ALOM - TP 9 - Patterns Cloud

Table of Contents

1. Présentation et objectifs

Le but de ce TP est de mettre en place quelques mécaniques pour les développements orientés cloud.

Nous allons :

- créer des profils pour chacun de nos micro-services
- exposer des métriques avec `spring-boot-actuator`
- charger les properties d'accès à notre base de données depuis un Vault.

2. Création de profils et activation

Aujourd'hui, nos micro-services doivent tourner sur plusieurs environnements distincts :

- notre poste de développeur
- un déploiement d'application Java chez *Clever Cloud*

On pourrait aussi imaginer vouloir créer un troisième environnement, de recette métier par exemple.

2.1. Extraction des profils

Pour chacun de vos micro-services :

- Créez un fichier de configuration `application-clever.properties` Ce fichier contiendra toutes les properties liées à l'environnement d'exécution Clever-Cloud, par exemple les URL des autres micro-services, et l'URL de connexion à la base de données.
- Créez un fichier de configuration `application-local.properties` Ce fichier contiendra toutes les properties liées à l'exécution en local de votre projet, par exemple les URL des autres micro-services en `localhost`, ainsi que les properties `server.port`

À cette étape, vous pouvez vider vos fichiers `application.properties`, dont le contenu a dû être migré dans les deux fichiers `application-local.properties` et `application-clever.properties`.

Il est parfois utile d'avoir des properties communes dans le `application.properties`. Attention par contre, dans le cas d'utilisation d'un profil, les properties du `application.properties` sont d'abord chargées, et ensuite les properties du profil `application-{profil}.properties`.

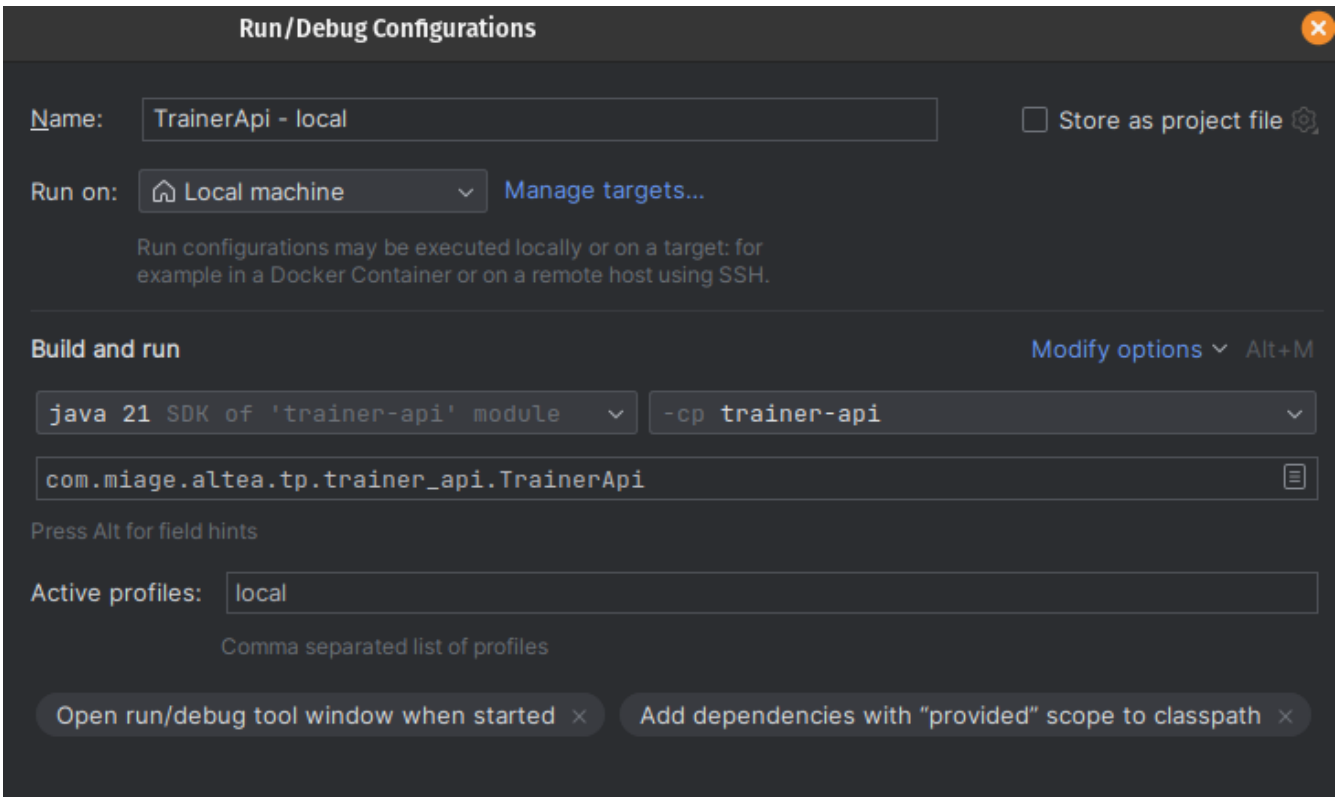
2.2. Activation des profils

2.2.1. Sur votre poste

Lorsque vous démarrez vos micro-services sur votre poste, il vous faut maintenant utiliser le profil `local`. Pour ce faire, vous pouvez indiquer à Spring que le profil local est le profil à utiliser par défaut en absence de tout autre profil.

Pour ce faire, nous allons ajouter un paramètre au lancement de notre application.

Dans IntelliJ, ce paramètre s'ajoute dans la fenêtre de lancement :



Pour les autres IDE, le paramètre peut être passé à la ligne de commande java :

```
java -jar trainer-api.jar --spring.profiles.active=local
```

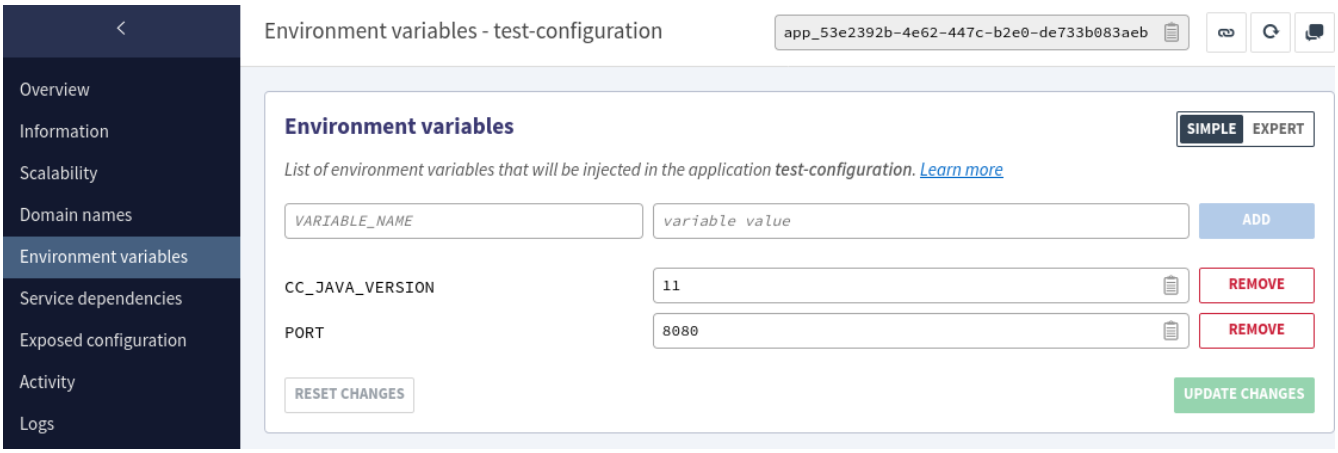
ou positionné avec une variable d'environnement :

```
export SPRING_PROFILES_ACTIVE=local
java -jar trainer-api.jar
```

2.2.2. Sur l'environnement Clever

Pour vos applications déployées sur Clever-Cloud, nous allons utiliser des variables d'environnement.

L'ajout d'une variable d'environnement se fait directement depuis l'onglet d'une application :



Ajoutez à vos applications la variable `SPRING_PROFILES_ACTIVE=clever`.

3. Exposition de métriques

L'exposition de métriques pour nos applications se fait avec `spring-boot-actuator`.

3.1. Dépendance maven

Ajoutez la dépendance maven suivante dans vos projets :

pom.xml

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

Ajouter la dépendance suffit à *spring-boot* pour configurer des routes par défaut.

Démarrez ensuite vos applications. Vous devriez y observer des logs dédiés à actuator au

démarrage :

```
INFO 28827 --- [main] o.s.b.a.e.web.EndpointLinksResolver      : Exposing 2
endpoint(s) beneath base path '/actuator'
TRACE 28827 --- [main] s.b.a.e.w.s.WebMvcEndpointHandlerMapping : Register "{GET
/actuator/health, produces [application/vnd.spring-boot.actuator.v3+json ||
application/vnd.spring-boot.actuator.v2+json || application/json]}" to
java.lang.Object
org.springframework.boot.actuate.endpoint.web.servlet.AbstractWebMvcEndpointHandlerMap
ping$OperationHandler.handle(javax.servlet.http.HttpServletRequest,java.util.Map<java.
lang.String, java.lang.String>)
TRACE 28827 --- [main] s.b.a.e.w.s.WebMvcEndpointHandlerMapping : Register "{GET
/actuator/health/**, produces [application/vnd.spring-boot.actuator.v3+json ||
application/vnd.spring-boot.actuator.v2+json || application/json]}" to
java.lang.Object
org.springframework.boot.actuate.endpoint.web.servlet.AbstractWebMvcEndpointHandlerMap
ping$OperationHandler.handle(javax.servlet.http.HttpServletRequest,java.util.Map<java.
lang.String, java.lang.String>)
TRACE 28827 --- [main] s.b.a.e.w.s.WebMvcEndpointHandlerMapping : Register "{GET
/actuator/info, produces [application/vnd.spring-boot.actuator.v3+json ||
application/vnd.spring-boot.actuator.v2+json || application/json]}" to
java.lang.Object
org.springframework.boot.actuate.endpoint.web.servlet.AbstractWebMvcEndpointHandlerMap
ping$OperationHandler.handle(javax.servlet.http.HttpServletRequest,java.util.Map<java.
lang.String, java.lang.String>)
TRACE 28827 --- [main] s.b.a.e.w.s.WebMvcEndpointHandlerMapping : Register "{GET
/actuator, produces [application/vnd.spring-boot.actuator.v3+json ||
application/vnd.spring-boot.actuator.v2+json || application/json]}" to public
java.util.Map<java.lang.String, java.util.Map<java.lang.String,
org.springframework.boot.actuate.endpoint.web.Link>>
org.springframework.boot.actuate.endpoint.web.servlet.WebMvcEndpointHandlerMapping$Web
MvcLinksHandler.links(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpSer
vletResponse)
```

Consultez enfin pour vos services le endpoint "/actuator/health" :

GET localhost:8080/actuator/health

```
{"status":"UP"}
```

3.2. Activer d'autres endpoints

`spring-boot-actuator` propose de nombreux endpoints par défaut.

https://docs.spring.io/spring-boot/docs/current/reference/html/production-ready-features.html#production-ready-endpoints

Exposez au moins les endpoints suivants :

- `env`
- `metrics`

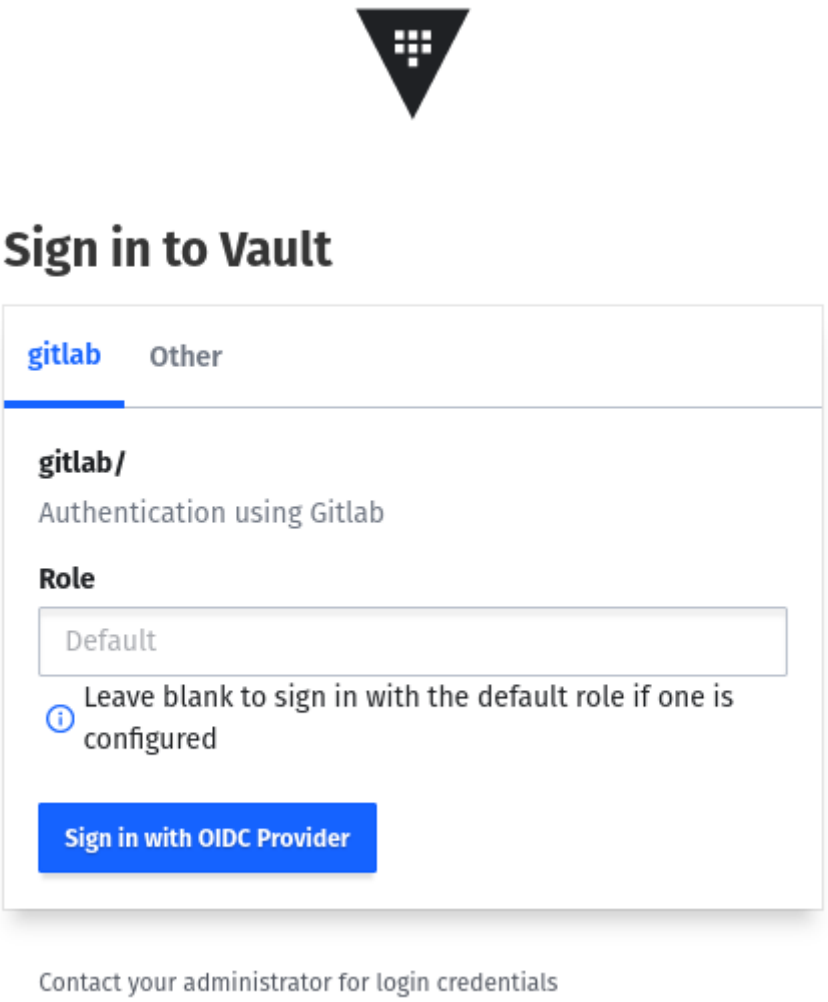
Allez jetter un œil aux endpoints suivants :

- http://localhost:8080/actuator/env
- http://localhost:8080/actuator/metrics
- http://localhost:8080/actuator/metrics/process.cpu.usage

4. Connection au Vault

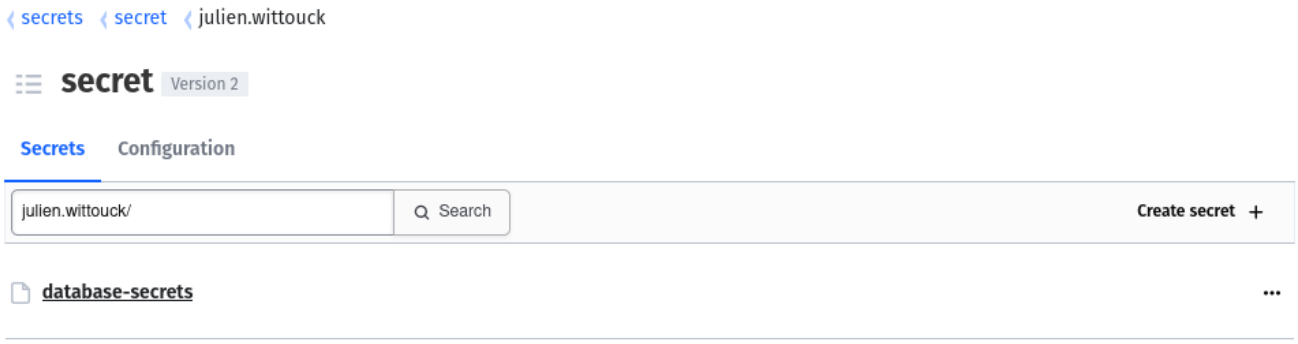
Un serveur *Vault* est disponible à l'adresse suivante : https://vault-alom-2023.cleverapps.io

Vous pouvez vous y connecter avec vos identifiants GitLab (laissez le rôle vide) :

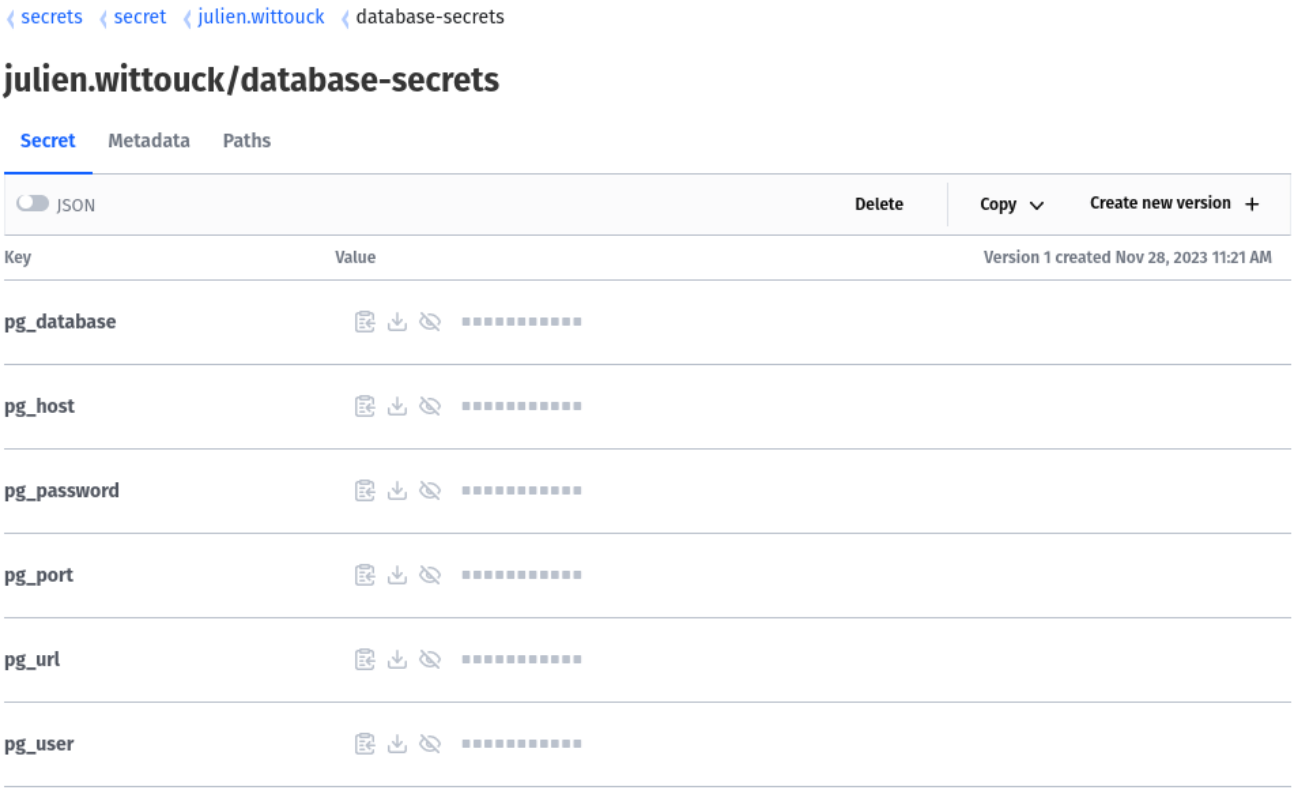


Une fois authentifié, ouvrez le *Secret Engine* nommé *secret/*.

Vous y trouverez un espace pour chacun d'entre vous. Vous avez les droits pour consulter / modifier les secrets qui vous appartiennent.



Un secret *database-secrets* a déjà été initialisé pour vous, avez les informations liées à votre base de données.



4.1. Spring Cloud Vault

Pour connecter votre application au Vault, ajoutez la dépendance suivante à vos projets :

pom.xml

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-vault-config</artifactId>
    <version>4.0.1</version>
</dependency>
```

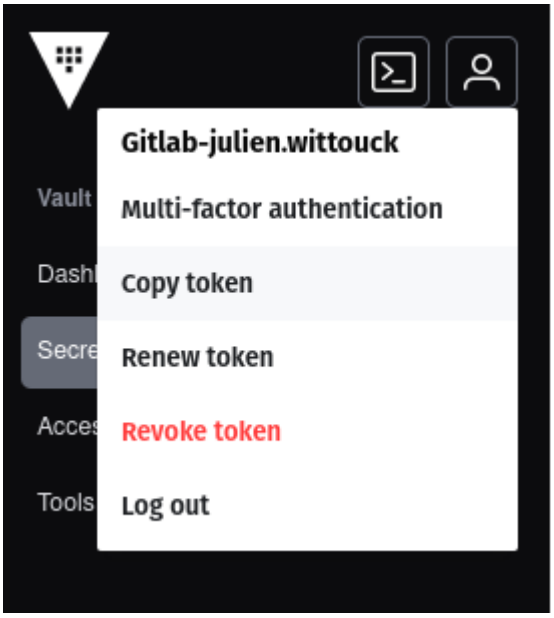
Configurez ensuite les properties suivantes :

application-clever.properties

```
spring.cloud.vault.uri=https://vault-alom-2023.cleverapps.io/
spring.cloud.vault.token=<token>
spring.config.import=vault://secret/<name>/database-secrets
```

Remplacez `<name>` par votre nom d'utilisateur GitLab (se terminant par `.etu`), et en supprimant les `<>`.

Pour le token, une fois connecté au Vault, vous pouvez en récupérer un en dans le menu.



Dans vos properties locales, si vous ne voulez pas utiliser le Vault, vous pouvez aussi ajouter la properties `spring.cloud.vault.enabled=false`.

Comme Clever Cloud exécute les tests avec maven au démarrage de l'application, tout en ayant la variable d'environnement `SPRING_PROFILES_ACTIVE` injectée, vous pouvez aussi ajouter un fichier `src/test/resources/application-clever.properties` vide pour éviter que les tests consomment les properties de prod.

4.2. Reconfiguration des properties

Reconfigurez vos properties, en particulier l'accès à la base de données (url, user, mot de passe). Les properties utilisées par Vault sont accessibles directement, vous pouvez par exemple écrire la properties suivante : `spring.datasource.username=${pg_user}`, la valeur `pg_user` de votre Vault sera chargée au démarrage de l'application.