Installation et configuration de CAS

Ils ont tellement bien choisi leur nom cette « bande d'intelligents ». Quand tu tapes **cas** dans un moteur de recherche, tu obtiens tout sauf le CAS que tu recherches.

CAS, c'est **Central Authentication Service**. C'est un système/service d'authentification central (juste central, et pas de gauche, ni de droite). C'est un peu comme les histoires de Shibboleth (http://www.yerbynet.com/Cours/ShibbolethInstallation.pdf) ou de ADFS (http://www.yerbynet.com/Cours/ADFS.pdf). Des applications vont déléguer à CAS leur service d'authentification. La différence entre CAS (version récente 5+), et Shibboleth, c'est que CAS supporte plus de mécanismes d'authentification que Shibboleth. Shibboleth est spécialement conçu pour faire du SAML, tandis que CAS, initialement conçu pour faire son protocole natif CAS, supporte maintenant bien plus de méthodes (CAS natif, SAML2, OpenID Connect, OAuth2, MFA, Authentification adaptative, …).

Qu'est ce qu'ils ont en commun ? Support du Web SSO, SAML, et surtout la difficulté de l'installation et de la prise en main. CAS est encore plus compliqué à installer que Shibboleth, avec une documentation qui ressemble à un labyrinthe créé exprès pour ne pas qu'on réussisse à s'en tirer. Même en ayant peur des mots, on n'hésiterait pas à dire :

« C'est une documentation de merde, et sauf respect pour les chiens (qui sauraient lire), c'est une nourriture pour chat. C'est à se demander si les chèvres savent écrire! C'est l'une des pires documentations avec lesquelles j'ai été confronté. »

Mais je ne l'ai pas dit.

Je vous livre quelques extraits de discussions sur le sujet de la documentation de CAS.

Extrait1 :

« As a new user of CAS, I'd like to voice my opinion that the official documentation of how one can get started with CAS is just awful. By this I mean not the lack of it, but rather how indirect, not step-by-step it is. Clarity could often be improved too.

In the end I managed to do what I hoped for, ie investigate CAS locally as an SSO solution, for which I needed to (1) run CAS server locally, (2) connect and authenticate using a simple CAS client locally, (3) run the service management app. However, the difficulty I had at most steps of getting it all to work make me really want to use something else even if I have to implement parts of it from scratch.. »

Extrait2 :

« While it is true that CAS is not a turnkey solution, and it is also true that the documentation has vastly improved since the early days of CAS, I think it is fair to say there is room for improvement in the CAS documentation.

The criticism that the documentation is somewhat lacking in terms of reliable tutorials seems to be fairly accurate. »

Extrait3 :

« As I see after almost 2 years documentation still don't getted better ...
... there are no some 1-2-3-Specs to finish even HelloWorld with CAS. But the release number grows extremely. 3.6, 4.0-1-2-3, 5.0-1-2-3, 6.0.

So - looks like the developers stacked in the similar way as users :)
I would wait till first release with suffix "STABLE", before start to use IT. After a week of rolling over LinkDoc-to-LinkDoc-to-LinkDoc I give Up. »

Bref ! Ils ont quand même fini par écrire un logiciel qui arrive à faire (parfois) ce qu'on lui demande.

On va donc tenter de décrire comment est-ce qu'on l'installe et comment il fonctionne.

1- Le paradigme d'installation de CAS

Il faut déjà commencer par expliquer comment va s'installer CAS.

Ce n'est pas un logiciel qu'on installe avec rpm, yum, apt, … ni un tar.gz qu'on télécharge, qu'on décompresse, puis on suit une procédure d'installation.

Avec CAS, il faut « programmer » son installation. Vous allez avoir un fichier xml, ou autre dans lequel vous allez décrire les modules de votre installation de CAS, leurs dépendances, ainsi que les repositories où aller les chercher. Par la suite, vous lancez une commande (compilation) qui va vous télécharger tous les requis et générer un .war (java), que vous exécuterez avec un conteneur tel que Tomcat, ou bien le lancerez directement en ligne de commande.

Pourquoi cela? Parce qu'il y a trop trop de modules et de fonctionnalités, trop de cas d'usage, de backend, … si bien que ce n'est pas une bonne idée de faire un paquetage qui contiendrait toutes les combinaisons pour le rendre disponible en téléchargement. De toute façon, ça ne marcherait pas, à moins d'avoir un fichier de config par défaut très balaise qui va avec, sans oublier que certaines fonctionnalités/méthodes peuvent entrer conflits et CAS ne démarrerait pas.

À noter que les modules qu'on rajoute ne viennent pas tous avec des comportements par défaut définis dans le code. Ce qui signifie que pour la plupart des modules qu'on rajoute, il faut aller chercher les lignes de configuration qui vont avec (dans le labyrinthe de documentation), puis les rajouter dans le fichier de configuration avant que CAS ne soit capable de démarrer de nouveau.

Donc, la technique c'est de compiler notre fichier de définition de notre CAS, puis d'exécuter le fichier généré.

Parlant de définition de notre CAS, il existe 2 méthodes : la méthode gradle, et la méthode maven.

Dans un cas, on va définir notre CAS dans un fichier gradle, et dans l'autre un fichier pom.xml.

Il existe des fichiers minimalistes qu'on peut retrouver sur Internet pour ne pas partir de 0.

Cependant, à terme, il n'y aura plus la méthode maven. Dommage, car c'est ce que j'ai utilisé, et c'est celle que je vais décrire, mais la transition n'est pas bien compliquée, car il suffit en théorie d'ajouter dans gradle les mêmes modules et dépendances qui se retrouvent dans le pom. La syntaxe est différente, mais la correspondance existe.

Résumé :
- On remplit notre fichier de définition (gradle ou maven),
- on compile,
- on récupère le war généré,
- on crée le fichier de config adapté aux modules qu'on a précisé dans notre fichier de définition,
- on crée les requis (base de données, LDAP, …),
- on lance notre war (avec notre fichier de conf).

Il est donc très important d'aller prudemment et de ne pas rajouter tous les modules en même temps. Il vaut mieux y aller un module at a time, puis vérifier qu'on est capable de le faire marcher avant d'en rajouter un autre, sauf bien sûr quand on a des interdépendances.

2- Que va contenir le CAS que je suis entrain de décrire?

Comme déjà expliqué, j'utilise la méthode **maven**, avec mon fichier **pom.xml** pour la définition de mes modules (des dépendances) de mon CAS.

Je n'utilise pas de Tomcat, ni de Wildfly, ni aucun autre conteneur pour exécuter mon .war. Je lance donc mon .war en ligne de commande, il utilise le embendded-Tomcat que j'aurai spécifié dans la définition de mon CAS. Il faut dire que pour un logiciel déjà complexe, je ne souhaite pas rajouter de la complexité supplémentaire avec d'autres logiciels non nécessaires.

D'ailleurs, les créateurs de CAS expliquent qu'on peut bien le lancer en ligne de commande. Pour une fois, je les cite :

*« **Production Quality***

All servlet containers presented here, embedded or otherwise, aim to be production ready. This means that CAS ships with useful defaults out of the box that may be overridden, if necessary and by default, CAS configures everything for you from development to production in today's platforms. In terms of their production quality, there is almost no difference between using an embedded container vs. an external one.

Embedded

Note that CAS itself ships with a number of embedded containers that allow the platform to be self-contained as much as possible. These embedded containers are an integral part of the CAS software, are maintained and updated usually for every release and surely are meant to and can be used in production deployments. You DO NOT need to, but can if you want to, configure and deploy to an externally configured container. »

Cela a suffi pour me convaincre, et comme on dit chez nous, « on ne cherche pas palabre où y en n'a pas ».

La version de CAS sera CAS-5.3.X. Le X peut changer, mais pas le 5.3. Si le 5.3 change pour devenir par exemple 5.2 ou 6.0, il y aura forcément des erreurs de config (des paramètres dont le nom change), et probablement, même des erreurs de compilation (des dépendances non satisfaites).

Je stocke certains éléments de configuration dans une base de données MySQL (notamment les services ou applications qui utiliseront l'authentification CAS, les tickets, ...). Donc, j'aurai besoin d'installer un mysql. J'ai vu beaucoup d'installations où les services sont dans des fichiers json, et les tickets je ne sais plus où, en mémoire par exemple. On peut donc trouver des installations où on n'a pas besoin du support MySQL.

Mes usagers/credentials viennent d'un annuaire Active Directory. J'aurai donc besoin de me brancher en LDAP à mon AD. Ça nécessite un module pour ça.

Étant donné que je me suis retiré un peu de complexité avec l'affaire de Tomcat, je m'en rajoute un chouïa en branchant ce CAS avec un Shibboleth, donc module supplémentaire. Pour le comme on dit, cette fois chez vous, « trop de viande n'a jamais gâté la sauce ».

On va aussi supporter du OAuth2, et on sera un IDP SAML => autant de dépendances à rajouter, puis à configurer.

Notre CAS va rouler sur le port 9443, puis on pourra le NATer depuis le 443 avec iptables/firewalld. On aurait aussi pu mettre un Apache ou Nginx en frontal et configurer un reverse proxy, mais pour ce cas-ci, je fais bien plus confiance à iptables qu'à Apache.

On utilisera une machine CentOS7, avec SELinux activé en mode Enforcing (parce qu'on sait comment ça marche, sinon Permissive), et un java version 8.

3- Les prérequis

Bien entendu, il faut déjà avoir une machine GNU/Linux (CentOS7). Sinon, bug du lecteur.

3-a/ MySQL (Pas nécessaire dans toutes les installations)

Installez MariaDB (***yum install mariadb-server mariadbmysql-connector-java***) et sécurisez le un minimum avec la commande ***mysql_secure_installation***. Puis, ajoutez la ligne ci-dessous dans le fichier */etc/my.cnf*:

```
bind-address=127.0.0.1
```

Vous pouvez créer le fichier */root/.my.cnf* et ajouter les credentials du root de mysql.

Cette partie est totalement optionnelle, et permet juste de ne pas avoir à entrer le mot de passe root à chaque fois qu'on veut faire des requêtes MySQL.

```
[client]
user=root
password=motdepasse_mysql_de_root
```

Suivi d'une relance de MySQL.

```
systemctl enable mariadb
systemctl start mariadb
```

On va tout de suite créer la BD qui va nous servir lors de notre installation de CAS :

```
[root@cas5 ~]# mysql
MariaDB [(none)]> create database casdb;
MariaDB [(none)]> grant all on casdb.* to casuser@'127.0.0.1' identified by 'Motdepasse';
```

3-b/ Installation de NTP

CAS doit générer, valider, et gérer des tickets dont la validité dure un certain délai. Il est donc très important que le temps soit très à temps.

Installez ntp (***yum install ntp***) et faites le écouter seulement sur sa loopback en ajoutant dans le fichier «/etc/ntp.conf» les lignes suivantes :

```
interface ignore wildcard
interface listen 127.0.0.1
interface listen::1
```

Relance de NTP et le tour est joué.

3-c/ Java

C'est simple : sans java, CAS ne pourra pas rouler. La version requise pour CAS-5.3 est Java8.

```
[root@cas5 ~]# yum install java-1.8.0-openjdk-devel
```

```
[root@cas5 ~]# rpm -qa | grep openjdk
java-1.8.0-openjdk-devel-1.8.0.191.b12-1.el7_6.x86_64
java-1.8.0-openjdk-1.8.0.191.b12-1.el7_6.x86_64
java-1.8.0-openjdk-src-1.8.0.191.b12-1.el7_6.x86_64
java-1.8.0-openjdk-headless-1.8.0.191.b12-1.el7_6.x86_64
java-1.8.0-openjdk-javadoc-1.8.0.191.b12-1.el7_6.noarch
```

```
[root@cas5 ~]# java -version
openjdk version "1.8.0_191"
OpenJDK Runtime Environment (build 1.8.0_191-b12)
OpenJDK 64-Bit Server VM (build 25.191-b12, mixed mode)
```

3-d/ Maven

Ça peut s'installer avec **yum** (enfin, je crois). C'est aussi inclus dans certaines archives de CAS téléchargées. On peut enfin le télécharger directement chez Apache.org (https://maven.apache.org/download.cgi).

Exemple :

```
[root@cas5 ~]# wget http://apache.mirror.iweb.ca/maven/maven-3/3.6.0/binaries/apache-maven-3.6.0-bin.tar.gz
[root@cas5 ~]# tar zxvf apache-maven-3.6.0-bin.tar.gz
[root@cas5 ~]# mv apache-maven-3.6.0 /usr/local/bin/maven
[root@cas5 ~]# chmod +x /usr/local/bin/maven/bin/mvn
[root@cas5 ~]# ln -s /usr/local/bin/maven/bin/mvn /usr/local/bin/mvn
[root@cas5 ~]# mvn -version
 Apache Maven 3.6.0 (97c98ec64a1fdfee7767ce5ffb20918da4f719f3; 2018-10-24T14:41:47-04:00)
```

```
Maven home: /usr/local/bin/maven
Java version: 1.8.0_191, vendor: Oracle Corporation, runtime: /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.191.b12-1.el7_6.x86_64/jre
Default locale: fr_CA, platform encoding: UTF-8
OS name: "linux", version: "3.10.0-957.5.1.el7.x86_64", arch: "amd64", family: "unix"
```

3-e/ Un certificat SSL valide

Soit vous utilisez letsencrypt (il y a des pour et des contre), soit vous allez chez un vendeur de certificat. Mais avant, il faut déjà générer le CSR.

Le nom DNS sur lequel sera accessible mon CAS est **cas5.yerbynet.com**.

Création du CSR et de la clé privé :

```
openssl req -sha256 -out cas5.yerbynet.com.csr -new -newkey rsa:4096 -nodes -keyout cas5.yerbynet.com.key
```

Génération du certificat : par exemple chez http://www.yerbynet.com/Cours/LetSEncrypt.html

On se retrouve avec 2 fichiers importants : ***cas5.yerbynet.com.key*** *et* ***cas5.yerbynet.com.crt (clé + certificat)*** et possiblement un 3è fichier qui est le Bundle (en fonction de l'autorité de certification et du type de certificat acheté). Supposons donc qu'on ait un 3è fichier ***ChainBundle2.crt***.

Ces 3 fichiers vont nous servir par la suite.

4- Téléchargement, compilation, et exécution de CAS

On va utiliser la méthode maven : https://apereo.github.io/cas/5.3.x/installation/Maven-Overlay-Installation.html

Il faudra donc télécharger les sources depuis https://github.com/apereo/cas-overlay-template/.

Ça peut se faire avec ***git***, ou en téléchargement le zip (faire attention à la branche choisie).

On installe dans ***/opt/***.

```
cd /opt/
wget https://github.com/apereo/cas-overlay-template/archive/5.3.zip
unzip cas-overlay-template-5.3.zip
cd /opt/cas-overlay-template-5.3/
```

On trouvera un fichier pom.xml dans le répertoire créé. C'est ce fichier pom.xml qu'il faudra modifier, compiler pour obtenir notre CAS avec les fonctionnalités dont on a besoin.

La commande pour compiler et générer le .war est ***mvn clean package***.

Exemple :

```
cd /opt/cas-overlay-template-5.3/
/usr/local/bin/maven/bin/mvn clean package
```

Le résultat de la compilation, si elle s'est bien passée se retrouvera dans **target/** (exemple : **/opt/cas-overlay-template-5.3/target/**).

```
[root@cas5 cas-overlay-template-5.3]# ls -l target/
 total 418556
 drwxr-xr-x. 5 root root       48  5 fév 14:46 cas
 -rwxr--r--. 1 root root 274899440  5 fév 14:46 cas.war
 -rw-r--r--. 1 root root 153695438  5 fév 14:46 cas.war.original
 drwxr-xr-x. 2 root root       28  5 fév 14:46 maven-archiver
 drwxr-xr-x. 3 root root       18  5 fév 14:46 war
```

C'est le fichier **cas.war** généré, qui une fois exécuté assure notre service CAS.

On pourra exécuter CAS avec la commande suivante :

```
java -jar /opt/cas-overlay-template-5.3/target/cas.war -Xms2.9G -Xmx3G -server
```

Cependant, il nous faut une config avant d'être capable de le faire. CAS s'attend à avoir sa config dans */etc/cas/config/cas.properties,* et */etc/cas/config/log4j2.xml*. Il faut donc lui donner

ces fichiers. On ne va pas les construire à la main de zéro. Car il existe une config exemple dans l'archive téléchargée.

```
cd /opt/cas-overlay-template-5.3/etc/
cp -va cas /etc/
'cas' -> '/etc/cas'
'cas/config' -> '/etc/cas/config'
'cas/config/cas.properties' -> '/etc/cas/config/cas.properties'
'cas/config/log4j2.xml' -> '/etc/cas/config/log4j2.xml'
```

Par la suite, on pourra rouler la commande de lancement de CAS (Exemple : ***java -jar cas.war***).

Notre travail consistera donc à rajouter progressivement nos modules/fonctionnalités dans le fichier de définition **pom.xml**, à compiler pour obtenir le fichier exécutable **cas.war**, à ajouter les config correspondantes dans le fichier de configuration **cas.properties,** puis à exécuter notre CAS et vérifier que la fonctionnalité en question marche comme souhaité.

5- Support https et le keystore JKS

Pour un système d'authentification, il n y a aucune raison de le rouler en http sauf pour jouer. Pas besoin de rajouter de dépendances dans pom.xml, c'est déjà pris en charge par défaut. Il faut mettre en place la configuration qui l'active.

Voici la config correspondante dans */etc/cas/config/cas.properties* :

```
server.port=9443
cas.server.name=https://cas5.yerbynet.com
cas.server.prefix=https://cas5.yerbynet.com/cas
cas.host.name=cas5.yerbynet.com
server.ssl.enabled=true
server.ssl.keyStore=file:///etc/cas/ssl/cas5.yerbynet.com.jks
server.ssl.keyStorePassword=mot_de_passe
server.ssl.keyPassword=mot_de_passe
```

Les éléments de la configuration qui n'existent pas sont à créer. Ici, il s'agit du fichier */etc/cas/ssl/cas5.yerbynet.com.jks.*

3 fichiers dont nous avons parlés dans les requis vont nous permettre de créer ce keystore JKS (**cas5.yerbynet.com.key, cas5.yerbynet.com.crt, ChainBundle2.crt)**.

mkdir /etc/cas/ssl/
(copier les 3 fichiers dans /etc/cas/ssl/)
cd /etc/cas/ssl/
openssl pkcs12 -export -in cas5.yerbynet.com.crt -inkey cas5.yerbynet.com.key -certfile ChainBundle2.crt -out cas5.yerbynet.com.p12
keytool -importkeystore -srckeystore cas5.yerbynet.com.p12 -keypass MOTDEPASSE -storepass *mot_de_passe* ***-srcstoretype pkcs12 -destkeystore cas5.yerbynet.com.jks -deststoretype JKS***
keytool -changealias -keystore cas5.yerbynet.com.jks -alias 1 : Mettre cas5.yerbynet.com

Après une relance de CAS, CAS devrait être accessible sur *https://cas5.yerbynet.com:9443/cas*.

Mais il faut ouvrir le firewall pour se faire :

firewall-cmd --permanent --add-port=9443/tcp
firewall-cmd --reload

Si on veut que cela soit accessible sur 443 (sans le 9443), on va tout de suite mettre les règles de NAT qu'il faut.

firewall-cmd --permanent --add-forward-port=port=443:proto=tcp:toport=9443
firewall-cmd --permanent --direct --add-rule ipv4 nat OUTPUT 0 -p tcp -o lo --dport 443 -j REDIRECT --to-ports 9443
firewall-cmd --permanent –add-service=https
firewall-cmd --reload

CAS sera ainsi disponible sur *https://cas5.yerbynet.com/cas*.

L'utilisateur par défaut pour faire des tests à ce niveau est **casuser** et son mot de passe est **Mellon**.

Si vous êtes fatigués de lire comme moi, je vous conseille quand même de continuer pendant que je vais prendre une pause fumée. Ça fait quand même quelques heures que j'écris sur CAS, et je commence à trouver ça intéressant, ce qui signifie que quelque chose ne tourne pas rond.
Blague à part, si vous avez pu vous rendre jusqu'ici, et que vos tests marchent, c'est que vous avez conquis CAS. Le reste, c'est une question de modules à rajouter en fonction de ses propres besoins et de bien les configurer en fonction de son environnement.
À retenir : Je n'ai pas besoin d'un module, je ne le rajoute pas, ce qui me fait une config en moins, et une potentielle faille (ou bug) évitée.

Ajoutons ces options que je recommande (je ne sais plus trop pour quelle raison d'ailleurs) avant d'aller de l'avant :

```
cas.monitor.endpoints.enabled=true
cas.monitor.endpoints.sensitive=false
cas.adminPagesSecurity.ip=Regex_Ips_Autorisées (exemple :^196\.20\.19\.(127|128)
cas.monitor.endpoints.status.enabled=true
endpoints.restart.enabled=false
endpoints.shutdown.enabled=false
```

=> Activation des endpoints et autorisation d'accès accordée à des IPs spécifiques + restriction de certaines actions dont shutdown et restart. Ça peut être vérifié en allant voir : *https://cas5.yerbynet.com/cas/status/dashboard*.

6- Branchement à un LDAP pour l'authentification via LDAP

- Le module Maven cas-overlay qui nous permet d'avoir le support LDAP à mettre dans *pom.xml* est :

```
<dependency>
    <groupId>org.apereo.cas</groupId>
    <artifactId>cas-server-support-ldap</artifactId>
    <version>${cas.version}</version>
</dependency>
```

C'est à rajouter dans la section des dépendances (**<dependencies>**).

Ensuite, comme déjà expliqué plusieurs fois dans ce document, il faut recompiler CAS avec la commande ***mvn clean package***.

- La configuration correspondante dans */etc/cas/config/cas.properties* ressemble à ceci :

```
cas.authn.accept.users=
cas.authn.ldap[0].type=AUTHENTICATED
cas.authn.ldap[0].ldapUrl=ldaps://ldaps.yerbynet.com
cas.authn.ldap[0].useSsl=true
cas.authn.ldap[0].useStartTls=false
cas.authn.ldap[0].connectTimeout=5000
cas.authn.ldap[0].subtreeSearch=true
cas.authn.ldap[0].searchFilter=sAMAccountName={user}
cas.authn.ldap[0].baseDn=ou=usagers,dc=yerbynet,dc=com
cas.authn.ldap[0].bindDn=cn=cas5,ou=usagers,dc=yerbynet,dc=com
cas.authn.ldap[0].bindCredential=xxxxxxxxx
cas.authn.ldap[0].principalAttributeList=sAMAccountName,displayName,sn,cn,givenName,m
emberOf,mail,mail:email,memberOf:group
cas.authn.ldap[0].poolPassivator=BIND
```

On en profite avec ***cas.authn.ldap[0].principalAttributeList*** le paramètre pour retirer une liste d'attribut de notre LDAP dont auront besoin des applications pour accorder les autorisations.

On peut être branché à plusieurs LDAP, et dans ce cas on aura les mêmes paramètres, mais indexés de 1, ou 2, … (exemple : ***cas.authn.ldap[1].ldapUrl=ldaps://ldaps1.yerbynet.com***).

Notez aussi qu'on utilise LDAPS (port 636) au lieu de LDAP (389). Il faudra ouvrir les firewalls en conséquence. À moins que les 2 serveurs (CAS et LDAP) ne soient sur un même segment de réseau bien sécurisé, ou même sur la même machine, on déconseillera d'utiliser 389. Par contre, il faut que LDAPS ait un certificat valide reconnu par le JAVA utilisé par CAS. On peut tester avec cette commande :

```
wget https://confluence.atlassian.com/kb/files/779355358/779355357/1/1441897666313/SSLPoke.class
java SSLPoke ldaps.yerbynet.com 636
 Successfully connected
```

En cas d'absence de succès, c'est que votre certificat de LDAPS n'est pas reconnu et accepté par Java. Il faut lui mettre un certificat valide.
Un workaround est de faire manger à Java le certificat en question.
Extraire le certificat ici :

```
openssl s_client -connect ldaps.yerbynet.com:636 -showcerts
```

Recopier le certificat dans */tmp/certificat.crt.*
Puis dire à Java de lui faire confiance :

```
keytool -import -v -trustcacerts -alias  ldaps.yerbynet.com -file /tmp/certificat.crt -keystore /etc/pki/java/cacerts -keypass changeit -storepass changeit
```

7- Enregistrement des tickets en BD

Par défaut, les tickets seront juste conservés en mémoire vive. Ce n'est pas forcément mauvais mais on sait ce qui arrivera lorsqu'on va rebooter l'OS suite à mise à jour. Donc, on devrait être capable de garder même son SSO après reboot de l'OS.

On va utiliser la base de données **casdb** qui a été créée lors des requis.

- Le support de cette fonctionnalité est activée en ajoutant les modules suivants dans notre modèle cas-overlay est (dans *pom.xml*) :

```
<dependency>
  <groupId>org.apereo.cas</groupId>
  <artifactId>cas-server-support-jpa-ticket-registry</artifactId>
  <version>${cas.version}</version>
</dependency>
<dependency>
  <groupId>org.apereo.cas</groupId>
  <artifactId>cas-server-support-jdbc-drivers</artifactId>
  <version>${cas.version}</version>
</dependency>
```

- Voici ce qu'il faut inclure dans *cas.properties* :

```
cas.ticket.registry.jpa.ticketLockType=NONE
cas.ticket.registry.jpa.jpaLockingTimeout=3600
cas.ticket.registry.jpa.healthQuery=
cas.ticket.registry.jpa.isolateInternalQueries=false
cas.ticket.registry.jpa.url=jdbc:mysql://localhost/casdb
cas.ticket.registry.jpa.dialect=org.hibernate.dialect.MariaDBDialect
cas.ticket.registry.jpa.leakThreshold=10
cas.ticket.registry.jpa.batchSize=1
cas.ticket.registry.jpa.defaultCatalog=
cas.ticket.registry.jpa.defaultSchema=cas5db
cas.ticket.registry.jpa.user=casuser
cas.ticket.registry.jpa.ddlAuto=create-drop
cas.ticket.registry.jpa.password=xxxxxxxxxxxx
cas.ticket.registry.jpa.autocommit=false
cas.ticket.registry.jpa.driverClass=org.mariadb.jdbc.Driver
cas.ticket.registry.jpa.idleTimeout=5000
cas.ticket.registry.jpa.dataSourceName=
cas.ticket.registry.jpa.dataSourceProxy=false
```

```
cas.ticket.registry.jpa.properties.propertyName=propertyValue
```

Ici, on fait un tour de passe passe avec CAS pour avoir un schéma de BD utilisable. La valeur de **cas.ticket.registry.jpa.ddlAuto** est d'abord mise à **create-drop**, suivi d'un arrêt/démarrage de CAS. La BD va ainsi être créée. Par la suite, on récupère la BD créée avec un mysqldump. On arrête CAS, on restaure la BD, puis avant de redémarrer CAS, on change le paramètre **cas.ticket.registry.jpa.ddlAuto** en **validate**.

```
cas.ticket.registry.jpa.ticketLockType=NONE
cas.ticket.registry.jpa.jpaLockingTimeout=3600
cas.ticket.registry.jpa.healthQuery=
cas.ticket.registry.jpa.isolateInternalQueries=false
cas.ticket.registry.jpa.url=jdbc:mysql://localhost/casdb
cas.ticket.registry.jpa.dialect=org.hibernate.dialect.MariaDBDialect
cas.ticket.registry.jpa.leakThreshold=10
cas.ticket.registry.jpa.batchSize=1
cas.ticket.registry.jpa.defaultCatalog=
cas.ticket.registry.jpa.defaultSchema=cas5db
cas.ticket.registry.jpa.user=casuser
cas.ticket.registry.jpa.ddlAuto=validate
cas.ticket.registry.jpa.password=xxxxxxxxxxxx
cas.ticket.registry.jpa.autocommit=false
cas.ticket.registry.jpa.driverClass=org.mariadb.jdbc.Driver
cas.ticket.registry.jpa.idleTimeout=5000
cas.ticket.registry.jpa.dataSourceName=
cas.ticket.registry.jpa.dataSourceProxy=false
cas.ticket.registry.jpa.properties.propertyName=propertyValue
```

Enfin, CAS peut être relancé sans crainte.

8- Enregistrement des services en BD

On va profiter de notre BD pour enregistrer nos services (applications) qui auront le droit d'utiliser CAS.

- Ajout du module dans le fichier *pom.xml* :

```
<dependency>
   <groupId>org.apereo.cas</groupId>
   <artifactId>cas-server-support-jpa-service-registry</artifactId>
   <version>${cas.version}</version>
</dependency>
```

- Configuration correspondante dans *cas.properties* :

```
cas.serviceRegistry.jpa.user=casuser
cas.serviceRegistry.jpa.password=xxxxxxxxxx
cas.serviceRegistry.jpa.driverClass=org.mariadb.jdbc.Driver
cas.serviceRegistry.jpa.url=jdbc:hsqldb:mem:cas-hsql-database
cas.serviceRegistry.jpa.dialect=org.hibernate.dialect.HSQLDialect
cas.serviceRegistry.jpa.url=jdbc:mysql://localhost/casdb
cas.serviceRegistry.jpa.dialect=org.hibernate.dialect.MariaDBDialect
cas.serviceRegistry.jpa.failFastTimeout=1
cas.serviceRegistry.jpa.healthQuery=
cas.serviceRegistry.jpa.isolateInternalQueries=false
cas.serviceRegistry.jpa.leakThreshold=10
cas.serviceRegistry.jpa.batchSize=1
cas.serviceRegistry.jpa.defaultCatalog=
cas.serviceRegistry.jpa.defaultSchema=
#cas.serviceRegistry.jpa.ddlAuto=create-drop
cas.serviceRegistry.jpa.ddlAuto=validate
cas.serviceRegistry.jpa.autocommit=false
cas.serviceRegistry.jpa.idleTimeout=5000
cas.serviceRegistry.jpa.dataSourceName=
cas.serviceRegistry.jpa.dataSourceProxy=false
cas.serviceRegistry.jpa.properties.propertyName=propertyValue
cas.serviceRegistry.jpa.pool.suspension=false
cas.serviceRegistry.jpa.pool.minSize=6
```

```
cas.serviceRegistry.jpa.pool.maxSize=18
cas.serviceRegistry.jpa.pool.maxWait=2000
cas.serviceRegistry.jpa.pool.timeoutMillis=1000
```

Ne pas oublier le ping pong du create/create-drop/validate pour générer le schéma de la BD.

À partir de ce point, on peut faire des tests un peu plus poussés de notre CAS. Mais pour se faire, il faut faire jump pour se retrouver à l'étape « CAS Management ». Il nous faut un CAS management afin d'ajouter des services qui vont rediriger leur authentification sur CAS.
Donc, pour le lecteur impatient de faire des tests plus intéressants, vas d'abord faire l'étape « CAS Management ».
Étant donné que je n'ai moi même pas encore écrit cette partie de « CAS Management » au moment où j'écris ces lignes, je ne sais pas si j'aurai le courage d'arriver au « CAS Management » un jour. Tu peux aussi ajouter le service à la main (oui manuellement), mais il aurait fallu ne pas enregistrer les services en BD, donc jump au début de cette section, et défait tout ce que tu viens de faire, puis vas à l'étape de « Test avec mod-cas de Apache ».
Voilà! Là je commence à écrire une documentation comme les gens de CAS eux même. LOL !
Sérieusement, vous pouvez ne pas ajouter toutes les fonctionnalités décrites dans ce document, d'où les propositions de JUMP ; et vous pouvez en effet vous rendre à la section « CAS Management », que j'écrirai forcément avant de publier ce document (ou pas).

9- SAML IDP

C'est avec cette fonctionnalité que CAS arrive à faire du Shibboleth comme on dit chez nous, c'est à dire du SAML.

- Modules dans *pom.xml* :

```
<dependency>
```

```
    <groupId>org.apereo.cas</groupId>
    <artifactId>cas-server-support-saml</artifactId>
    <version>${cas.version}</version>
  </dependency>
  <dependency>
    <groupId>org.apereo.cas</groupId>
    <artifactId>cas-server-support-saml-idp</artifactId>
    <version>${cas.version}</version>
  </dependency>
```

- *cas.properties* :

```
# Metadata
cas.authn.samlIdp.entityId=https://cas5.yerbynet.com/cas/idp
cas.authn.samlIdp.scope=cas5.yerbynet.com
cas.authn.samlIdp.attributeQueryProfileEnabled=false
cas.authn.samlIdp.metadata.cacheExpirationMinutes=30
cas.authn.samlIdp.metadata.failFast=true
cas.authn.samlIdp.metadata.privateKeyAlgName=RSA
cas.authn.samlIdp.metadata.requireValidMetadata=true
cas.authn.samlIdp.metadata.location=file:///etc/cas/saml
# Logout
cas.authn.samlIdp.logout.forceSignedLogoutRequests=true
cas.authn.samlIdp.logout.singleLogoutCallbacksDisabled=false
# Response
cas.authn.samlIdp.response.defaultAuthenticationContextClass=
cas.authn.samlIdp.response.defaultAttributeNameFormat=uri
cas.authn.samlIdp.response.signError=false
cas.authn.samlIdp.response.skewAllowance=5
cas.authn.samlIdp.response.attributeNameFormats=attributeName->basic|uri|unspecified|
custom-format-etc,...
# Ticket
cas.authn.samlIdp.ticket.samlArtifactsCacheStorageName=samlArtifactsCache
cas.authn.samlIdp.ticket.samlAttributeQueryCacheStorageName=samlAttributeQueryCache
```

```
mkdir /etc/cas/saml/
```

Une compilation suivie d'une relance de CAS devrait créer les fichiers de metadata dans */etc/cas/saml/*.
Pour valider que ça marche, rendez-vous sur le endpoint correspondant : *https://cas5.yerbynet.com/cas/idp/metadata*. On devrait retrouver notre fichier de metadata comme il est dans */etc/cas/saml/*. C'est cette URL qu'il faudra fournir au fournisseurs de services avec qui on souhaite faire du SAML.

Juste pour rigoler, faisons aussi du SAML SP. Encore une fois, vous pouvez sauter la section suivante; c'est d'ailleurs la section la plus farfelue.

10 - SAML SP : Délégation de l'authentification à Shibboleth

On va donc brancher notre CAS à un Shibboleth. CAS va réorienter l'authentification vers Shibboleth.

- *pom.xml*

```
<dependency>
   <groupId>org.apereo.cas</groupId>
   <artifactId>cas-server-support-pac4j-webflow</artifactId>
   <version>${cas.version}</version>
</dependency>
```

- *cas.properties*

```
cas.authn.pac4j.saml[0].keystorePassword=xxxxxxxxxx
cas.authn.pac4j.saml[0].privateKeyPassword=xxxxxxxxxx
cas.authn.pac4j.saml[0].keystorePath=/etc/cas/ssl/samlKeystore.jks
```

```
cas.authn.pac4j.saml[0].keystoreAlias=cas5.yerbynet.com
cas.authn.pac4j.saml[0].serviceProviderEntityId=urn:mace:saml:pac4j.org
cas.authn.pac4j.saml[0].serviceProviderMetadataPath=/etc/cas/saml/sp-metadata.xml
cas.authn.pac4j.saml[0].identityProviderMetadataPath=https://shib.yerbynet.com/idp/shibboleth
cas.authn.pac4j.saml[0].maximumAuthenticationLifetime=3600
cas.authn.pac4j.saml[0].destinationBinding=urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect
cas.authn.pac4j.saml[0].wantsAssertionsSigned=true
cas.authn.pac4j.saml[0].assertionConsumerServiceIndex=-1
cas.authn.pac4j.saml[0].principalAttributeId=cn
cas.authn.pac4j.saml[0].clientName=Shibboleth
```

Notez que j'ai légèrement modifié la BD pour que ça fonctionne correctement :

```
alter table SAML2_ATTRIBUTE_QUERY_TICKETS modify column object varchar(10000);
```

J'ai doublé la taille de *SAML2_ATTRIBUTE_QUERY_TICKETS.*

Ensuite, il faudra faire les modifications qui s'imposent dans le Shibboleth avec qui on se connecte (attribute-filter, metadata-providers, relying-party).

Voir : http://www.yerbynet.com/Cours/ShibbolethInstallation.pdf

11- OAUTH2 provider

- *pom.xml* :

```
<dependency>
  <groupId>org.apereo.cas</groupId>
  <artifactId>cas-server-support-oauth-webflow</artifactId>
  <version>${cas.version}</version>
</dependency>
```

- *cas.properties* :

```
cas.authn.oauth.refreshToken.timeToKillInSeconds=2592000
cas.authn.oauth.code.timeToKillInSeconds=30
cas.authn.oauth.code.numberOfUses=1
cas.authn.oauth.accessToken.releaseProtocolAttributes=true
cas.authn.oauth.accessToken.timeToKillInSeconds=7200
cas.authn.oauth.accessToken.maxTimeToLiveInSeconds=28800
cas.authn.oauth.grants.resourceOwner.requireServiceHeader=true
cas.authn.oauth.userProfileViewType=FLAT
#cas.authn.oauth.userProfileViewType=NESTED|FLAT
```

- Modifications BD : il faut créer la table **OAUTH_TOKENS**

Il faut créer la table pour l'enregistrement de nos tickets OAuth2 dans notre BD **casdb**.

Voici le SQL de la création de la table :

```
CREATE TABLE `OAUTH_TOKENS` (
 `TYPE` varchar(31) NOT NULL,
 `ID` varchar(255) NOT NULL,
 `NUMBER_OF_TIMES_USED` int(11) DEFAULT NULL,
 `CREATION_TIME` datetime DEFAULT NULL,
 `EXPIRATION_POLICY` longblob NOT NULL,
 `EXPIRED` bit(1) NOT NULL,
 `LAST_TIME_USED` datetime DEFAULT NULL,
 `PREVIOUS_LAST_TIME_USED` datetime DEFAULT NULL,
 `AUTHENTICATION` longblob NOT NULL,
 `scopes` longblob,
 `SERVICE` longblob NOT NULL,
 `ticketGrantingTicket_ID` varchar(255) DEFAULT NULL,
 PRIMARY KEY (`ID`),
 KEY `FKq0iyiw6b2rufxe2ocmpsqdo50` (`ticketGrantingTicket_ID`),
 CONSTRAINT `FKq0iyiw6b2rufxe2ocmpsqdo50` FOREIGN KEY
(`ticketGrantingTicket_ID`) REFERENCES `TICKETGRANTINGTICKET` (`ID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Les endpoints correspondants fournis par cette fonctionnalité sont :

```
https://cas5.yerbynet.com/cas/oauth2.0/authorize
https://cas5.yerbynet.com/cas/oauth2.0/accessToken
https://cas5.yerbynet.com/cas/oauth2.0/token
https://cas5.yerbynet.com/cas/oauth2.0/profile
```

- Documentation supplémentaire :

https://apereo.github.io/cas/5.3.x/installation/OAuth-OpenId-Authentication.html

https://tools.ietf.org/html/rfc6749#section-4.1

https://alexbilbie.com/guide-to-oauth-2-grants/

En ce qui concerne ce document, je n'ajouterai plus de module/fonctionnalité CAS.
La suite sera consacrée à un peu de cosmétique que je trouve tout de même nécessaire d'en parler. On va par exemple mettre en place le script de démarrage plutôt que de le démarrer à la main. On va rouler CAS avec un utilisateur différent de root. On va regarder comment on peut fournir des attributs statiques, et comment tester tout ceci avec Apache. Mais avant, regardons l'installation de « CAS-Management ».

12- CAS Management

CAS, pour ceux qui ne le savent pas, mais qui lisent ce document depuis le début, permet à des services ou applications d'authentifier leurs utilisateurs via lui. Il y a donc une config qui est faite au niveau de l'application pour déléguer l'authentification à CAS, mais il faut aussi que CAS « connaisse » l'application, ainsi que les éléments (attributs) dont l'application a besoin pour accorder des autorisations à ses usagers. CAS pourrait par exemple pour une application donnée lui renvoyer la date de naissance d'un usager qui s'authentifie si des autorisations sont accordées en fonction de l'âge dans l'application, ou bien renvoyer les groupes de l'utilisateur

pour une autre application qui se base sur des groupes. CAS pourrait lui même accorder les autorisations en fonction de critères qu'on lui fournit.

Bref, à moins de connaître tous les paramètres, et d'avoir des cas de figures très simples à configurer, il nous faut un instrument pour faire connaître à CAS les applications qu'il autorise à l'utiliser, ainsi que les attributs qu'il leur renvoie respectivement.

On aurait pu imaginer que ce module, crucial, aurait été inclus dans le core de CAS (ou même sous forme de module), mais non, c'est un service supplémentaire à télécharger, compiler, installer, et exécuter, tout comme CAS.

Heureusement, il se compile comme CAS (*pom.xml* + le fichier *management.properties)*. Donc, on bénéficiera de toute l'expérience acquise dans la compilation de CAS pour nos opérations sur CAS Management.

12-1/ Installation de CAS-Management

Il est à récupérer ici : https://github.com/apereo/cas-management/tree/5.3.x

Il faut décompresser l'archive, ou le récupérer avec git, puis accéder au répertoire ainsi créé.

On va aussi le mettre dans /*opt*/. On aura donc */opt/cas-management-5.3.x/* qui contiendra notre *pom.xml* à modifier.

La commande pour compiler reste la même : ***mvn clean package***

CAS-Management a juste besoin d'accéder en lecture/écriture à notre système d'enregistrement de services tel que spécifié dans CAS. Dans notre cas, nous avons décidé d'enregistrer nos services dans une BD MySQL, donc CAS-Mangement doit avoir accès à cette BD en lecture/écriture. On pourrait lui donner uniquement accès à la table des services.

Il faut donc qu'on rajoute le module **jpa-service-registry.**

```
<dependency>
  <groupId>org.apereo.cas</groupId>
  <artifactId>cas-server-support-jpa-service-registry</artifactId>
  <version>${cas-mgmt.version}</version>
```

```
    </dependency>
    <dependency>
      <groupId>org.apereo.cas</groupId>
      <artifactId>cas-server-support-jdbc-drivers</artifactId>
      <version>${cas-mgmt.version}</version>
    </dependency>
```

12-2/ Configuration CAS-Management

Le fichier de config générale s'appelle */etc/cas/config/management.properties*, et par défaut, le fichier */etc/cas/config/log4j2-management.xml* permettra de configurer les logs.

- Généralités dans */etc/cas/config/management.properties*

```
server.context-path=/cas-management
server.port=8443
logging.config=file:/etc/cas/config/log4j2-management.xml
server.ssl.enabled=true
server.ssl.keyStore=file:///etc/cas/ssl/cas5.yerbynet.com.jks
server.ssl.keyStorePassword=xxxxxxxxxx
server.ssl.keyPassword=xxxxxxxxxx
mgmt.serverName=https://cas5.yerbynet.com
```

On roule l'application sur le port 8443, avec SSL activé.

Ce qui nous donne l'URL : https://cas5.yerbynet.com:8443/cas-management

Il nous faudra donc ouvrir le port 8443 dans notre firewall. On pourrait juste l'ouvrir pour les IPs qui ont le droit de modifier les services de CAS.

```
firewall-cmd --permanent --zone=public --add-rich-rule='rule family="ipv4" source address="AddressIP1" port protocol="tcp" port=8443 accept'
firewall-cmd --permanent --zone=public --add-rich-rule='rule family="ipv4" source address="AddressIP2" port protocol="tcp" port=8443 accept'
firewall-cmd --reload
```

- Connexion à la BD

On utilise exactement les mêmes paramètres de connexion que ceux de CAS.

```
cas.serviceRegistry.jpa.user=casuser
cas.serviceRegistry.jpa.password=xxxxxxxxxx
cas.serviceRegistry.jpa.driverClass=org.mariadb.jdbc.Driver
cas.serviceRegistry.jpa.url=jdbc:mysql://localhost/casdb
cas.serviceRegistry.jpa.dialect=org.hibernate.dialect.MariaDBDialect
cas.serviceRegistry.jpa.failFastTimeout=1
cas.serviceRegistry.jpa.healthQuery=
cas.serviceRegistry.jpa.isolateInternalQueries=false
cas.serviceRegistry.jpa.leakThreshold=10
cas.serviceRegistry.jpa.batchSize=1
cas.serviceRegistry.jpa.defaultCatalog=
cas.serviceRegistry.jpa.defaultSchema=
cas.serviceRegistry.jpa.ddlAuto=validate
cas.serviceRegistry.jpa.autocommit=false
cas.serviceRegistry.jpa.idleTimeout=5000
cas.serviceRegistry.jpa.properties.propertyName=propertyValue
cas.serviceRegistry.jpa.pool.suspension=false
cas.serviceRegistry.jpa.pool.minSize=6
cas.serviceRegistry.jpa.pool.maxSize=18
cas.serviceRegistry.jpa.pool.maxWait=2000
cas.serviceRegistry.jpa.pool.timeoutMillis=1000
```

- Gestion de l’accès à l’interface de CAS-Management

Écoutez bien ceci. Par défaut, CAS vient avec un usager pour s’y connecter qui est casuser/Mellon. CAS-Management est quant à lui censé gérer les services qui utilisent l’authentification CAS, mais son comportement par défaut est de se connecter à un CAS pour authentifier les utilisateurs qui doivent l’utiliser. Quel bourrage d’urnes! Les élections sont vraiment truquées ici. Donc, le service qui est censé gérer les services, est, lui même, censé être un service de CAS. Mais comment accéder à CAS-Management alors, puisque son service n’est pas encore créé dans CAS? Il y a bien un service générique qui existe par défaut dans

CAS pour CAS-Management, mais ce n'est pas en phase d'installation qu'il faut l'utiliser. Bref, on désactive ça rapidement en ajoutant lignes ci-dessous à *management.properties* :

```
cas.server.name:
cas.server.prefix:
```

On initialise à vide le nom du serveur CAS (cas.server.name).

Ensuite, on ajoute une autorisation par IP, ou bien avec un utilisateur par défaut.

```
mgmt.authzIpRegex=IP1|IP2|IP3
```

Une fois toutes nos configurations terminées et notre CAS en pleine confiance, on pourra décider d'authentifier CAS-Management à travers CAS en créant un service spécifique pour CAS-Management dans CAS avec des restrictions bien comme il faut. Voilà !

- Se donner la possibilité de manipuler les attributs (via des filtres ou même pour la sélection des attributs qu'on envoie)

Pour avoir la possibilité de sélectionner des attributs dans l'interface de CAS-Management, il faut qu'on les déclare dans la config de CAS-Management.

```
cas.authn.attributeRepository.stub.attributes.givenName=givenName
cas.authn.attributeRepository.stub.attributes.mail=mail
cas.authn.attributeRepository.stub.attributes.cn=cn
cas.authn.attributeRepository.stub.attributes.sn=sn
cas.authn.attributeRepository.stub.attributes.monAttribut=tonAttribut
```

Cet extrait de *management.properties* va permettre de sélectionner les attributs *mail, cn, sn, givenName, tonAttribut* dans l'interface de CAS-Management.

13- SystemD et démarrage automatique

On aura besoin de deux scripts **cas-management.service** et **cas.service**.

vim /usr/lib/systemd/system/cas.service

```
# Systemd unit file for cas5
[Unit]
Description=Apereo CAS 5
After=syslog.target network.target

[Service]
Type=simple
NonBlocking=true

Environment=JAVA_HOME=/usr/lib/jvm/jre
Environment='JAVA_OPTS=-Djava.awt.headless=true -Xms3.9G -Xmx4G -
Djava.net.preferIPv4Stack=true -Djava.security.egd=file:/dev/urandom -XX:
+UseParallelGC -server'
ExecStart=/opt/cas5/cas.war

User=cas
Group=cas

[Install]
WantedBy=multi-user.target
```

Pour que ce script marche, il faut créer l'utilisateur et le groupe cas., et il faut recopier */opt/cas-overlay-template-5.3/target/cas.war* dans */opt/cas5/cas.war.*

vim /usr/lib/systemd/system/cas-management.service

```
# Systemd unit file for cas management
[Unit]
Description=Apereo CAS 5 Management
After=syslog.target network.target

[Service]
Type=simple
```

```
Environment=JAVA_HOME=/usr/lib/jvm/jre
Environment='JAVA_OPTS=-Djava.awt.headless=true -Xms1600m -Xmx1700m -
Djava.net.preferIPv4Stack=true -Djava.security.egd=file:/dev/urandom -XX:
+UseParallelGC -server'
ExecStart=/opt/cas5/cas-management.war

User=cas
Group=cas

[Install]
WantedBy=multi-user.target
```

Il faut aussi copier */opt/cas-management-5.3.x/target/cas-management.war* dans */opt/cas5/*.

L'utilisateur système **cas** doit être capable d'exécuter les .war, et être capable d'écrire dans /*etc/cas/*

Puis on active tout ceci :

```
systemctl daemon-reload
systemctl enable cas
systemctl enable cas-management
systemctl restart cas
systemctl restart cas-management
```

On n'aura donc plus besoin de lancer des ***java -jar*** en ligne de commande.

14- Branchement d'un apache avec mod-cas

Quelques URLs de documentations :

https://github.com/apereo/mod_auth_cas/blob/master/README

https://stackoverflow.com/questions/8939487/how-to-support-require-group-foobar-in-mod-auth-cas

Il faudrait avoir un Apache installé quelque part sur une machine. Le module pour que Apache puisse causer avec CAS s'appelle **mod_auth_cas**. Il faut installer ce module.

Par la suite, on configure un petit vhost qui contiendra par exemple :

```
    CASLoginURL https://cas5.yerbynet.com/cas/login
    CASValidateURL https://cas5.yerbynet.com/cas/serviceValidate
    CASValidateURL https://cas5.yerbynet.com/cas/samlValidate
    CASCookiePath /var/lib/cas/
#    CASCertificatePath /etc/pki/tls/certs/testcas5.yerbynet.com.crt # dépend du certificat
    CASValidateSAML On
#    CASDebug On

      <Location /secured>
            Authtype CAS
#           Require cas-attribute id:roger # si je veux juste roger
#           Require valid-user # n'importe quel usager authentifié
            Require cas-attribute memberOf:CN=Personne,CN=Users,DC=yerbynet,DC=net
      </Location>

      <Location /secured2>
            Authtype CAS
            Require cas-attribute cn:roger , cn:anna , cn:matou, cn:latifa
            Require cas-attribute email:ry@yerbynet.com
      </Location>
```

On a deux ressources sur ce serveur Apache à protéger **/secured** et **/secured2**. Si on suppose que le vhost en question est https://testcas.yerbynet.com, on devra créer un (ou 2) service dans CAS (grâce à CAS-Management) qui porte l'URL « ^https://testcas.yerbynet.com/secured.* » et qui soit capable de retourner les attributs utilisés dans Mod-CAS (email, cn et memberOf).

Ce sujet nous mène tout droit au point 15 qui porte sur les attributs.

15- Les attributs à envoyer

Documentation :

https://apereo.github.io/cas/5.3.x/integration/Attribute-Release-Policies.html

https://apereo.github.io/cas/5.3.x/installation/Configuration-Properties.html#principal-resolution

Avec une ligne de config de ce genre, CAS va chercher les attributs dans un LDAP, et est prêt à les fournir aux applications qui en demandent.

```
cas.authn.ldap[0].principalAttributeList=sAMAccountName,displayName,sn,cn,givenName,
memberOf,mail,mail:email,memberOf:group
```

Notez qu'on renomme **mail** en **email**, et on fournit à la fois **email** et **mail**. Certaines applications chercheront l'adresse de courriel dans l'attribut **mail**, tandis que d'autres s'attendront à recevoir **email**. Il faut que CAS soit prêt à se plier aux désirs de l'application. On pourrait aussi renommer les attributs au niveau de la définition du service en utilisant les attributs mappés (**Return Mapped**).

On peut aussi utiliser un deuxième annuaire :

```
cas.authn.attributeRepository.ldap[1].attributes.uid=sAMAccountName
cas.authn.attributeRepository.ldap[1].attributes.displayName=displayName
```

On peut avoir des attributs statiques :

```
cas.authn.attributeRepository.stub.attributes.MonAttributStatic=<valeur_que_je_veux>
```

Comme il y a des attributs statiques, il y a aussi des attributs scriptés.

Exemple très basique de script **Groovy** :

vim /etc/cas/scripts/test-attributes.groovy

```
import java.util.*
import java.util.Map
import java.util.HashMap
import org.apereo.cas.support.saml.services.*
import org.apereo.cas.support.saml.*

def Map<String, List<Object>> run(final Object... args) {
    def attributes = args[0]
    def logger = args[1]
    def casProperties = args[2]
    def casApplicationContext = args[3]

    String myuserid;
    Map usersid = new HashMap();

    usersid.put("matou", 0);
    usersid.put("anna", 1);
    usersid.put("jennifer", 2);

    logger.info("Current attributes received are {}", attributes)
    logger.info("Le CN est {}", attributes['cn'].get(0))
    logger.info("Le mail est {}", attributes['mail'].get(0))
```

```
    myuserid = attributes['cn'].get(0)


    if (usersid.containsKey(myuserid))
        return [cn:attributes['cn'].get(0), displayName:attributes['displayName'].get(0),
mail:attributes['mail'].get(0)]
    //else return null;
    else {
        logger.warn("NNNNNNUUUUUUUUUUUULLLLLLLLLLLLLLLLL");
        logger.warn("Le return  cn:'null', displayName:'null', mail:'null@yerbynet.com'");
        return [sAMAccountName:'null', username:'null', cn:'null', displayName:'null',
mail:'null@yerbynet.com']
    }
}
```

Oui, c'est du Java.

On appelera ensuite ce script dans la définition d'un service, en spécifiant le chemin d'accès au script.

16- Documentation des endpoints de notre CAS

https://cas5.yerbynet.com/cas : C'est par là que tout démarre

https://cas5.yerbynet.com/cas/status/dashboard : Le tableau de bord de CAS

https://cas5.yerbynet.com/cas/status/services : Un aperçu sur les services autorisés au authentifier via CAS (impossible de modifier ici)

https://cas5.yerbynet.com/cas/login : La page de login

https://cas5.yerbynet.com/cas/login#divAttributes : Récupérer tous les attributs du compte qui est connecté (envoyés ou non à l'appli)
https://cas5.yerbynet.com/cas/logout : Logout (à appeler par les appli qui veulent se délogguer)
https://cas5.yerbynet.com/cas/v2/api-docs : La doc de tous les endpoints qui sont offerts par notre CAS (description, méthode d'appel, paramètres consommés, format, informations générées, code d'erreur, ...)
https://cas5.yerbynet.com/cas/idp : Le entityID de SAML (Shibboleth) de CAS
https://cas5.yerbynet.com/cas/idp/metadata : La metadata du IDP de CAS
https://cas5.yerbynet.com:8443/cas-management/manage.html : Gestion des services
Delegate to Shibboleth :
https://cas5.yerbynet.com/cas/sp/metadata
https://cas5.yerbynet.com/cas/sp/idp/metadata

Vous l'aurez compris, on est plus proche de la fin que du début, et pour ceux qui n'aiment jamais la fin des bouquins, c'est le moment où jamais d'arrêter de continuer de lire. Il est tout à fait possible que j'arrête aussi d'écrire à partir d'ici et que « vous soyez quand même faits ».
Allez, attardons nous un bon coup avant de passer à l'interface Web. C'est peut être le moment de sortir fumer pour les non-fumeurs. Les fumeurs quant à eux ont dû fumer quelques paquets tout au long de la lecture et des expérimentations.

17- Customisations de l'interface web (la page de login par exemple)

Encore une fois, je me suis fait avoir ici et j'ai perdu du temps avant de comprendre ce qu'il fallait faire. Non, il n y a pas de paramètres de configuration pour influencer l'interface graphique. Oui, tu as tout compris, il faut repasser par la compilation de CAS.
Pour se faire, il faut créer le répertoire **src/main/resources/** dans **/opt/cas-overlay-template-5.3/**.

- Pour chaque élément qu'on veut customiser :

- il doit être recopié dans ce dossier **/opt/cas-overlay-template-5.3/src/main/resources/** depuis **/opt/cas-overlay-template-5.3/target/classes/** en respectant l'arborescence.
- Puis l'élément est modifié.
- On recompile CAS : ***/usr/local/bin/maven/bin/mvn clean package*** (on doit être positionné dans le répertoire **/opt/cas-overlay-template-5.3/**)
- On arrête le CAS qui roule : ***systemctl stop cas***
- Puis on le *republie* dans **/opt/cas/5** : ***cp /opt/cas-overlay-template-5.3/target/cas.war /opt/cas5/cas.war && chown cas:cas /opt/cas5/cas.war***
- On relance enfin CAS qui est censé contenir la modification : ***systemctl restart cas***

Pour aller plus loin dans la customisation, il faut lire : https://apereo.github.io/cas/5.3.x/installation/User-Interface-Customization.html et confier le boulot à des graphistes et des développeurs Web.

18- Throttle des tentatives d'authentification

https://groups.google.com/a/apereo.org/forum/#!topic/cas-user/EkS2Jg06Vgo
C'est une fonctionnalité buguée. Pas la peine de perdre du temps avec. Je l'ai signalé à UNICON qui m'a proposé de corriger le bug moyennant un financement. On va donc patienter que le fix sorte sans avoir à débourser quoi que ce soit.

Conclusion

J'ai personnellement installé et réinstallé, puis re-re-installé CAS depuis quelques mois. J'ai utilisé deux (ou trois) douzaines d'heures pour rédiger ce document. Comme d'habitude, je ne

vais pas le relire intégralement. Généralement, je me relis au fur et à mesure que j'écris, parce que je m'ennuies lorsque je me relis entièrement. J'ai dû me tordre un peu la mémoire, parce qu'il y a certains éléments de configuration et certains comportements de la compilation que j'avais oubliés. J'espère et je pense que ce document ne comporte pas beaucoup de bugs de documentation. J'espère donc vous avoir donné de bonnes bases pour une installation plus poussée de CAS. Vous pourrez produire des installations plus sophistiquées avec des BD en redondance, plusieurs CAS en redondance, du HA, du Glusterfs, sur du Opensteck, … À la fin, ça ne marchera peut être pas, ou bien ça va plutôt donner de la HI (Haute Indisponibilité), mais vous l'aurez fait.

Le plus important à retenir, c'est que tout est dans les **dépendances/modules** qu'on rajoute (pom.xml ou gradle.properties), et pour chaque module, il faut ajouter sa configuration dans cas.properties. Il ne serait pas très futé de rajouter tous vos modules en une seule étape (à moins de les avoir déjà rodés lors d'une installation précédente), sinon ça risque de faire mal. Le mieux, c'est de rajouter tranquillement ses modules, un à la fois, en prenant le temps de boire entre chaque ajout de module, quelque soit ce que vous buvez.

Je vous refile certaines des URLs que j'ai consultées lors de mes différentes difficultés d'installations de CAS, et je vous souhaite une très belle digestion. (C'est pour ça qu'il fallait boire beaucoup).

Pour finir, si vous n'êtes pas d'accord avec CAS, allez voir Keycloak ou Gluu.

Sources :

POC : https://artduweb.com/tutoriels/cas-sso

POC : https://aldian.developpez.com/tutoriels/javaee/authentification-centralisee-sso-cas/

Demo : https://apereo.atlassian.net/wiki/spaces/CASUM/pages/103261417/Demo

Téléchargement et documentations officielles : https://github.com/apereo/cas/releases

Tuto Youtube : https://www.youtube.com/watch?v=uuN2CvJ8I58

Keystore : https://tomcat.apache.org/tomcat-8.0-doc/ssl-howto.html

https://jcenter.bintray.com/org/apereo/cas/cas-server-webapp-tomcat/5.3.2/cas-server-webapp-tomcat-5.3.2.war

https://groups.google.com/a/apereo.org/forum/#!topic/cas-user/xnV2g1FCZqQ

Shibbolethiser cas :

https://apereo.atlassian.net/wiki/spaces/CASUM/pages/103261296/Shibboleth-CAS+Integration

overlay : https://apereo.github.io/2018/06/09/cas53-gettingstarted-overlay/

https://stackoverflow.com/questions/tagged/cas

https://github.com/casinthecloud/cas-overlay-demo/tree/5.3.x +++

https://apereo.github.io/cas/5.3.x/installation/Configuration-Properties.html#embedded-container : Toute la conf (Tomcat, LDAP, JDBC, ...)

https://dacurry-tns.github.io/deploying-apereo-cas/pdf/deploying-apereo-cas.pdf : La seule documentation d'installation "complète" que je viens de trouver (pour cas 5.2).

https://dacurry-tns.github.io/deploying-apereo-cas/introduction_overview.html

https://github.com/apereo/cas/milestones : Le cycle de release.

https://groups.google.com/a/apereo.org/forum/#!topic/cas-user/3yuMmrL0IrU : Proxyisation

https://uwaterloo.ca/information-systems-technology/services/cas/client-authentication-methods : Jouer avec les attributs

https://groups.google.com/a/apereo.org/forum/#!topic/cas-user/dPf36An-qyU : TGC & cas.tgc.crypto.encryption.key

https://github.com/spring-projects/spring-security-oauth/blob/master/spring-security-oauth2/src/test/resources/schema.sql : OAUTH2 BD

https://oauth.net/2/

http://students.sunyocc.edu/index.aspx?id=28478

https://apereo.github.io/cas/5.3.x/integration/Attribute-Release-Policies.html

https://apereo.github.io/cas/5.3.x/installation/Configuration-Properties.html#principal-resolution

https://apereo.github.io/cas/5.3.x/installation/User-Interface-Customization.html

https://groups.google.com/a/apereo.org/forum/#!topic/cas-user/EkS2Jg06Vgo