

# Installation ESUP-SGC

- [Pré-requis](#)
- [PostgreSQL](#)
- [Optimisation PostgreSQL](#)
- [Backup / restauration :](#)
- [Mise à jour de PostgreSQL :](#)
- [Paramétrage mémoire JVM :](#)
- [Sources <https://github.com/EsupPortail/esup-sgc>](#)
- [Tests](#)
- [Packaging \(et compilation\)](#)
- [Déploiement](#)
- [Configurations systèmes](#)
- [Mises à jour](#)
  - [mise à jour depuis un tag](#)
  - [repackaging et redéploiement](#)
- [Configuration](#)

## Pré-requis

- Java OpenJdk 11 : le mieux est de l'installer via le système de paquets de votre linux.
- Maven : le mieux est de l'installer via le système de paquets de votre linux.
- PostgreSQL 9 ou > : le mieux est de l'installer via le système de paquets de votre linux.
- Tomcat (Tomcat 9) : <http://tomcat.apache.org/>
- Apache + libapache2-mod-shib2 : [https://services.renater.fr/federation/documentation/guides-installation/index#installer\\_un\\_sp\\_shibboleth](https://services.renater.fr/federation/documentation/guides-installation/index#installer_un_sp_shibboleth) [la documentation ci-avant reprend également cette partie]
- Git

## PostgreSQL

L'ensemble des données est stocké dans une base de données, photos comprises, cela nous a amené à utiliser PostgreSQL (et non MySQL) pour ses possibilités de streaming sur les blobs.

Sous debian :

```
apt-get install postgresql
```

dans pg\_hba.conf : ajout de

host	all	all	127.0.0.1/32	password
------	-----	-----	--------------	----------

Redémarrage de postgresql

Création de la base :

```
su postgres
psql
create database esupsgc;
create USER esupsgc with password 'esup';
grant ALL ON DATABASE esupsgc to esupsgc;
ALTER DATABASE esupsgc OWNER TO esupsgc;
```

Cette application a été développée en utilisant Spring ROO et donc ses technologies associées.

Comme annoncé ci-dessus, l'application a cependant été développée avec PostgreSQL : lecture/écriture des blobs dans une transaction par streaming ; indexation postgresql (usage de tsvector/tsquery).

Pour une bonne gestion des blob de cette application, il faut ajouter dans PostgreSQL un trigger sur la base de données sur la table big\_file. La fonction lo\_manage est nécessaire ici.

Sous debian :

```
apt-get install postgresql-contrib
```

Puis la création de l'extension lo se fait via un super-user:

avec postgresql 9 et supérieur :

```
apt-get install postgresql-contrib
psql
\c esupsgc
CREATE EXTENSION lo;
```

Et enfin ajout du trigger (afin que les tables soient préalablement créées, notamment la table big\_file sur lequel on souhaite mettre le trigger lo\_manage, il faudra avant cela démarrer une fois esup-sgc (avec le paramètre 'create' dans le persistence.xml)) :

```
CREATE TRIGGER t_big_file BEFORE UPDATE OR DELETE ON big_file FOR EACH ROW EXECUTE PROCEDURE lo_manage
(binary_file);
```

CF <https://www.postgresql.org/docs/9.4/static/lo.html>

Vous devez démarrer l'application une fois. Ne pas oublier ensuite, pour ne pas écraser la base au redémarrage, de modifier src/main/resources/META-INF/persistence.xml : create-> update - cf ci-dessous.

Ajouter la contrainte postgresql supplémentaire :

```
alter table card_desfire_ids ADD CONSTRAINT unique_desfire_ids_desfire_ids_key UNIQUE (desfire_ids,
desfire_ids_key);
```

## Optimisation PostgreSQL

Si les configurations par défaut de PostgreSQL peuvent suffire, il peut être toutefois intéressant de les adapter pour que PostgreSQL puisse mettre à profit les ressources matérielles de votre serveur.

<https://pgtune.leopard.in.ua> peut vous permettre d'obtenir des paramétrages satisfaisants.

Suivant la distribution utilisée, le stats\_temp\_directory peut ne pas être monté en RAM. Sous debian stats\_temp\_directory est bien monté en tmpfs au travers du répertoire /run, mais ce n'est pas le cas sous centos 7 par exemple.

Cela peut être alors très pénalisant et provoquer énormément d'IO disque ou/et l'usage de 100% d'un cpu par un voir plusieurs vacuum postgresql.

**Monter le stats\_temp\_directory en RAM** est une opération rapide, simple et peu coûteuse dont il ne faut pas se priver sur une installation d'un PostgreSQL - la documentation suivante peut aider par exemple : <http://hacksoclock.blogspot.com/2014/04/putting-statstempdirectory-on-ramdisk.html>

## Backup / restauration :

Avec l'utilisateur postgres backup :

```
pg_dump -b -F d -f /backup/esupsgc-dump esupsgc
```

restauration :

```
pg_restore -d esupsgc /backup/esupsgc-dump
```

et la conf CRON

```
11 12,19,23 * * * postgres rm -f /opt/pg-backup/esupnfctag-`date +%A-%HH`.dump.bz2 && pg_dump -f /opt/pg-
backup/esupnfctag-`date +%A-%HH`.dump esupnfctag && bzip2 /opt/pg-backup/esupnfctag-`date +%A-%HH`.dump
21 00 * * * postgres rm -rf /opt/pg-backup/esupsgc-dump && pg_dump -b -F d -f /opt/pg-backup/esupsgc-dump
esupsgc
```

## Mise à jour de PostgreSQL :

Comme on recommande d'utiliser la version de postgresql de votre distribution (debian par exemple), la mise à jour de PostgreSQL se fait simplement via la mise à jour de paquets.

Lors d'une mise à jour de version de votre distribution cependant (apt dist-upgrade), si vous souhaitez profiter de la nouvelle version de postgresql, il faut effectuer des commandes de mises à jour supplémentaires.

debian propose en effet nativement et par défaut la gestion des postgresql en cluster via les commandes pg\_ ; pg\_lsclusters permet par exemple de lister les clusters en place. Aussi lors d'une mise à jour de distribution, la version de PostgreSQL est amené à changer, Debian fait alors cohabiter les deux installations sur le même serveur.

```
# pg_lsclusters
Ver Cluster Port Status Owner   Data directory          Log file
13  main    5432 online postgres /var/lib/postgresql/13/main /var/log/postgresql/postgresql-13-main.log
15  main    5433 online postgres /var/lib/postgresql/15/main /var/log/postgresql/postgresql-15-main.log
```

Le cluster lié à la nouvelle version de postgresql est disponible et opérationnel et en écoute en 5433, le cluster utilisé jusque-là écoute toujours en 5432. Si la possibilité de cluster postgresql peut permettre d'utiliser plusieurs postgresql indépendants sur un même serveur, ici souvent on souhaitera simplement migrer le cluster jusque-là utilisé dans sa nouvelle version, c'ad pour reprendre l'exemple ci-dessus, supprimer ce nouveau cluster 15 (non utilisé) et migrer de version le cluster en version 13 vers la version 15.

Voici les commandes permettant de réaliser cette opération :

- supprimer le cluster postgres 15 créé par défaut :

```
pg_dropcluster --stop 15 main
```

- mettre à jour le postgres 13 (vers 15)

```
pg_upgradecluster --method=upgrade --link 13 main
```

Cette procédure est très explicite et vous avertira en cas d'erreurs. Si à la fin de la procédure vous n'avez pas d'erreurs, PostgreSQL 15 a dû automatiquement prendre le relais sur PostgreSQL 13 (les fichiers de configurations ont été copiés vers la nouvelle version, la base également) et écoute sur le port 5432 (pas de changement).

```
# pg_lsclusters
Ver Cluster Port Status Owner   Data directory          Log file
13  main    5433 down  postgres /var/lib/postgresql/13/main /var/log/postgresql/postgresql-13-main.log
15  main    5432 online postgres /var/lib/postgresql/15/main /var/log/postgresql/postgresql-15-main.log
```

Suite à cette mise à jour, le postgresql 13 a été conservé mais éteint (et a pris un nouveau port d'écoute en 5433).

Durant la procédure, une information indique que vous devez passer un script d'optimisation

Pensez effectivement à le lancer en tant qu'utilisateur postgres.

```
postgres@dgs-13-5912-1752:/usr/lib/postgresql/15/bin/vacuumdb --all --analyze-in-stages
```

- supprimer le "backup" PostgreSQL 13 ; une fois que vous avez bien vérifié que tout fonctionne bien avec la version nouvellement mise à jour (15 ici donc):

```
pg_dropcluster 13 main
```

*Une telle mise à jour d'un postgresql 13 vers un postgresql 15 proposant une base de données esup-sgc avec 100.000 cartes/photos stockées en base de données (~20GB de données) est opérée par ce biais en moins d'1 minute.*

## Paramétrage mémoire JVM :

Pensez à paramétrer les espaces mémoire JVM :

```
export JAVA_OPTS="-Xms1024m -Xmx1024m -XX:MaxPermSize=256m"
```

Pour maven :

```
export MAVEN_OPTS="-Xms1024m -Xmx1024m -XX:MaxPermSize=256m"
```

## Sources <https://github.com/EsupPortail/esup-sgc>

```
cd /opt
git clone https://github.com/EsupPortail/esup-sgc
```

## Tests

Quelques tests junit sont implémentées dans esup-sgc mais ils ne sont pas lancés par défaut, ni à la compilation ni lors du packaging.

Ces tests correspondent plus à des tests d'intégration que des tests unitaires, et peuvent permettre de détecter des problèmes de configuration, de disponibilités de services et de code également bien sûr.

Ils ne couvrent pas toutes les configurations, ni l'ensemble du code, mais ils seront améliorés et peuplés en fonction des retours que l'on pourra avoir.

Pour les lancer, tapez depuis les sources :

```
mvn clean test -DskipTests=false
```

Vous devriez obtenir dans la console quelque chose comme :

```
Results :
Tests run: 10, Failures: 0, Errors: 0, Skipped: 0
[INFO] -----
[INFO] BUILD SUCCESS
```

Vous pouvez également trouver plus de logs dans les fichiers donnés dans le répertoire **target/surefire-reports**

Le fichier `src/test/resources/META-INF/spring/esup-sgc-test.properties` vous permet de spécifier un eppn identifiant un utilisateur sur lequel certaines commandes (de récupération d'information uniquement) seront lancées au travers de certains tests junit.

## Packaging (et compilation)

```
cd /opt/esup-sgc
mvn clean package
```

## Déploiement

On copie/colle le répertoire webapp packagé ainsi dans le tomcat :

```
rm -rf /opt/tomcat-esup-sgc/webapps/ROOT && cp -rf /opt/esup-sgc/target/sgc-1.3.0 /opt/tomcat-esup-sgc/webapps/ROOT
```

On arrête le tomcat avant et on le redémarre ensuite.

## Configurations systèmes

- Logs : `src/main/resources/log4j.properties`
- Base de données :
  - `src/main/resources/META-INF/spring/database.properties` pour paramètres de connexion
  - `src/main/resources/META-INF/persistence.xml` pour **passage de create à update après premier lancement** (création + initialisation de la base de données)
- Mails : `src/main/resources/META-INF/spring/email.properties`

## Mises à jour

Les mises à jour devraient consister à fusionner votre version de sgc avec configurations (et donc commits) internes et le tag nouvellement proposé.

Cela peut se faire via une commande de type **git pull** :

```
git pull origin esup-sgc-1.3.0
```

### **mise à jour depuis un tag**

Lors d'une mise à jour majeure de l'application, lancez la commande suivant pour mettre à jour la base :

```
mvn compile exec:java -Dexec.args="dbupgrade"
```

### **repackaging et redéploiement**

N'oubliez pas alors de repackager et redéployer esup-sgc (cf ci-dessus packagin et déploiement).

## Configuration

[Configurations ESUP-SGC et ESUP-NFC-TAG-SERVER](#)