

# SAÉ 3 - Serveurs

**Ne faites rien d'illégal ni d'abusif<sup>1</sup> sur ces serveurs**  
**Si vous vous faites bannir, tous les serveurs seront bannis.**

## SSH

**SSH** (Secure **SH**ell) est à la fois un protocole<sup>2</sup> et un outil permettant d'accéder à un serveur de façon sécurisée.

Même si dans SSH il y a le nom de Shell, accéder à une machine en SSH peut signifier se connecter à cette machine pour y travailler mais aussi pouvoir envoyer ou récupérer des données (fichiers).

“De façon sécurisée” signifie que la liaison entre le client (votre poste de travail) et la machine cible (généralement un serveur) fait intervenir du chiffrement de bout en bout.

## Clés

Le mécanisme de chiffrement dans SSH met en œuvre des clés de chiffrement asymétriques. Cela signifie qu'un jeu de deux clés est utilisé : l'une, qui est publique, permet de chiffrer les messages, alors que la seconde, qui est privée, permet de déchiffrer tout message chiffré avec la clé publique.

L'asymétrie vient du fait que ce qui est chiffré avec une clé ne peut pas être déchiffré avec la même clé.

Dans cette paire de clés “publique-privée”, la clé publique peut être donnée à n'importe qui mais la clé privée doit rester secrète et ne jamais quitter l'ordinateur de son propriétaire.



Si la clé privée venait à fuiter suite à une intrusion ou une maladresse du propriétaire (copie sur une clé USB qu'on prête, envoi par mail, envoi sur un dépôt Git, etc.) il faudra la considérer comme compromise et il faudra la remplacer au plus vite et ne plus l'utiliser nulle part. C'est le b-a-ba de la sécurité !



<sup>1</sup> Téléchargements illégaux, surconsommation de bande passante, attaque, intrusion, etc.

<sup>2</sup> cf le cours de réseau ou encore le cours sur les Web Services, à venir en R4.01

## Principe de fonctionnement

Supposons deux acteurs dans un échange informatisé : **A** et **B**. Ces deux acteurs ont créé chacun une paire de clés asymétriques :

 PubA +  PrivA







 PubB +  PrivB

Quand la connexion entre les deux acteurs se met en place, chacun des deux communique à l'autre sa clé publique :

A envoie  PubA à B

B envoie  PubB à A

Cet échange n'a pas besoin d'être sécurisé car ces clés sont publiques. Ça tombe bien car, à ce stade de la rencontre des deux acteurs, rien n'est encore en place pour qu'ils puissent communiquer entre eux de façon sécurisée !

Puisque ce qui est chiffré à l'aide de la  **PubA** ne peut être déchiffré qu'à l'aide de la  **PrivA**, et inversement pour l'autre paire de clés, et que ces clés  **PrivA** et  **PrivB** restent conservées bien à l'abri chez **A** (pour la  **PrivA**) et chez **B** (pour la  **PrivB**), les deux acteurs peuvent donc déchiffrer les messages que l'autre acteur communique de façon chiffrée.

Voici donc résumé le chemin d'un message de **A** vers **B** :

**Message en clair** ➡ chiffrement par **A** avec la  **PubB** ➡

u66rk7bñ·n@k\$?????€


🔒 66K7bñ-ñ0%\$??????€[] ➡ déchiffrement par **B** avec sa  **PrivB** ➡

## Message en clair

## Usage des clés

Les clés peuvent aussi servir à autoriser un accès à une machine (généralement un serveur), en plus de chiffrer les échanges.

Même si les deux acteurs s'échangent leurs clés publiques pour discuter, ça ne donne pas automatiquement une autorisation de se connecter sur la machine de l'autre, heureusement !

Pour autoriser **A** à se connecter sur **B**, il faut que **B** ajoute la  PubA dans sa liste blanche de clients autorisés à se connecter chez lui. C'est évidemment **B** qui décide de cela, il est quand même maître chez lui !

## Génération des clés

Si vous ne disposez pas déjà d'une paire de clés, il faut en générer une en tapant ceci :

```
ssh-keygen -t rsa
```

Lors de la génération, des questions vous sont posées. Répondez-y ainsi :

- Nom du fichier à générer, laissez ce qui est proposé (**id\_rsa**). Si vous avez déjà une clé **id\_rsa**, choisissez un autre nom, mais vous aurez 2 paires de clés ! Est-ce bien ce que vous voulez ? N'écrasez pas une clé **id\_rsa** déjà existante sans avoir conscience de ce que vous faites car, si vous l'utilisez déjà ailleurs, cette clé sera perdue à jamais.
- Passphrase : une phrase clé qui chiffre (qui protège) votre clé privée. C'est une sécurité supplémentaire au cas où votre clé privée venait à fuir, ce qui n'est jamais bon de toute façon. Pour votre usage à l'IUT, vous pouvez ne rien mettre. Sinon il faudra saisir cette phrase clé à chaque fois<sup>3</sup>.  
Notez quand même que cette clé étant sur un serveur de l'IUT, un administrateur y a donc accès. Ce n'est pas aussi "privé" que sur un ordinateur personnel. Pensez aussi que vos sauvegardes d'ordinateur personnel (vous en faites bien une, n'est-ce pas ?) devraient idéalement se faire sur un disque chiffré aussi.

Les deux clés générées se trouvent désormais dans le dossier **~/.ssh** et s'appellent **id\_rsa** (clé privée) et **id\_rsa.pub** (clé publique).

## Installation des clés

### Cas 1 : Installation par un administrateur

Vous pouvez maintenant communiquer votre clé publique (**id\_rsa.pub**) à qui vous voulez, notamment à l'administrateur d'un serveur pour qu'il vous place dans sa liste blanche.

SAÉ 2023 : envoyez-moi votre clé publique par mail (**[gildas.quiniou@univ-rennes.fr](mailto:gildas.quiniou@univ-rennes.fr)**)

### Cas 2 : Installation depuis une machine déjà autorisée

Une autre solution pour installer votre clé publique dans la liste blanche du serveur est de disposer déjà d'un accès à la machine en question, depuis une autre machine. Par exemple, vous avez un accès depuis le PC de l'IUT et vous voulez ajouter un accès pour

---

<sup>3</sup> Il y a possibilité de ne saisir la phrase clé qu'une seule fois tant que la session n'est pas fermée.

vosre PC personnel. Dans ce cas, votre PC de l'IUT peut être utilisé pour installer la clé de votre PC personnel puisque votre compte sur le PC de l'IUT est déjà dans la liste blanche du serveur.

Donc, vous pouvez soit le faire vous-même soit charger quelqu'un de votre équipe de le faire, sous réserve que cette autre personne dispose déjà d'un accès.

Voici les étapes à suivre :

- Fournissez votre clé publique **id\_rsa.pub** à celui qui va faire l'ajout
- Celui qui fait l'ajout ne doit surtout pas remplacer son propre **id\_rsa.pub** par le vôtre sur sa machine (dans son dossier **.ssh**) ! Il doit placer le vôtre sur son bureau par exemple.
- Il doit ensuite lancer cette commande depuis le dossier où se trouve votre id\_rsa.pub (par exemple depuis son bureau) :

```
ssh-copy-id votre_id_rsa.pub login_de_connexion@ip_ou_nom_du_serveur
```

## SSH à l'IUT

A l'IUT, SSH n'est pas utilisable pour sortir vers une machine externe. Chez vous il n'y aura aucun problème.

Nous allons donc voir un outil complémentaire qui ne sera nécessaire que depuis les machines de l'IUT, il s'agit de Putty.

## Putty

Putty est un client SSH conçu initialement pour Windows car Linux dispose déjà d'un client SSH natif (la commande **ssh**).

S'il n'était qu'un simple client SSH, Putty serait resté dans le monde Windows.

Mais Putty offre d'autres fonctionnalités, dont celle de pouvoir sortir vers l'extérieur en SSH via un proxy Socks.

C'est cette fonctionnalité que nous allons mettre en œuvre uniquement à l'IUT pour vous permettre de vous connecter au serveur que nous mettons à votre disposition pour la SAÉ.

## Format Clé SSH

Putty utilise un autre format de clé que celui produit par la commande **ssh-keygen** utilisée précédemment. Il faut donc convertir à l'aide de cette commande, à exécuter dans un Terminal :

```
cd ~/.ssh  
puttygen id_rsa -o id_rsa.ppk
```

Attention, cette clé **id\_rsa.ppk** est la même que **id\_rsa**, c'est juste un reformatage du fichier. **PPK** signifie **P**utty **P**rivate **K**ey, et c'est donc un format qui n'est adapté qu'à Putty. De ce fait, n'envoyez pas ce fichier **id\_rsa.ppk** vers le serveur. En plus, comme il s'agit d'une clé privée, elle doit donc, elle aussi, rester bien au chaud sur votre ordinateur et ne servir qu'à Putty.

## Configuration

Lancez l'application Putty depuis le menu de l'interface graphique Linux.

La fenêtre **Putty configuration** s'affiche dès le lancement de l'application.

Remplissez les champs suivants :

- **Session > Host name (or IP address)** : ce qui vous a été communiqué comme IP ou nom de serveur pour la SAÉ
- **Port** : **22** (valeur par défaut)
- **Saved sessions** : donnez le nom que vous voulez. Vous reviendrez ici pour cliquer sur Save en fin de configuration (ou Load pour charger une configuration plus tard)
- **Connection > Proxy** :
  - **Proxy type** : **SOCKS 5**
  - **Proxy hostname** : **148.60.237.99**
  - **Port** : **2222**
  - **Username** : votre login de compte PC
  - **Password** : votre mot de passe associé
- **Connection > SSH > Auth > Private key file for authentication** :  
**/home/etuinfo/votre\_login/.ssh/id\_rsa.ppk** (adapter la partie **votre\_login**)
- **Connection > SSH > Tunnels** :
  - **Source port** : **10220**
  - **Destination** : **127.0.0.1:22**
  - **Local**
- **Connection > Data > Auto-login username** : **debian**

Une fois cette configuration prête, n'oubliez pas de la sauvegarder (bouton **Save** dans la section **Saved sessions**).

Vous pouvez désormais cliquer sur le bouton **Open** pour établir une connexion. Vous allez obtenir un Terminal connecté au serveur.


Vous pouvez travailler dans ce Terminal mais, en plus de ce Terminal, Putty a mis en place un tunnel SSH qui va vous permettre de travailler avec VSC connecté au serveur. Ce tunnel n'est pas un VPN mais s'y apparente un peu et vous permet un accès au serveur comme s'il était sur le réseau local. SSH s'occupe de faire transiter la partie du trafic réseau destiné au serveur, sur ce tunnel chiffré.

## VSC

Cette partie est valable pour les PC de l'IUT ainsi que pour votre propre machine personnelle.

Pour vous connecter au serveur, il vous faut installer l'extension **Remote development** de Microsoft. Cette extension est un package qui en installe 4.

## Configurer l'hôte SSH

Cliquez le bouton  en bas à gauche de la fenêtre, ce qui ouvre un menu en haut de fenêtre.

Choisissez **Se connecter à l'hôte...** dans ce menu.

Choisissez **Configurer les hôtes SSH...** en bas du menu suivant.

Choisissez la ligne **/home/etuinfo/votre\_login/.ssh/config** dans le menu suivant

Ça doit ouvrir le fichier **config** dans l'éditeur VSC. Si ce n'est pas le cas, recommencez ou interrogez l'enseignant·e.

### Cas d'un PC de l'IUT

Placez ceci dans le fichier **config** :

```
Host ServeurSAE
  HostName 127.0.0.1
  User debian
  Port 10220
```

Sauvez et fermez **config**.

## Cas d'un PC personnel

Placez ceci dans le fichier **config** :


```
Host ServeurSAE
  HostName ip_ou_nom_du_serveur
  User root
```

Sauvez et fermez **config**.

## Se connecter au serveur

A l'IUT uniquement : Lancez Putty avant tout.

A l'IUT ou chez vous :

Cliquez le bouton  en bas à gauche de la fenêtre, ce qui ouvre un menu en haut de fenêtre.

Choisissez **Se connecter à l'hôte...** dans ce menu.

Choisissez **ServeurSAE** dans la liste

Si tout est bien configuré et, dans le cas d'une connexion depuis l'IUT, si Putty a été lancé et ouvert sur la bonne session, vous devriez obtenir une connexion au serveur dans VSC.

Il ne vous reste plus qu'à choisir d'ouvrir un dossier (distant) sur le répertoire de travail souhaité.

Vous avez aussi à votre disposition un Terminal dans VSC, qui vous permettra de lancer des commandes sur le serveur auquel vous êtes connecté.

## Root

Quand vous vous loguez sur le serveur, vous êtes l'utilisateur **debian**. Pour travailler avec Docker notamment, vous devrez passer en **root**. Soit vous préfixez vos commandes avec un **sudo**, soit vous passez en utilisateur **root**, avec toutes les précautions d'usage à respecter ensuite !

Pour passer en **root** sur le serveur utilisez la commande :

```
sudo su
```

# Docker

Docker est installé sur vos serveurs. Il est peut-être plus aisé de travailler en **root** avec Docker.

Vous trouverez un dossier **/docker** contenant l'environnement suivant :

- Un conteneur serveur Web **nginx**
- Un conteneur reverse-proxy permettant un accès SSL au serveur Web et une gestion automatique d'un certificat Let's Encrypt.
- Un conteneur moteur de base de données MariaDB
- Un conteneur PHPMysqlAdmin pour administrer la base de données MariaDB
- Un conteneur moteur de base de données PostgreSQL
- Un conteneur PgAdmin pour administrer la base de données PostgreSQL

## Bases de données

Vous choisissez la base de données que vous voulez entre MariaDB et PostgreSQL.

Vu du conteneur Web, le serveur MariaDB s'appelle **mariadb**, port **3306**

Vu du conteneur Web voit le serveur PostgreSQL s'appelle **postgresdb**, port **5432**

L'utilisateur par défaut sur chacune de ces bases s'appelle **sae3** et son mot de passe vous sera communiqué par équipe par votre enseignant.

Une base de données vierge a aussi été créée sur chacun de ces serveurs et s'appelle aussi **sae3** sur chacun d'eux.

## FQDN<sup>4</sup>

Les FQDN de votre serveur vous seront communiqués par l'enseignant.

Exemples pour l'équipe **biloute** :

- **https://biloute.ventsdouest.dev** Pour le site Web
- **https://dbadmin-biloute.ventsdouest.dev** Pour l'admin BDD

## Travail sur le serveur

Attention, évitez de travailler directement sur le serveur.

Votre dossier de travail Web est dans **/docker/sae/data/html**

---

<sup>4</sup> Fully Qualified Domain Name. Exemples : **google.com** ou **etudiant.univ-rennes.fr**



Utilisez Git, travaillez chez vous, committez, poussez sur le dépôt puis pulllez le dépôt vers le serveur.

**On sera attentif à vos commits.**

Pour la partie BDD, à vous de voir...

## **.env (Ne changez rien dans ce fichier !!!)**

Un fichier **.env** dans **/docker/sae**. contient des variables d'environnement pour Docker.

## **Git**

Votre dépôt Git doit contenir, à sa racine, un dossier **html/** qui représente la racine du site Web.

Vous devez cloner votre dépôt dans **/docker/sae/data/web** sur le serveur Web.

De cette façon, votre dépôt Git (représenté par le dossier caché **.git**), en étant un niveau au-dessus de la racine du serveur Web (dans **/docker/sae/data/web**) sera protégé d'un accès externe par Internet. C'est une question de sécurité fondamentale à bien comprendre. Demandez à votre enseignant si ce n'est pas clair.

Sur votre machine personnelle, vous devez faire de même et voici le workflow de votre développement :

- Vous travaillez sur votre machine personnelle
- Vous **commitez** vos travaux et vous **poussez** vers Git(hub | lab)
- Sur le serveur, vous **pulllez** depuis Git(hub | lab)
- Vous ne travaillez pas sur le serveur, ce qui signifie que vous ne devez jamais faire de **push** depuis le serveur, sinon vous avez raté quelque chose !

Pour la partie BDD, idéalement vous devez aussi travailler sur votre machine personnelle et prévoir un mécanisme de mise à jour de la BDD du serveur afin qu'elle soit synchro avec le code **push/pullé**

Prévoyez des fichiers de configuration (**.env** par exemple) que chacun pourra adapter sur sa machine personnelle et sur le serveur, mais ne placez pas ces fichiers sous la supervision de Git sinon vous allez vous marcher sur les pieds mutuellement à chaque **push/pull**. Mettez dans Git des fichiers "modèles" de ces fichiers de configuration (**.env.template** par exemple) et adaptez vos fichiers de configuration personnels si et quand nécessaire.

## **PHPMyAdmin**

Utilisez comme login et mot de passe les valeurs contenues dans les variables **DB\_USER** et **MARIADB\_PASSWORD** du fichier **.env**. Vous avez accès directement à la base **sae**.

## PgAdmin

Utilisez comme login `${EQUIPE}@dbadmin-sae.com` où `${EQUIPE}` est à remplacer par la valeur contenue dans la variable `EQUIPE` du fichier `.env` et comme mot de passe la valeur contenue dans la variable `PGADMIN_PASSWORD` du fichier `.env`.

Ce qui suit suppose que vous ayez choisi la langue française dans la page de login.

Une fois connecté, dans le **Tableau de bord**, choisir **Ajouter un nouveau serveur** et renseigner les informations suivantes :

- **Général** > **Nom** : SAE
- **Connexion** :
  - **Nom d'hôte / Adresse** : postgresdb
  - **Port** : 5432
  - **Base de données de maintenance** : sae
  - **Nom d'utilisateur** : la valeur de `DB_USER` dans le fichier `.env`
  - **Mot de passe** : la valeur de `DB_ROOT_PASSWORD` dans le fichier `.env`

## Paramètres pour les scripts PHP (dans conteneurs)

Voici maintenant les paramètres à utiliser dans vos scripts PHP pour accéder aux BDD :

- **Host** = mariadb pour MariaDB ou postgresdb pour PostgreSQL
- **Port** = 3306 pour MariaDB ou 5432 pour PostgreSQL, mais comme ce sont les ports par défaut, généralement il n'est pas nécessaire de les préciser.
- **User** = sae
- **Password** = le contenu de la variable `MARIADB_PASSWORD` dans le fichier `.env` pour MariaDB ou le contenu de la variable `DB_ROOT_PASSWORD` pour PostgreSQL.
- **Database** = sae

## Sauvegardes

Vous êtes responsable de votre serveur.

**Vous devez vous occuper de vos sauvegardes.**

Si vous ne faites pas vos sauvegardes très régulièrement et que vous perdez vos données, ce ne sera **pas une excuse recevable** pour ne pas terminer le projet dans les temps. Vérifiez aussi que les sauvegardes sont **lisibles et complètes**.

Vous êtes des informaticien·ne·s en formation et ceci fait partie de notre métier !

## Procédure de backup

Tout votre environnement de travail est dockerisé.

Les volumes sont dans **/docker/data**. Vous avez uniquement besoin de sauvegarder ce dossier.

Sur le serveur, arrêtez d'abord votre environnement Docker (en **root**) :

```
cd /docker/sae
docker compose down
```

On supposera que vous restez dans ce dossier ensuite.

Créez une archive :

```
tar czvf data.tgz data
```

Depuis votre PC personnel (via VSC ouvert sur le serveur par exemple), récupérez cette archive et conservez-la avec un nom et une date sur votre PC personnel. Conseil : faites-le sur au moins 2 PC, on ne sait jamais !

N'oubliez pas de redémarrer votre environnement Docker :

```
docker compose up -d
```

Jetez un œil aux logs durant le démarrage :

```
docker compose logs -f
```

Vous pouvez ensuite **CTRL+C**, ça interrompt l'affichage des logs, pas les conteneurs.

## Procédure de restauration

ATTENTION : il est conseillé de faire un backup avant de restaurer quoi que ce soit !

Sur le serveur, arrêtez d'abord votre environnement Docker (en **root**) :

```
cd /docker/sae
docker compose down
```

On supposera que vous restez dans ce dossier ensuite.

Déposez un **data.tgz** à restaurer (récupéré depuis une sauvegarde précédente) dans **/docker/sae**.

Désarchivez :

```
tar xzvf data.tgz
```

Attention, ça écrase tout !

N'oubliez pas de redémarrer votre environnement Docker :

```
docker compose up -d
```

Jetez un œil aux logs durant le démarrage :

```
docker compose logs -f
```

Vous pouvez ensuite **CTRL+C**, ça interrompt l'affichage des logs, pas les conteneurs.

Il est peut-être intéressant d'expérimenter dès le départ un backup suivi d'une modif BDD et de quelques fichiers suivi d'une restauration du backup. Ca évitera de faire des bêtises le jour où vous devrez le faire dans le stress 😊

Note : le désarchivage écrase tout mais ne supprime pas des fichiers qui n'étaient pas présents dans le backup. Si vous voulez vraiment avoir une restauration propre, renommez **data/** en **data.old/** par exemple, avant la restauration (mais après l'arrêt de l'environnement Dockerisé évidemment)

Dans le doute, demandez à votre admin préféré...