La Barbe dans le cambouis

Sécurisons une bonne vieille application Java "Entreprise"

`François Le Droff` - un des plombiers derrière `adobe.io`

`Romain Pelisse` - corrige des bogues sur les APIs de nos grand parents pour `Red Hat`

Agenda

- Prérequis
- lab1 - notre application ("build and deploy")
- lab2 - mise en place d'une solution d'identity
- lab3 - mise en place de Swagger
- lab4 - mise en place d'un reverse-proxy filtrant
- lab5 - mise en place du chiffrement SSL
- lab6 - intégration avec Sprint Security
- Lab6b - jouons avec oAuth, identification avec Github
- lab7 - base de données - chiffrement et connexion sécurisée
- lab8 - protéger le système de l'application avec le Security Manager et SELinux
- lab9 - intégrité du fichier du contenu du fichier de journalisation

Prérequis

- Systèmes d'exploitation:
  - Linux: bien sûr, désactiver `SELinux`
  - MacOS: Démerdez vous, payez vos impôts, bande d'hipsters
  - Windows: Merci d'installer un vrai OS
  - ou a minima faites en sorte d'avoir un `bash (unix shell)` à disposition
- Outillage:
  - Installez Java - JDK 8 (nous avons testé le lab avec la version `1.8.0_162-b12` et `openjdk version "1.8.0_161)`
  - Venez avec votre IDE/`vi` préféré, peu nous importe ;)
  - Installez Maven (nous avons testé le lab avec `Apache Maven 3.3.9` et `Apache Maven 3.5.0 (Red Hat 3.5.0-6))`
  - Installez docker (nous avons testé la version `18.04.0-ce-rc2` et `1.13.1, build caba767-unsupported)`
  - Installez docker-compose (nous avons testé la version `1.20.1` et `1.17.1, build 6d101fb)`
  - Installez OpenSSL (nous avons testé avec `OpenSSL 1.0.1p 9 Jul 2015` et `1.1.0g-fips @ Nov 2017)`

Prérequis

Pour plus de comfort durant le lab et ne pas souffir des aléas du réseau du palais des congrès, lancer les commandes suivantes à l'avance:

- Clonez notre repo git
  - `git clone https://github.com/francoisledroff/la_barbe_dans_le_cambouis.git`
  - `cd la_barbe_dans_le_cambouis`
- Un peu de Docker ensuite:
  - `docker-compose -f src/main/docker/keycloak.yml up`
  - `src/main/docker/build-docker-image.sh`
  - `docker build -t mongo src/main/docker/mongo/`
- et enfin construisez notre bonne vieille application Java (et télecharger internet pour satisfaire maven)
  - `mvnw clean install`

Une bonne vielle application Java...

Spring Boot

Spring Boot permet de créer facilement des applications Spring avec peu de configuration.

Fonctionalités:

- Crée des applications Spring autonomes
- Intégre directement Tomcat, Jetty ou Undertow (pas besoin de déployer des fichiers WAR)
- Fournit des POM «starter» orientées pour simplifier votre configuration Maven
- Aucune génération de code et aucune exigence pour la configuration XML

https://projects.spring.io/spring-boot/

Lab1 - Construire et démarrer le micro -service

15' ⏱️

```
$ git check-out lab1
$ 'cd' into 'danslecambouis'
$ mvnw compile (erreur)
```

- implémenter le corps de la méthode 'ping' puis exécuter et tester le service

```
$ mvn spring-boot:run
$ curl http://localhost:9080/ping
```

Lab1 - Construire et démarrer le micro -service

Pour aller plus loin...

- Modifier '/ping' pour accepter un paramètre (hostname)
- et exécuter un 'ping' (OS) sur le nom de domaine fourni en argument
- et retourner le temps de réponse du premier ping ("[hostname]: 20ms").

La sécurité commence par ...

KEYCLOAK

Keycloak

Solution Open Source (RedHat) de gestion de l'identité et d'accès: plus besoin de gérer le stockage des utilisateurs ou l'authentification des utilisateurs.

- SSO: Connectez-vous une fois pour plusieurs applications
- Protocoles standards: OpenID Connect, OAuth 2.0 et SAML 2.0
- LDAP et Active Directory: connectez-vous aux répertoires d'utilisateurs existants
- Connexion sociale
- Clustering
- Thèmes: personnalisez l'aspect et la convivialité
- Extensible: codez !
- Mot de passe:personnalisez les règles de mot de passe

https://www.keycloak.org/

Lab2 - Démarrer Keycloak

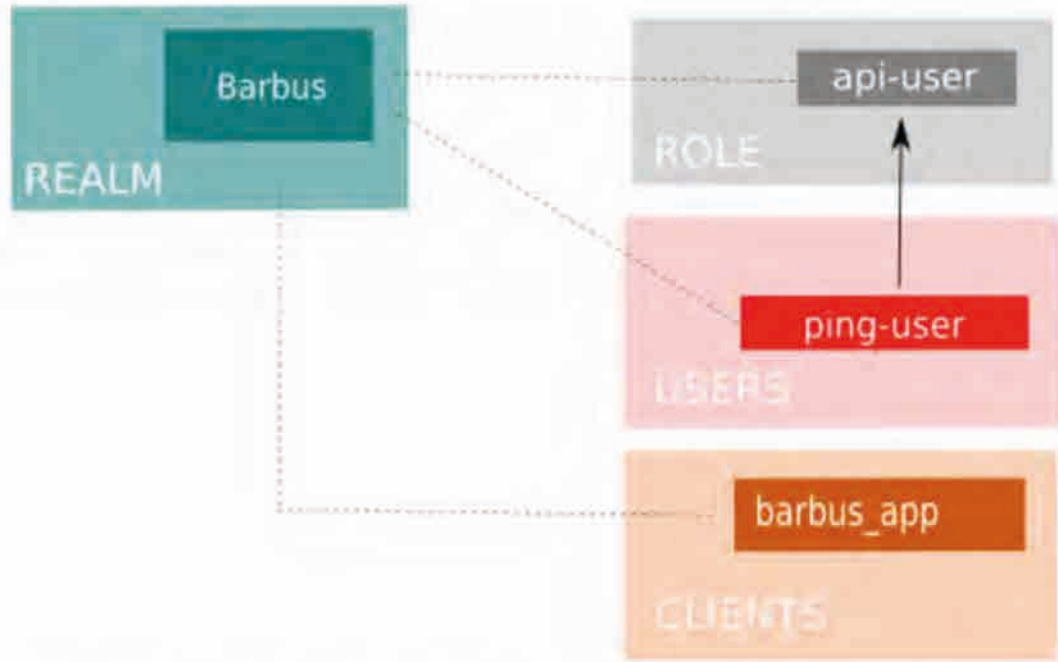
20' ⏱️

- `git checkout lab2`
- Démarrer l'instance Keycloak:

```
docker-compose -f src/main/docker/keycloak.yml up
```

- vérifier en vous connectant sur http://locahost:8080
- identificant de connexion `admin/admin`

Lab2 - Définir de notre royaume de sécurité



Pour plus de détails suivre ce billet de Red Hat

Lab2 - Mettre en place l'authentification à l'aide Keycloak

- modifier la configuration fournie dans `application.properties` pour qu'elle fonctionne avec votre instance keycloak
- vérifier que l'accès au service /account requiert une authentification et que /ping reste accessible

Pour aller plus loin...

- Incident: un utilisateur a été compromis: regardez avec Keycloak comment le déconnecter

Lab3 - Swagger

10' ⏱️

- `git checkout lab3`
- cette branche met en place une documentation en ligne pour notre API Rest avec Swagger
- Tester l'accès et le bon fonctionnement à la documentation: `/swagger-ui.html`
- Ajouter un role `api-doc` dans Keycloak pour limiter l'accès à `swagger`

Lab4 - Mise en place d'un serveur inverse filtrant avec Nginx

25' ⏱️

- `git checkout lab4`
- Prérequis: Construisez le container nginx :

```
./src/main/docker/build-docker-image.sh
```

- Démarrer l'instance nginx:

```
$ docker-compose -f src/main/docker/nginx.yml up
```

- Modifier la configuration fournie pour rediriger les requêtes vers votre service (placer votre configuration dans `src/main/docker/nginx-config/basic-rp/` qui sera monté `/etc/nginx.conf.d/`)

Lab4 - Mise en place d'un serveur inverse filtrant avec Nginx

- Tester avec curl `http://localhost/ping`
- Ajouter une redirection pour aussi "masquer" keycloak derrière ce proxy
- Modifier en conséquence la configuration de votre service et celle de Keycloak
- Interdir les requêtes `POST`, `DELETE`, `HEAD` et `PUT` en direction de `/ping`
- Configurer nginx pour nettoyer les requêtes vers `/ping` en retirant tout paramètres

Pour aller plus loin...

- Limiter les requêtes vers Swagger au strict minimum requis:
- n'autoriser requêtes vers `/swagger-ui*` à part `/swagger-ui.html`, n'autoriser que les `GET`, retirer les arguments des requêtes...

Lab5 - Mise en place du chiffrement SSL devant /ping et swagger

20' ⏱️

- `git checkout lab5`
- regardez la configuration fournie `src/main/docker/nginx-config/conf.d/nginx.ssl.conf`
- générer les certificats nécessaires et ajoutez le volume associé dans `src/main/docker/nginx.yml`
- tester que SSL fonctionne (`https://localhost/`)
- tester que `/ping` and `/account` sont toujours accessible via https

- Documentation pour la génération des certificats et clés nécessaires

Pour aller plus loin...

- Incident: la clef privé a été compromise: révoquez le certificat et déployez un nouveau certificat.

Pause

10' ⏱️

(ou continuer à geeker, bande de nerds)

Lab6 - Intégration avec Spring Security

15' ⏱️

- Frameworking your way out of security...

- (... because we can't `gradle` the shit out of it)

Lab6 - Mise en place de Spring Security

- `git checkout lab6`
- Ajouter Spring Security aux dépendances du projet

```xml
<dependency>
  <groupId>org.keycloak</groupId>
  <artifactId>keycloak-spring-boot-starter</artifactId>
</dependency>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

- Migrer la configuration de KeyCloak du `application.properties` dans une classe qui étend `KeycloakWebSecurityConfigurerAdapter`
- Tester et vérifier que l'intégration avec Keycloak fonctionne toujours sans encombre
- Ref: Spring Boot with Keycloak

Lab6 - CSRF et Cache control

- Vérifier le bon fonctionnement de l'API `/stuff`
- Regardez la documentation détaillant la protection contre le CSRF et la gestion du cache à l'aide de Spring Security
- Jouez avec les requêtes `curl` suivantes:

```
$ curl -v http://localhost:9091/stuff
$ curl -v -H "Content-Type: application/json" \
   -X POST -d '{"title":"un titre","text":"du texte"}' \
   http://localhost:9091/stuff
$ curl -v -H "Content-Type: application/json" \
   -X DELETE \
   -d '{"title":"un titre","text":"du texte"}' \
   http://localhost:9091/stuff
```

- Vérifiez que les messages de retour sont cohérents avec les attentes

```
{"timestamp":1523630530507,"status":403,
"error":"Forbidden", "message":"Could not
verify the provided CSRF token because your
session was not found.","path":"/stuff"}
```

Lab6 - CSRF et Cache control

- Vérifiez la présence des entêtes de sécurité ajoutées par Spring Security:

```
Cache-Control: no-cache, no-store,
    max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Content-Type-Options: nosniff
Strict-Transport-Security:
    max-age=31536000 ; includeSubDomains
X-Frame-Options: DENY
X-XSS-Protection: 1; mode=block
```

Lab6b - Loggez vous avec Github

15' ⏱️

- `git checkout lab6b`
- Nous allons passé de KeyCloak à Github et utiliser GitHub en tant que fournisseur d'identité.
- Nous allons nous appuyer sur la norme ouverte pour l'autorisation appelée oAuth2
  - What the Heck is oAuth ?
- Créez une application OAuth sur Github
  - lisez cette doc https://developer.github.com/apps/building-oauth-apps/authorization-options-for-oauth-apps/#web-application-flow
- Renseignez vos github `clientId` et `clientSecret` dans le fichiers de configuration Spring
- Testez l'application avec votre navigateur: `http://localhost:9091`

Lab6b - oAuth et REST

Essayez de jouer avec notre API avec `curl`

```
curl -v http://localhost:9091/stuff

curl -v -H "Content-Type: application/json" --cookie "JSESSIONID=2E8
```

- pas très REST ?
- et oAuth ?

Lab7 - Base de données

20' ⏱️

- `git checkout lab7`
- on amis en place MongoDB pour vous
- à vous de rajouter une connexion sécurisée (SSL)

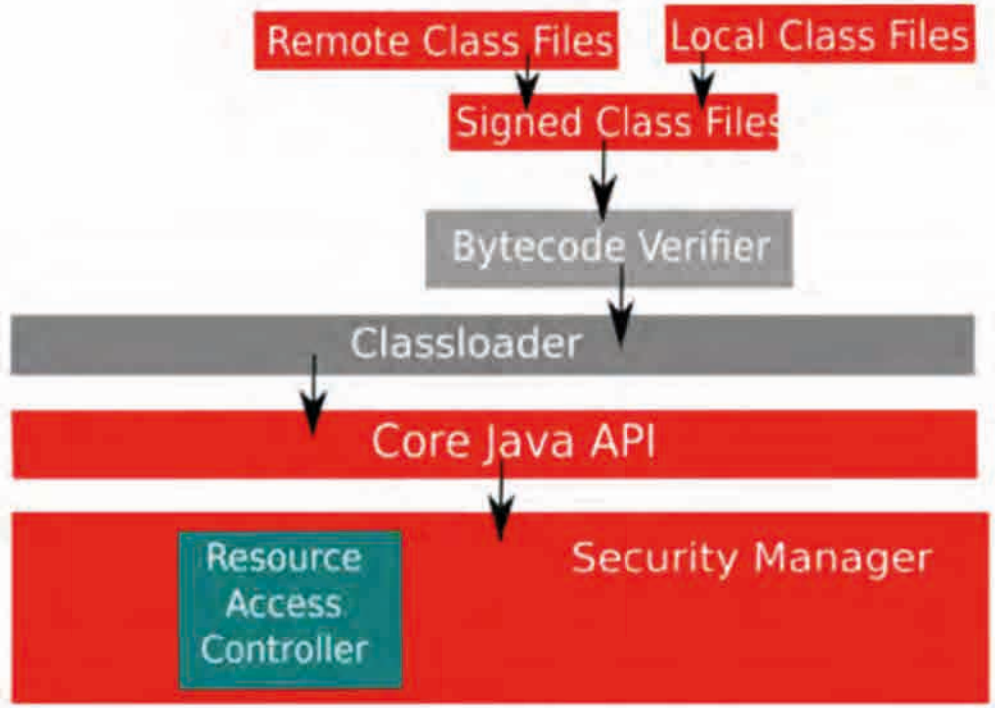
Quelques pointeurs:

- http://mongodb.github.io/mongo-java-driver/3.0/driver/reference/connecting/ssl/?_ga=1.122423051.1001600813.1475930911
- https://docs.mongodb.com/manual/tutorial/configure-x509-client-authentication/
- http://mongodb.github.io/mongo-java-driver/3.0/driver/reference/connecting/ssl/?_ga=1.122423051.1001600813.1475930911

Pour aller plus loin...

- chifrer les données dans mongoDB

Lab8 - Le "Security Manager" (par configuration)



Lab8 - Le "Security Manager" (par configuration)

25' ⏱️

- Démarrez votre service en activant le système manager (sans configuration, "deny all")

```
 $ java ... \
-Djava.security.manager \
-Djava.security.policy=file:///no.policy
```

- Note: si le fichier `no.policy` n'existe pas (ou est vide), aucune autorisation donnée à l'applicatif!!!

Lab8 - Le "Security Manager" (par configuration)

- Avec Spring Boot:

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
      <configuration>
        <jvmArguments>
            -Djava.security.manager
            -Djava.security.policy=file:///no.policy
        </jvmArguments>
      </configuration>
    </plugin>
```

Lab8 - Le "Security Manager" (par configuration)

- Modifiez la configuration suivante pour permettre le démarrage du service avec le "security manager"

```
grant {
   permission java.util.PropertyPermission  "*", "read, wri
   permission java.util.PropertyPermission "java.awt.headle
   permission java.io.FilePermission
    "[path-to]/devoxx-france-lab-2018.git/danslecambouis",
   permission java.lang.reflect.ReflectPermission "*";
};
```

Lab8 - Le "Security Manager" (version programmatique)

- Un début de solution ici: beaucoup de boulot.
- Faisons autrement... Créons notre propre "SecurityManager"!

```
public class PingSecurityManager extends SecurityManager {..
```

- Et injectez le dans votre application:

```
if ( System.getSecurityManager() == null )
    System.setSecurityManager(new PingSecurityManager());
```

- Surchargez les méthodes `checkPermission` (laissez la vide) pour autoriser l'app a accéder aux propriétés
- Surchargez la méthode `checkAccept` pour autoriser le démarrage du serveur

Lab8 - Le "Security Manager" (version programmatique)

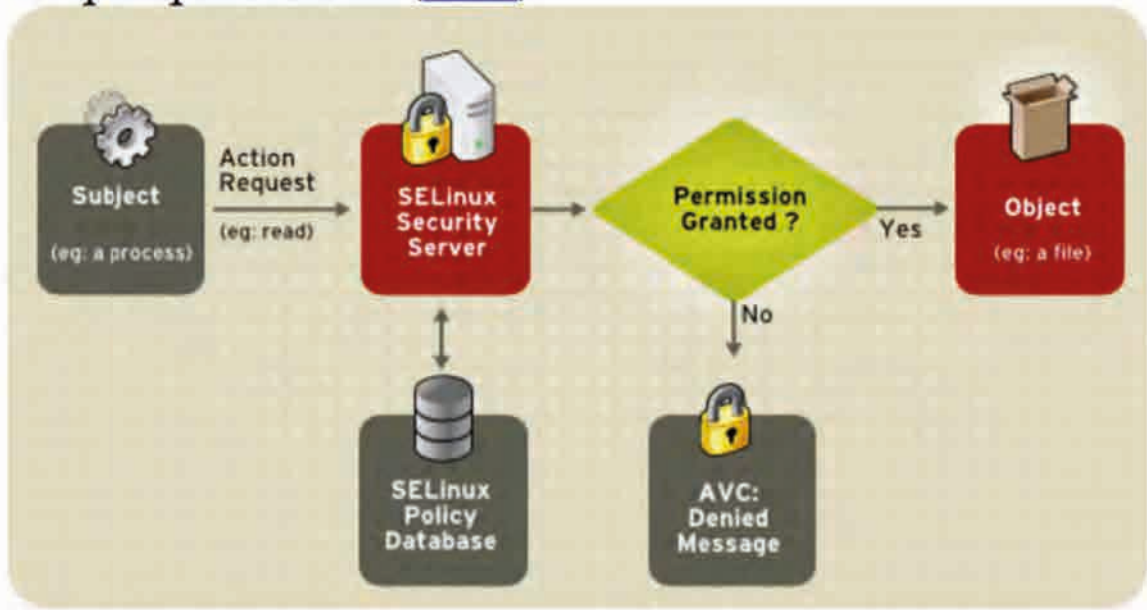
- Ajoutez les méthodes nécessaires pour autoriser l'accès à la base de données

Pour aller plus loin...

- implémentez `checkPermission` de manière à n'autoriser que les accès aux propriétés nécessaires

Lab8 - SELinux

- SELinux en quelques mots... (Docs)



- Tutorial SELinux (Part 1 et Part 2)
- Objectif: Créez un utilisateur `pingservice` appartenant au group `microservices` confiner à `/ping` (volume monté, mappé sur la racine du projet)

Lab8 - SELinux

- Installez les dépendances nécessaires à SELinux (RHEL/Fedora)

```
$ dnf install -y policycoreutils \
policycoreutils-python selinux-policy \
selinux-policy-targeted libselinux-utils\
setroubleshoot-server setools setools-console \
mcstrans java-1.8.0-openjdk-devel
```

- Créez un group et un utilisateur pour exécuter le service:

```
$ sudo groupadd -g 666 microservices
$ sudo useradd -M -u 111 -g 666 -d /ping \
-s /bin/bash  pingservice
```

- Modifiez l'utilisateur pour l'associer au service pour SELinux:

```
$ semanage login -a -s user_u pingservice
```

Lab8 - SELinux

- Vérifiez que l'association a été crée:

```
$ sudo semanage login -l
Login Name           SELinux User         MLS/MCS Range
__default__          unconfined_u         s0-s0:c0.c1023
pingservice          user_u               s0
root                 unconfined_u         s0-s0:c0.c1023
```

- Créez un répertoire /ping et déployez dedans le service ping:

```
$ sudo mkdir /ping
cp -Rv danslecambouis/ /ping/
cp -Rv ~/.m2/ /ping/.m2/
```

- Transférez le répertoire à l'utilisateur pingservice:

```
$ sudo chown -R pingservice:microservices /ping/
```

Lab8 - SELinux

- (Re)activez SELinux:

```
$ sudo setenforce 1
$ getenforce
Enforcing
```

- Démarrez le service avec l'utilisateur pingservice et vérifier que /ping est accessible:

```
$ sudo su - pingservice
$ cd /ping/danslecambouis/
$ ./mvnw spring-boot:run
```

- Modifiez le service `/ping` pour que chaque accès à ce denier soit journaliser dans le fichier `/var/log/ping.log`
- Redémarrez le service et vérifiez que le service ne peut écrire dans le fichier `/var/log/ping.log`

Lab8 - SELinux

Pour aller plus loin...

- Adapter la configuration SELinux pour autoriser l'accès, en écriture, au fichier de journalisation

Lab9 - Boss de fin de nivaeau: DevSec...

30' ⏱️

- Implémenter un fichier de journalisation qui loggue chaque appel à `/ping`
- Adaptez la configuration du `SecurityManager` et de `SELinux` pour autoriser l'accès en écriture au fichier de journalisation `/var/log/microservice/ping.log`
- Potentiellement, une attaque peut changer le contenu du fichier de journalisation - sans que `SELinux` et le `Security Manager` n'interviennent...
- Concevez et implémenter un mécanisme pour assurer que le contenu du fichier ne puisse pas être compromis

Conclusion

- On a fini
- on espère que vous aussi
- et que ça vous a plu !