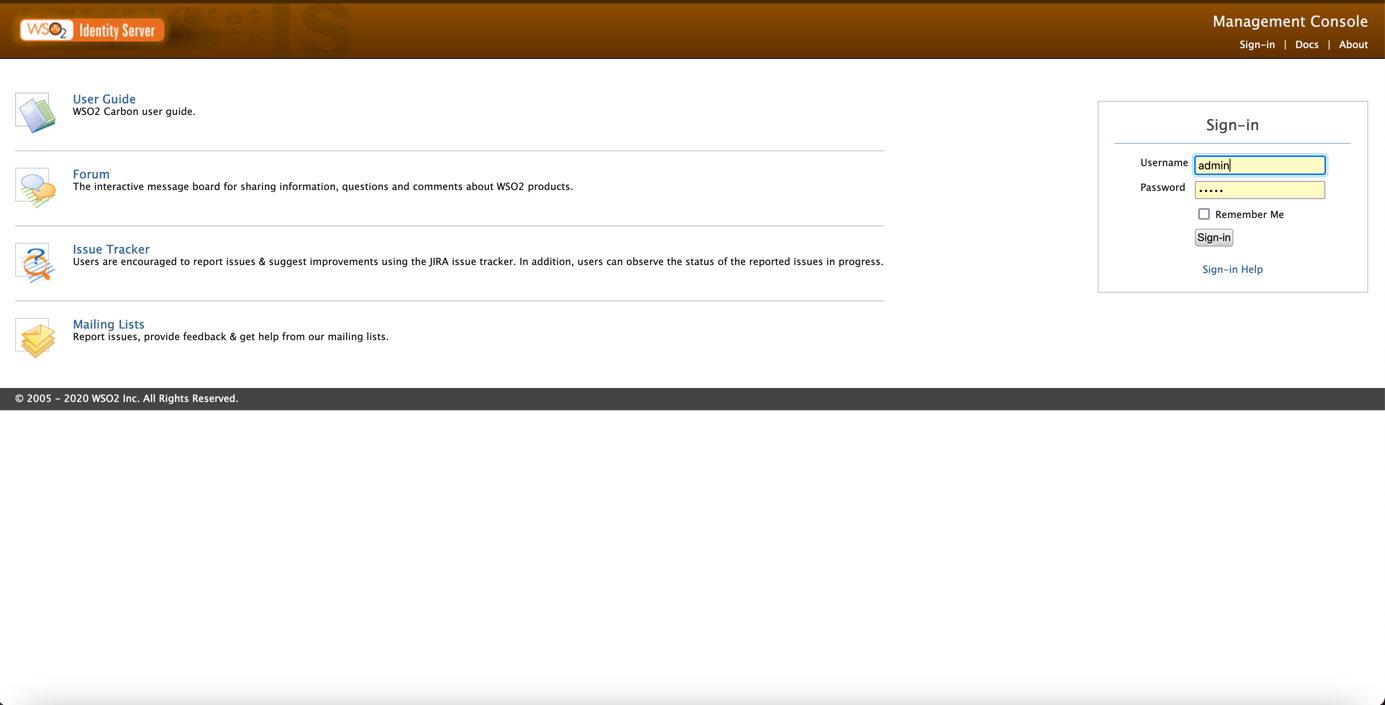
/// docker pull …

Démarrage du WSO2 identity server :

//docker run ..

Le serveur est accessible depuis : <https://localhost:9443/carbon/>

//images avec admin/admin



Connectez-vous à la console d’administration avec l’utilisateur : admin/admin

Dans la suite de ce tuto, nous aurons besoin d’un certificat permettant d’authentifier notre application (au près du serveur WSO2) et valider les requêtes SAML2 envoyées par l’application. Nous allons donc générer un certificat auto-signé.

**Création d’un keystore avec la clé privée et le certificat auto-signé :**

keytool -genkey -keyalg RSA -alias saml2cert -keystore keystore.jks -storepass password -validity 365 -keysize 2048

**Extraction du certificat du keystore :**

keytool -export -keystore keystore.jks -alias saml2cert -file certificate.cer pass:password

**Afficher le certificat :**

keytool -printcert -rfc -file certificate.cer

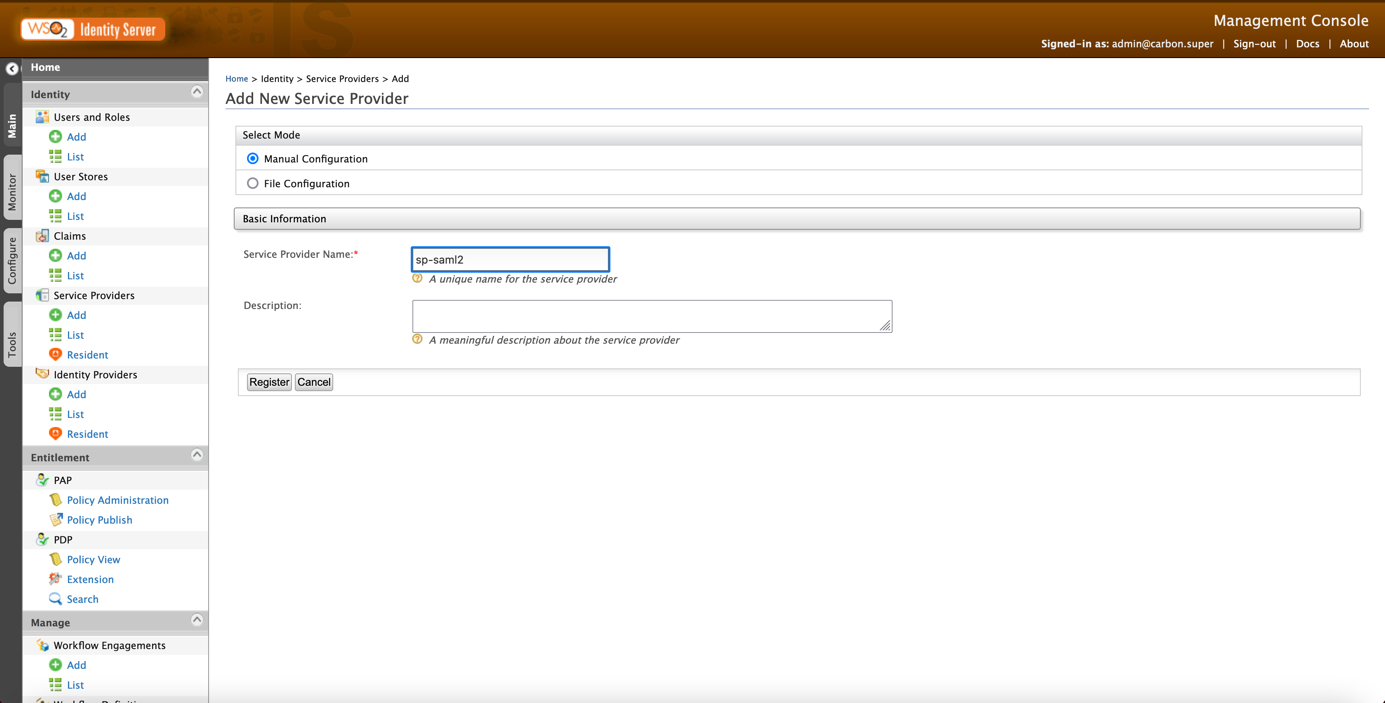
**SAML**

Le SAML (pour Security Assertion Markup Language) est un standard ouvert, basé sur XML et développé par OASIS (consortium international œuvrant pour la standardisation de formats de fichiers). Il définit le processus permettant à un utilisateur d’accéder à plusieurs applications via une authentification unique (Single Sign-On ou SSO).

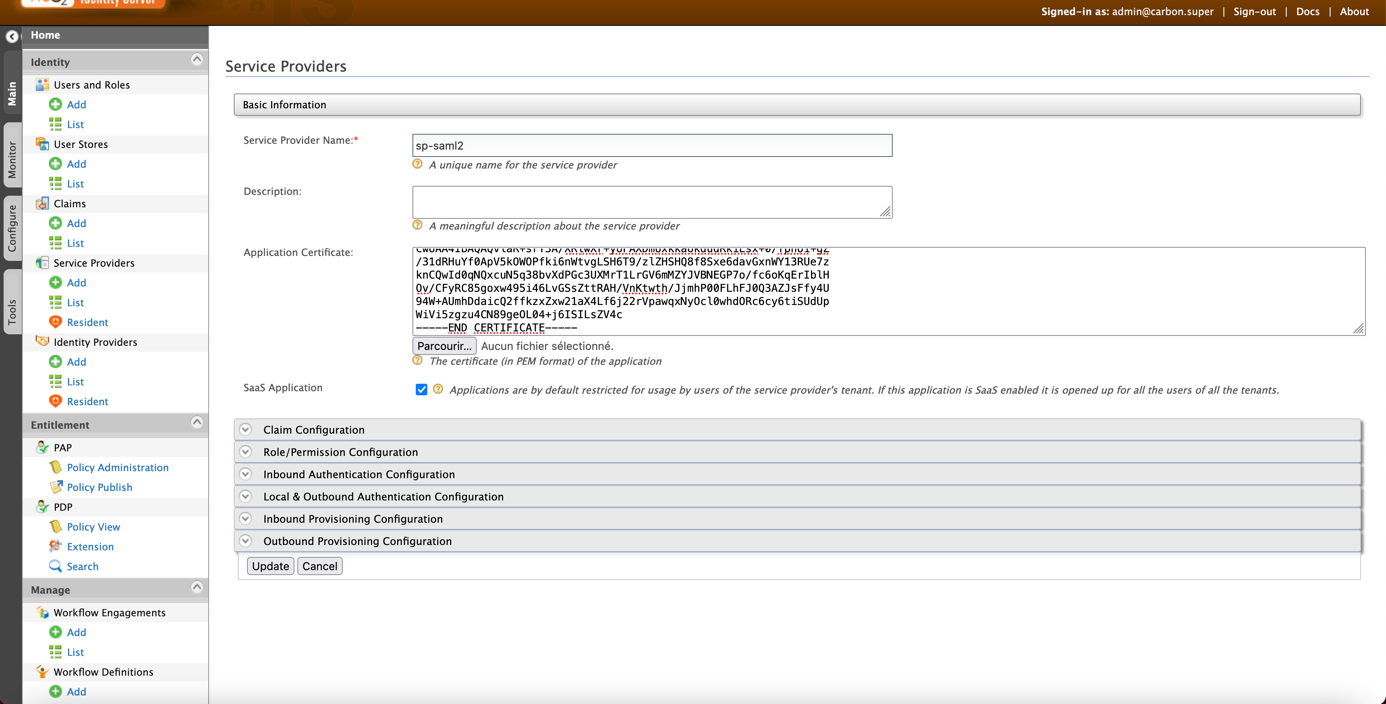
SAML2 est la version 2 du standard SAML pour l’échange d’identités d’authentification et d’autorisation entre les domaines de sécurité. Il s’agit d’un protocole basé sur XML qui utilise des jetons de sécurité contenant des assertions et des informations utilisateur.

Création de service provider SAML2

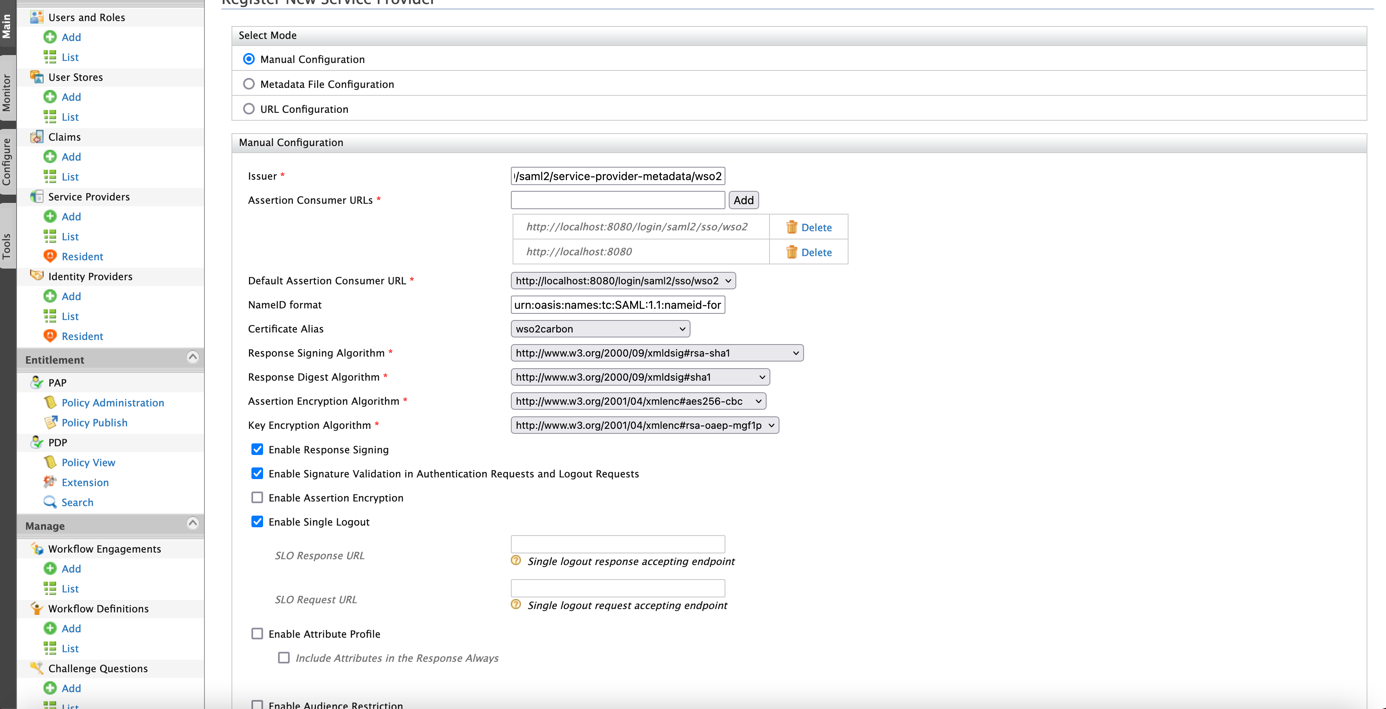
Connectez-vous à la console d’administration avec l’utilisateur : admin/admin : Identity > Service Providers > Add. Renseigner le nom du Provider puis cliquer sur Register :



La page **Basic Information** s’affiche. Renseigner le certificat précédemment généré dans **Application Certificat** et cocher la case **SaaS Application**



Dans **Inbound Authentication Configuration” -> SAML2 Web SSO Configuration, cliquer sur le lien Configure :**



Renseigner les valeurs **Issuer** et **Assertion Consumer URLs** puis cliquer sur Register :

* + **Issuer** : <http://localhost:8080/saml2/service-provider-metadata/wso2>
  + **Assertion Consumer URLs** :
    - <http://localhost:8080/login/saml2/sso/wso2>
    - <http://localhost:8080>

Lancer l’application via la commande :

mvn spring-boot:run -Dspring-boot.run.profiles=saml2

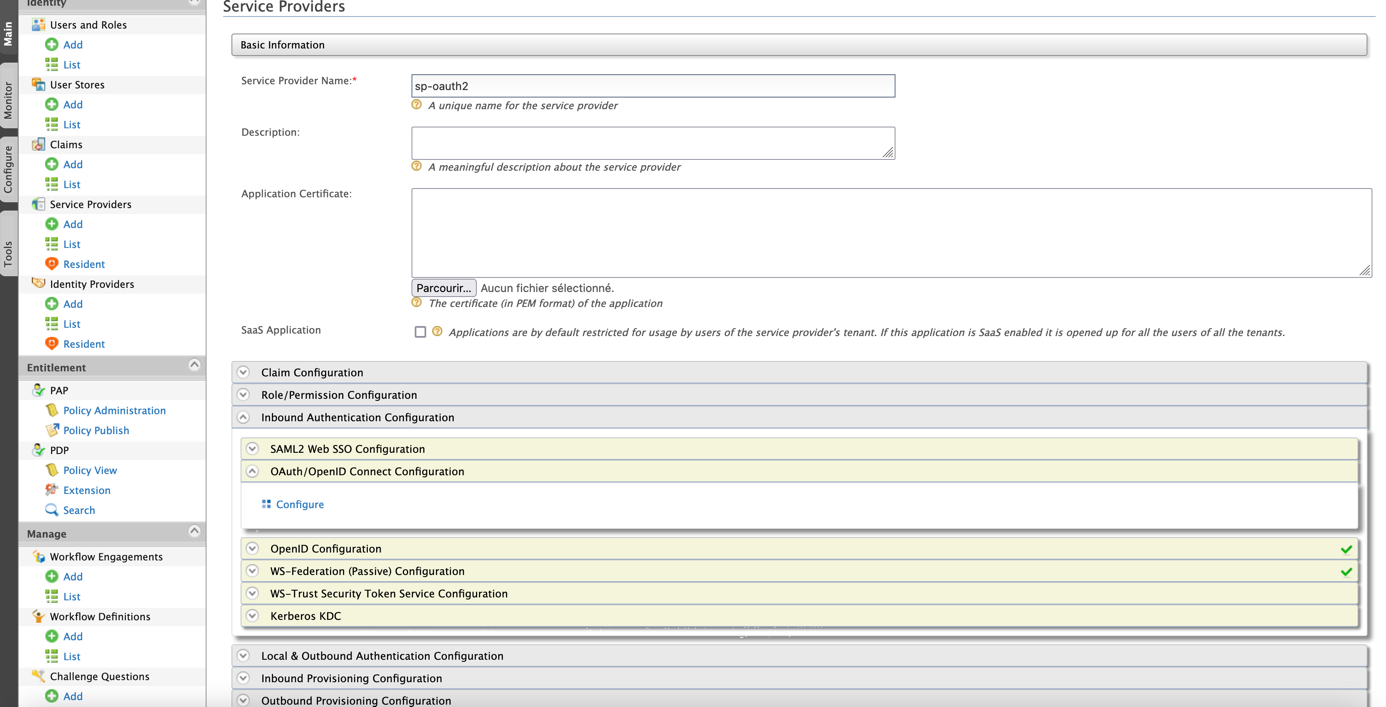
L’application est accessible : <http://localhost:8080/login>

OAuth2

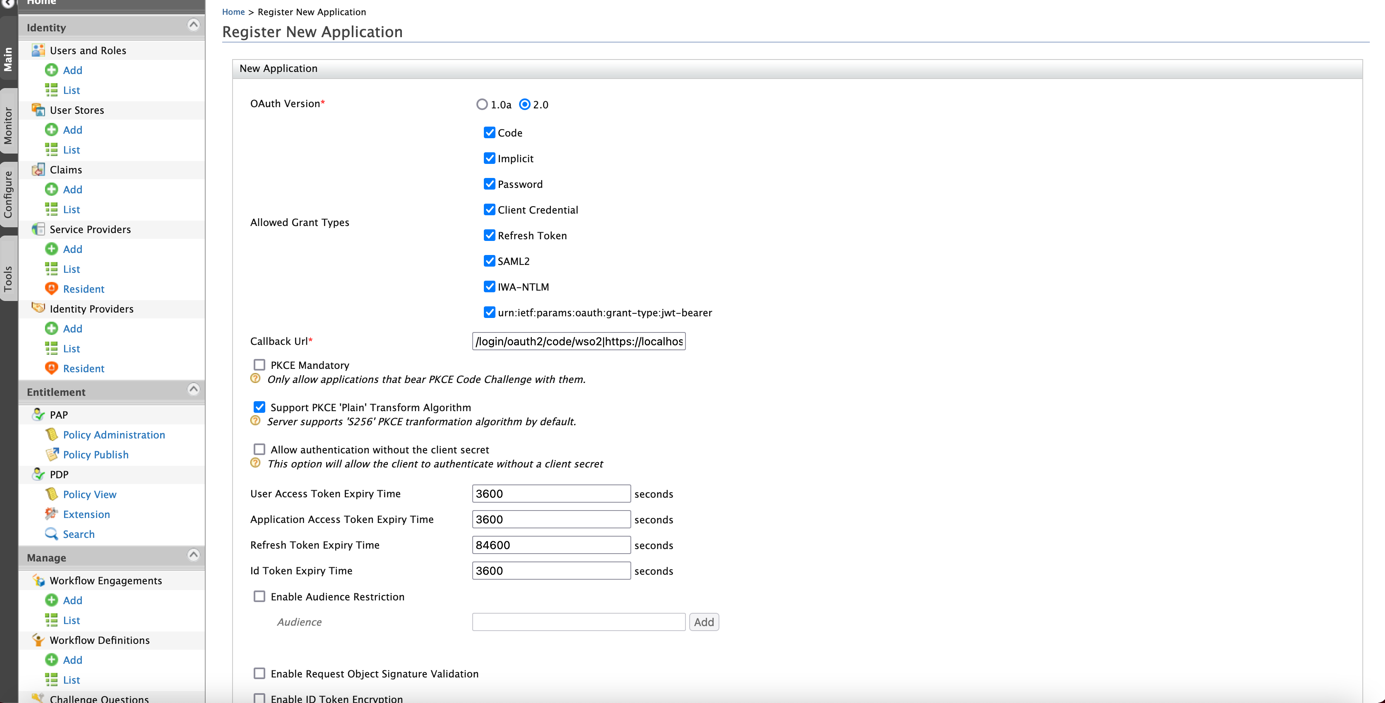
OAuth est un standard de **délégation d’autorisation**. Pour être plus précis, OAuth, dans sa version 2, est un standard ouvert basé sur JWT (JSON Web Token, standard permettant l’échange sécurisé de jetons grâce à une vérification de l’intégrité des données) permettant à des services d’accéder à des données d’autres services via une **autorisation** du propriétaire de celles-ci et **sans communication de mot de passe**.

Création de service provider OAuth2

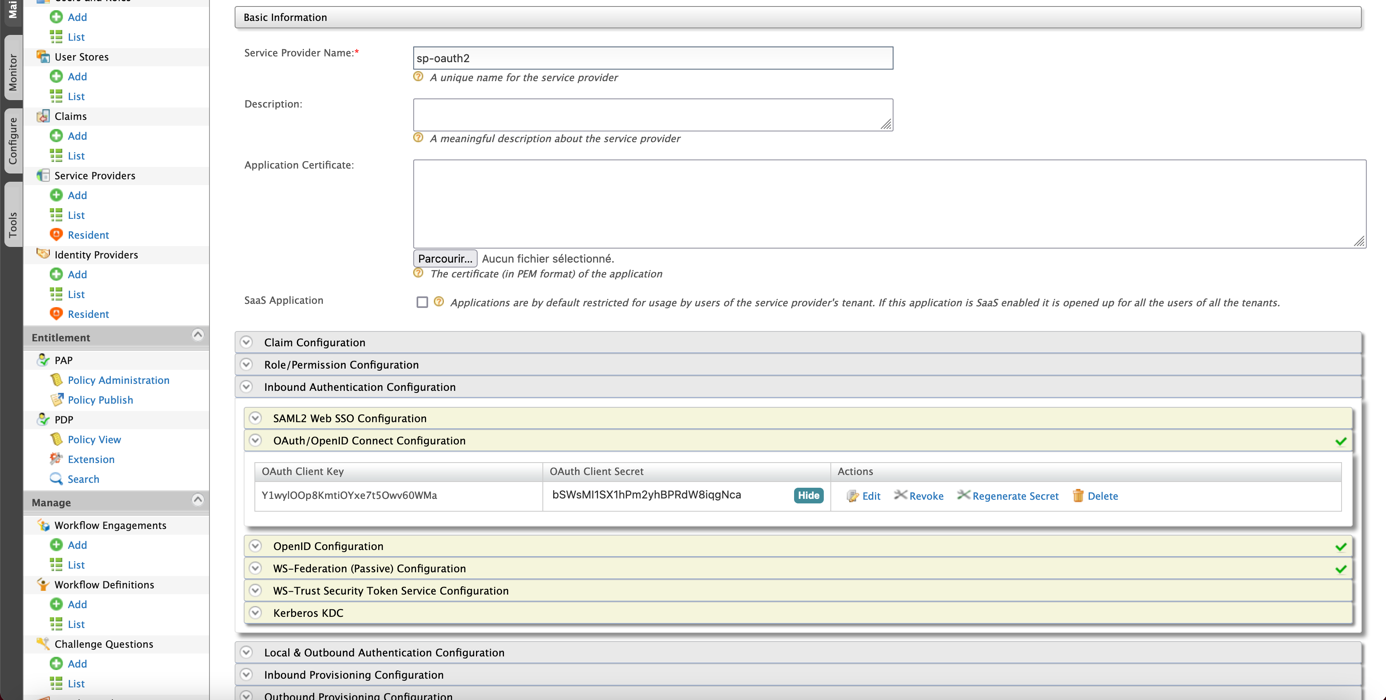
Connectez-vous à la console d’administration avec l’utilisateur : admin/admin : Identity > Service Providers > Add. Renseigner le nom du Provider puis cliquer sur Register :



Dans **Inbound Authentication Configuration” -> OAuth/OpenID Connect Configuration, cliquer sur le lien Configure :**

****

Dans **Call back url** renseiner cette valeur expression régulière puis cliquer sur **Update**: regexp=(<https://localhost:8080/login/oauth2/code/wso2|https://localhost:8080>)



Noter bien les informations : **OAuth Client Key** et **OAuth Client Secret** pour la configuration coté application.

Lancer l’application via la commande :

mvn spring-boot:run -Dspring-boot.run.profiles=oauth2

L’application est accessible : [http://localhost:8080](http://localhost:8080/login)