

# Admineasy

Rapport de projet tuteuré - Mars 2018

Mélissa Da Costa

Antoine Dujardin

Chloé Trugeon

Kylian Bohl

Tuteur : Régis Brouard



# Remerciements

Nous tenons à remercier Régis Brouard d'avoir accepté d'être notre tuteur pour ce projet. Même si nous ne l'avons que peu consulté ses conseils se sont avérés fort utiles et nous ont permis de partir dans la bonne direction.

Nous remercions également les enseignants de l'IUT pour leurs cours, en particulier Konstantin Verchinne pour tout ce qui est réseau et systèmes et Denis Monnerat pour le Web.

|  |           |
|--|-----------|
| <b>Introduction</b>                                | <b>3</b>  |
| <b>Préparation</b>                                 | <b>4</b>  |
| Etude de l'existant                                | 4         |
| Étude de l'environnement                           | 4         |
| État des lieux                                     | 4         |
| Analyse des besoins                                | 6         |
| <b>Organisation</b>                                | <b>7</b>  |
| Définition de la procédure                         | 7         |
| Répartition  | 8         |
| <b>Bases de données</b>                            | <b>9</b>  |
| PostgreSQL   | 9         |
| InfluxDB   | 10        |
| <b>Client</b>                                      | <b>11</b> |
| Présentation                                       | 11        |
| Fonctionnement                                     | 12        |
| Diagrammes   | 13        |
| Logiciels externes                                 | 15        |
| Informations récupérées                            | 16        |
| Installation                                       | 17        |
| Désinstallation                                    | 20        |
| <b>Serveur</b>                                     | <b>21</b> |
| Les différents serveurs et leurs fonctions         | 21        |
| Mise en place                                      | 26        |
| <b>Interface web</b>                               | <b>27</b> |
| Style et critères ( d'une Interface Homme Machine) | 27        |
| Fonctionnalités                                    | 29        |
| Lien avec le serveur                               | 32        |
| Diagrammes UML                                     | 33        |
| Connection a InfluxDB                              | 34        |
| <b>Améliorations possibles</b>                     | <b>35</b> |
| <b>Le site officiel</b>                            | <b>37</b> |
| <b>Difficultés rencontrées</b>                     | <b>38</b> |
| <b>Conclusion du groupe</b>                        | <b>39</b> |
| <b>Conclusions personnelles</b>                    | <b>40</b> |
| <b>Glossaire</b>                                   | <b>41</b> |

# Introduction

Dans tous les milieux professionnels, l'informatique est essentielle. Toutes les structures doivent gérer un parc informatique, et la surveillance des machines est un élément complexe à intégrer. Si cette surveillance est efficace, elle permettra de détecter des problèmes rapidement.

Une problématique a rapidement été créée en se basant sur le modèle de l'IUT, qui comporte une centaine d'ordinateurs. L'objectif est d'avoir une vue d'ensemble du parc informatique, ainsi que des caractéristiques de chaque machine et ses statistiques de fonctionnement, afin d'identifier un éventuel problème et pouvoir intervenir au plus vite.

Ce projet est composé de trois éléments essentiels : le client, le serveur, et une interface web qui se servent chacun de bases de données. Cet angle transversal a beaucoup motivé la réalisation de ce projet : chacun a un domaine privilégié. Ce projet a mis en valeur la communication entre différents langages et dispositifs informatiques: serveurs, bases de données, clients, interfaces homme-machine. Le principal objectif a été de réaliser cette application, Admineasy, de façon à ce qu'elle soit non seulement la plus riche en termes de fonctionnalités, mais aussi la plus facile à mettre en œuvre.

On trouvera dans ce document les différentes étapes de la réalisation d'Admineasy, avec tout d'abord ses objectifs, et les fonctionnalités prévues. Puis les trois éléments essentiels de sa création : le client, le serveur et l'interface web, et leur fonctionnalités.

## Préparation

### Etude de l'existant

#### Présentation générale de l'établissement

Admineasy a pour but d'aider les administrateurs réseau à lister tous les ordinateurs présents sur leur réseau, et connaître leur état. Il serait utilisé dans tous types d'entreprises.

### Étude de l'environnement

Bien que ce ne soit pas le métier principal de l'entreprise, administrer un réseau est nécessaire dans toutes les entreprises. Une application comme AdminEasy participe donc à un large ensemble de métiers, notamment dans les secteurs secondaire et tertiaire. Elle se place principalement dans un marché B TO B (business to business), mais peut aussi être dans un marché B TO C (business to client).

### État des lieux

Avec l'environnement concurrentiel de Porter, on peut analyser plusieurs aspects en lien direct avec la concurrence comme:

#### Pouvoir de négociation des clients

En effet, les administrateurs réseaux étant nombreux, ils exercent une certaine force qui oriente les créateurs d'applications sur ce qu'ils désirent.

#### Pouvoir de négociation des fournisseurs

Dans notre cas, les "fournisseurs" sont les développeurs qui créent l'application qui est ensuite ajoutée sur le marché. Ici, pour administrer un réseau, plusieurs fonctionnalités peuvent être mises en place et les administrateurs réseaux n'ont pas tous les mêmes besoins. Certains souhaitent avoir accès à des fonctionnalités spécifiques ce qui n'est pas toujours disponible, ils sont alors contraints d'utiliser une application qui ne leur convient pas entièrement. Donc cela donne plus de pouvoir aux développeurs qui auront toujours des clients car les administrateurs réseau sont obligés de se servir de certains outils pour exercer leur travail.

#### Menace des produits ou services de substitution

Plusieurs systèmes de surveillance d'ordinateurs pour entreprise existent déjà, ce qui laisse aux administrateurs réseau un certain choix. Ils ont donc la possibilité de changer d'application.

Parmi ces choix, on retrouve les suites [Nagios](#) et [Shinken](#) qui permettent d'effectuer le monitoring des

systèmes, des protocoles, des applications ou des bases de données notamment.

Il existe encore un très grand nombre de logiciels similaires comme on peut le voir sur ces recherches google :

<https://www.google.fr/search?q=enterprise+monitoring+tools>

<https://www.google.fr/search?q=enterprise+ticketing+systems>

## Menace d'entrants potentiels sur le marché

Sachant que les besoins des administrateurs réseaux peuvent évoluer rapidement au fil des avancées technologiques et que le monde de l'informatique est très mouvant, il peut y avoir à tout moment une nouvelle application qui répond aux besoins des administrateurs réseaux.

## Degré de concurrence

Comme dit précédemment, les besoins des administrateurs réseau peuvent varier d'un administrateur à l'autre. C'est pour cela qu'il n'y a pas une très grande rivalité (pour le moment) car il n'existe pas deux applications très similaires.

# Analyse des besoins

## Description des besoins

Le besoin principal est la surveillance des ordinateurs. L'administrateur doit pouvoir accéder aux caractéristiques de tous les ordinateurs dans son réseau, ainsi que leur utilisation. L'administrateur pourra aussi définir des alertes, par exemple si un ordinateur chauffe trop.

Si possible, nous ajouterons un système de rapport d'erreur. Les utilisateurs pourront créer des rapports d'erreurs, qui seront envoyés à l'administrateur. Cette fonctionnalité n'est pas essentielle à l'application.

## Définition de l'objectif du projet

L'objectif de l'application Admineasy est d'avoir une vue d'ensemble d'un parc informatique.

La façon la plus simple d'accéder à cette vue d'ensemble est l'interface web. Celle-ci permet, via un simple site internet, d'avoir accès à toutes les informations dont un administrateur réseau pourrait avoir besoin. Cette interface a pour objectif de rassembler les caractéristiques générales du parc informatique, mais aussi quelques caractéristiques plus précises afin de détecter d'éventuels dysfonctionnements.

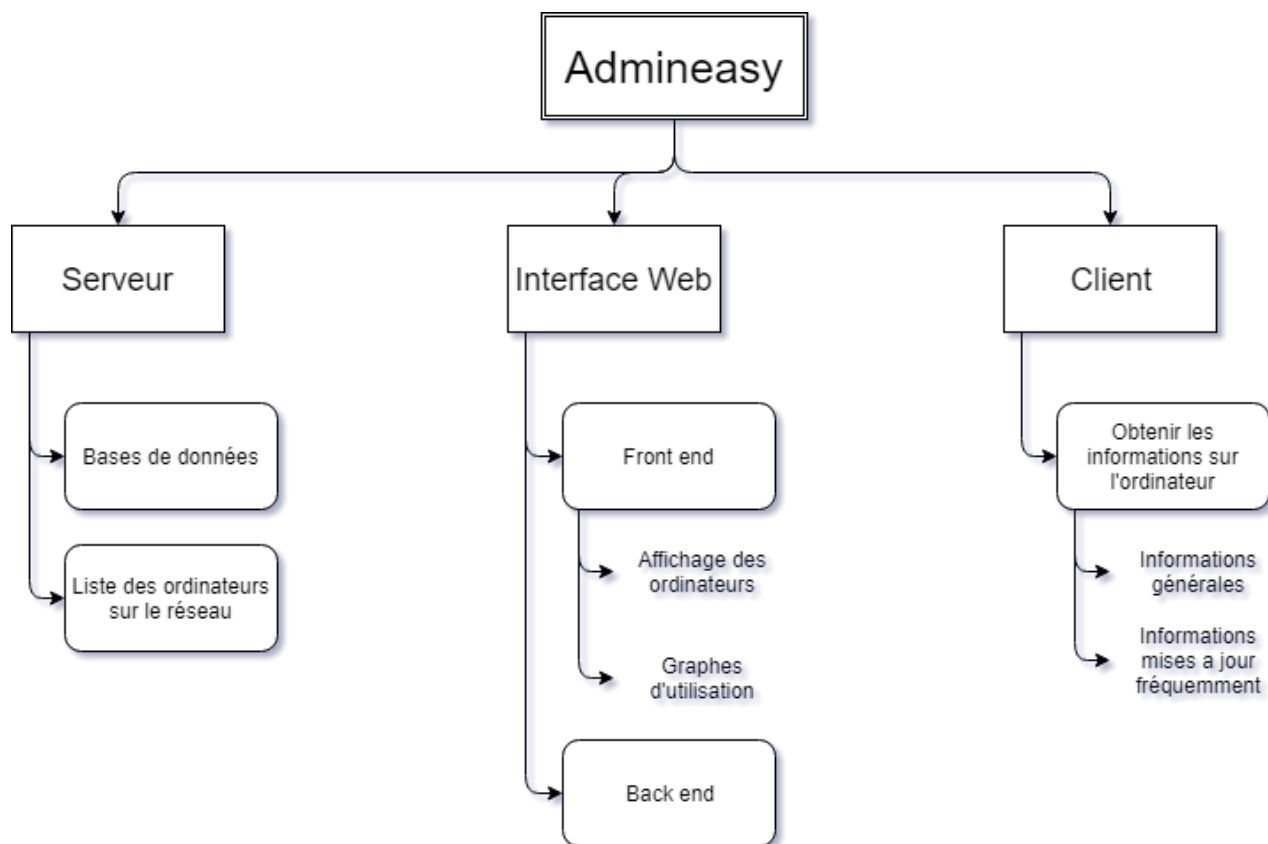
Etant donné qu'une vue d'ensemble ne comporte pas toutes les informations disponibles mais seulement les plus pertinentes, il a alors fallu déterminer quelles étaient les informations essentielles. La priorité était le contrôle du parc informatique et le maintien de son bon état. Si une erreur est détectée, la personne en charge du parc informatique en est visuellement notifiée via le site internet et peut alors intervenir. Une erreur est détectée grâce à l'analyse des caractéristiques variables de chaque machine (utilisation du processeur, sa température, etc). Également la liste des machines présentes dans le parc informatiques ainsi que les caractéristiques stables (nom, ip, etc) de chacune de ces machines est un des objectifs de notre application.

L'élément permettant l'accès à toutes ces informations est la base de données. Elle permet de stocker les informations que l'on souhaite exploiter. Ainsi, au vu de la quantité d'informations devant être stockées dans la base de données, il était nécessaire de séparer leurs différentes natures. D'un côté les informations stables et de l'autre les informations instables, qui changent au cours du temps.

## Organisation

### Définition de la procédure

Découpage en lot



Ces différents lots se retrouvent dans l'arborescence du code source.

### Github

Le VPN et les bases de données communes nous ont permis de travailler ensemble sur les parties serveur et client. Ce partage nous a garanti que les applications restaient compatibles.

Github nous a permis de partager nos développements. Nous avons aussi utilisé le système de release et le wiki. Ces outils nous ont permis de collaborer plus efficacement.



# Répartition

## Mélissa et Chloé

Mélissa et Chloé se sont occupées de la partie interface Web. Leur qualités et leurs affinités avec le web leur ont permis de mettre en place le site de surveillance d'Admineasy.

## Antoine

Antoine s'est occupé du client, c'est-à-dire la capture de l'ensemble des informations sur les machines, Linux et Windows, et leur transmission vers la base de donnée. Il a choisi de le développer en Python car il appréciait ce langage et il souhaitait approfondir ses connaissances des librairies correspondantes. Il a aussi mis en place le processus pour obtenir un exécutable, installer le programme et l'exécuter en tant que service.

Antoine avait mis en place un système de surveillance des ressources système dans son serveur avant de commencer le projet. Il avait utilisé telegraf pour obtenir des données dans une base InfluxDB, affichées sur Grafana. Il a utilisé ces connaissances pour choisir les bases de données, et mettre en place une page Grafana pour vérifier l'exactitude des données stockées dans la base de données InfluxDB.

Antoine a installé les bases de données sur un serveur virtuel hébergé chez Scaleway, et a mis en place un serveur OpenVPN pour pouvoir se connecter directement aux bases de données.

## Kylian

Kylian s'est occupé de la gestion des serveurs ainsi que de la création du site officiel d'Admineasy.

Son intérêt et ses connaissances relatives aux serveurs de traitement web ont fait de lui le responsable de ces services. De plus, il a pu mettre en place, en collaboration avec Antoine pour les accès au VPN un accès audit VPN permanent sur une Raspberry qui héberge également les services web et Node.js.

Enfin, il est chargé de la communication autour du projet, notamment avec la mise en place du site officiel d'Admineasy.

# Bases de données

## PostgreSQL

Nous avons choisi PostgreSQL parce que c'est une base de données très populaire et réputée très puissante que nous n'avions jamais utilisé.

Il y a deux tables sur cette base de données :

### machines

Stocke les informations qui changent rarement sur les ordinateurs où le client est installé.

| machines                      |
|-------------------------------|
| + name: string                |
| + os_complete: string         |
| + os_simple: string           |
| + os_version: string          |
| + user_name: string           |
| + connection_time: timestamp  |
| + cpu_name: string            |
| + cpu_cores: int              |
| + cpu_threads: int            |
| + cpu_hyperthreading: boolean |
| + cpu_freqmin: int            |
| + cpu_freqmax: int            |
| + ram_total: int              |
| + swap_total: int             |
| + local_ip: string            |
| + net_ifaces: string          |
| + disk_names: string          |

| connected         |
|-------------------|
| + date: timestamp |
| + ip: string      |

### connected

stocke les adresses IP de tous les ordinateurs du réseau et l'heure où ils ont été détectés.

# InfluxDB

InfluxDB est une base de données de séries temporelle : Elle est optimisée pour stocker et récupérer des données indexées sur le temps. Cela veut dire

Nous y stockons les informations sur l'utilisation des ordinateurs qui changent fréquemment, pour pouvoir faire des graphes d'utilisation et trouver des anomalies. Par exemple, une montée de température rapide ou une utilisation CPU trop élevée.

Chaque mesure contient, en plus des valeurs ci-dessous, le nom de la machine qui a produit cette mesure.

| cpu         |
|-------------|
| + freq: int |
| + used: int |

| ram                    |
|------------------------|
| + total_bytes: int     |
| + used_bytes: int      |
| + available_bytes: int |
| + used_percent: int    |

| swap                   |
|------------------------|
| + total_bytes: int     |
| + used_bytes: int      |
| + available_bytes: int |
| + used_percent: int    |
| + output_bytes: int    |
| + input_bytes: int     |

| network                 |
|-------------------------|
| + interface: string     |
| + sent_bytes: int       |
| + sent_packets: int     |
| + received_bytes: int   |
| + received_packets: int |
| + out_error: int        |
| + out_drop: int         |
| + in_error: int         |
| + in_drop: int          |

| disk_io            |
|--------------------|
| + disk: string     |
| + read_bytes: int  |
| + read_count: int  |
| + read_time: int   |
| + write_bytes: int |
| + write_count: int |
| + write_time: int  |

| partition           |
|---------------------|
| + partition: string |
| + total: int        |
| + used: int         |
| + available: int    |
| + used_percent: int |

| temp             |
|------------------|
| + device: string |
| + average: int   |
| + highest: int   |
| + lowest: int    |
| + high: int      |
| + critical: int  |

| fan              |
|------------------|
| + rpm: int       |
| + device: string |

| battery               |
|-----------------------|
| + is_plugged: boolean |
| + percent_left: int   |
| + seconds_left: int   |

# Client

## Présentation

Le client ("Harvester") tourne sur toutes les machines dans un réseau local. Il envoie les informations sur les ressources et leur utilisation aux bases de données, qui sont hébergées par un serveur situé sur le réseau local. Pendant le développement, le serveur était un serveur virtuel hébergé chez Scaleway. Nous avons utilisé un VPN pour simuler un réseau local afin de pouvoir tous accéder aux mêmes données.

Le client est programmé en Python. Il n'a pas de contraintes de performances importantes, donc utiliser un langage plus efficace n'aurait pas apporté de bénéfice important. Il peut vérifier sa propre utilisation de ressources système et utilise généralement moins de 0.7% de CPU et 30MB de RAM.

Le client utilise beaucoup de libraires. Ces librairies fournissent, pour chaque système, l'ensemble des informations souhaitées. Pour faciliter l'installation et ne pas devoir forcer l'installation de Python et des librairies, nous avons utilisé Pyinstaller pour créer un exécutable. Il inclut un environnement d'exécution et toutes les librairies. Il peut être lancé par n'importe quelle machine qui utilise le même système d'exploitation que celui qui a été utilisé pour construire l'exécutable.

Les étapes permettant de mettre en place un environnement de développement pour le client sont détaillées dans les annexes.

Le code source est disponible sur Github: <https://github.com/antoine-42/admineasy/tree/master/client>

# Fonctionnement

Le client est installé en tant que service sur Linux et Windows. Cela permet au programme d'être lancé dès que l'ordinateur est allumé et ne dépend pas de la connexion d'un utilisateur. Il est aussi invisible pour l'utilisateur de l'ordinateur s'il n'a pas de droits d'administration.

Quand le client est lancé, il commence par essayer de se connecter aux bases PostgreSQL et InfluxDB. S'il est lancé directement par l'utilisateur et l'option de débogage est activé, il peut le lancer en mode hors-ligne. Sinon, le client va mettre à jour les données dans la base PostgreSQL, puis mettre à jour les données dans la base InfluxDB toutes les 5 secondes.

Le fichier de configuration est situé dans: [répertoire d'installation]\harvester\harvester\settings.json. Dans ce fichier, on peut changer les informations de connexion aux bases de données, et activer l'option de débogage.

## Interface

Nous avons implémenté une interface en ligne de commande. Elle peut être utilisée par l'administrateur système s'il lance le programme directement en activant l'option de débogage. Cette interface donne toutes les informations que le programme récupère:

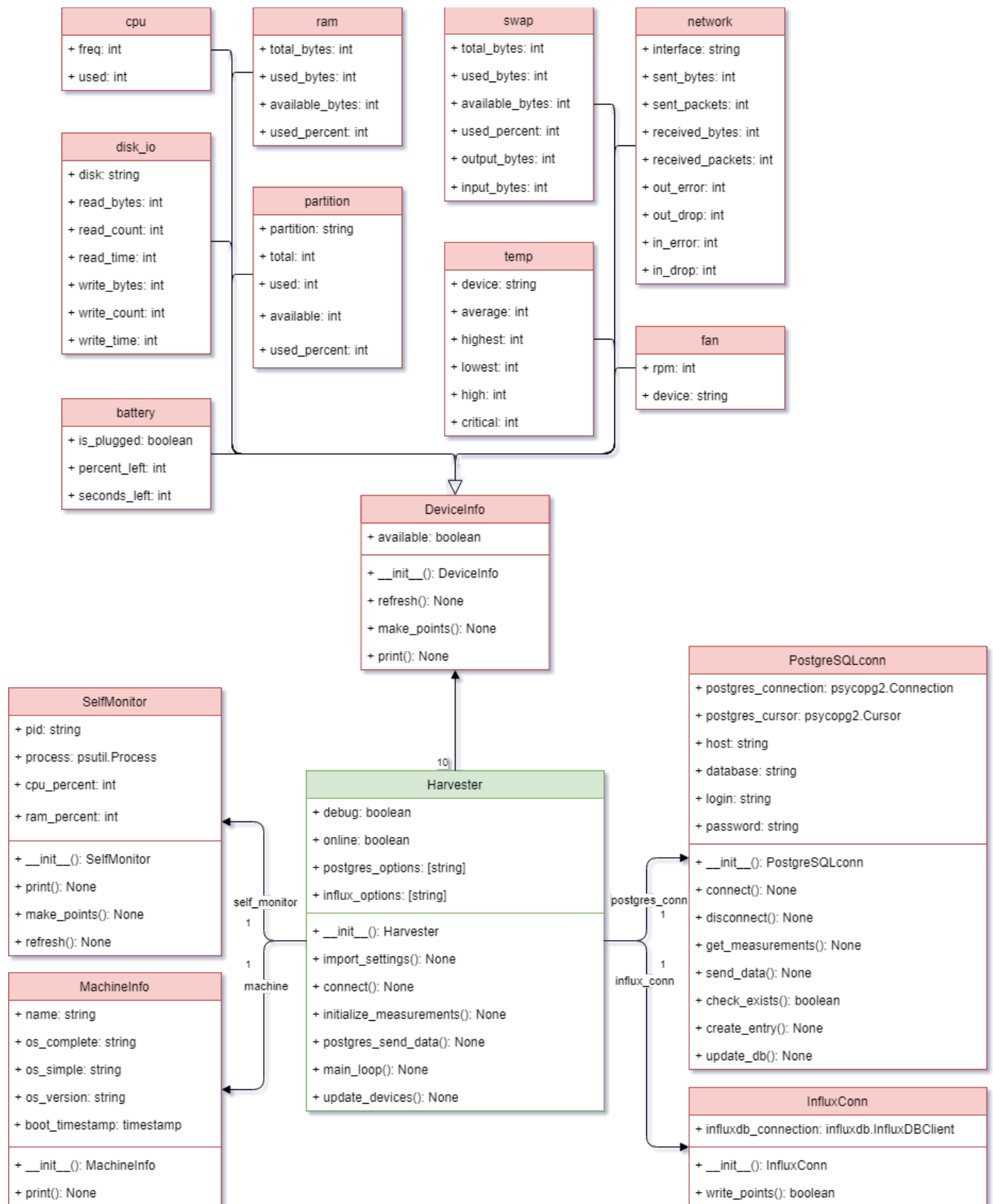
```
Update 1 starting at 2018-02-24T19:34:24.093638

User: Antoin  connected at: 2018-02-23T23:38:25
CPU: Intel(R) Core(TM) i7-5820K CPU @ 3.30GHz  Cores: 6  Hyperthreading: True  Usage: 11.9 %  Frequency: 3300.0 mhz
RAM: 17.1 GB  Used: 12.2 GB (71.7 %)
SWAP: 22.4 GB  Used: 19.1 GB (85.3 %)
Interface: Ethernet  received: 2284.5 MB  sent: 153.9 MB  errors: 0
Interface: Ethernet 2  received: 2284.5 MB  sent: 153.9 MB  errors: 0
Interface: Loopback Pseudo-Interface 1  received: 2284.5 MB  sent: 153.9 MB  errors: 0
Interface: Local Area Connection* 10  received: 2284.5 MB  sent: 153.9 MB  errors: 0
Disk: C:\  mount: C:\  file system: NTFS  total: 249.5 GB  used: 234.4 GB
Disk: D:\  mount: D:\  file system: NTFS  total: 3000.5 GB  used: 2028.5 GB
Disk: PhysicalDrive0  total read: 8.1 GB  total written: 7.5 GB
Disk: PhysicalDrive1  total read: 3.5 GB  total written: 2.9 GB
Disk: PhysicalDrive2  total read: 0.0 GB  total written: 0.0 GB
Disk: PhysicalDrive3  total read: 0.0 GB  total written: 0.0 GB
Disk: PhysicalDrive4  total read: 0.0 GB  total written: 0.0 GB
Disk: PhysicalDrive5  total read: 0.0 GB  total written: 0.0 GB
Disk: PhysicalDrive6  total read: 0.0 GB  total written: 0.0 GB
Self usage:  CPU: 0.5 %  RAM: 0.14 %

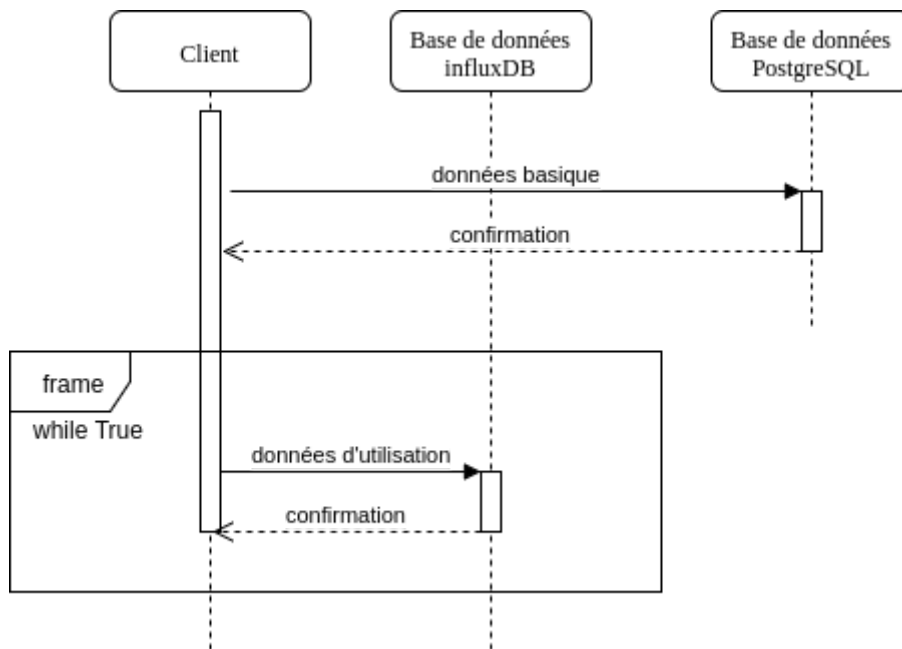
Update 1 finished at 2018-02-24T19:34:25.282279  duration: 1.2 s
```

# Diagrammes

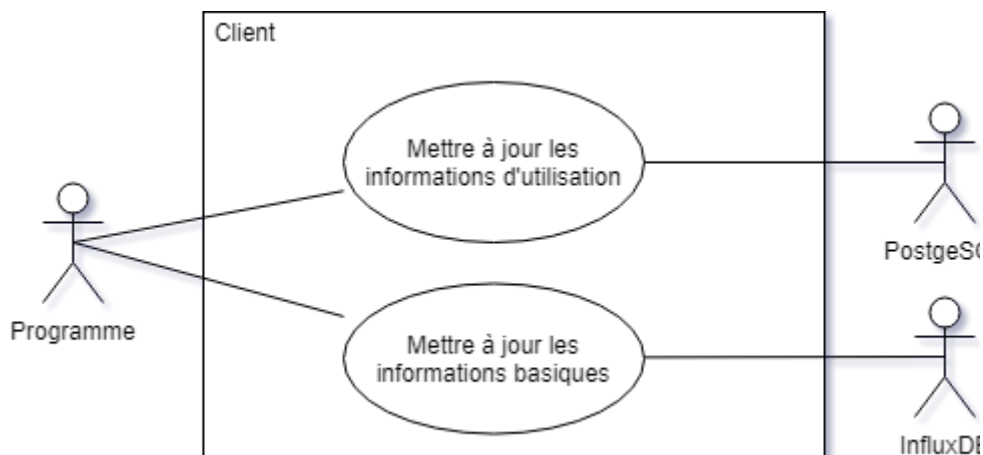
## Diagramme de classe



## Diagramme de séquence



## Diagramme de cas d'utilisation



# Logiciels externes

## Librairies

Nous avons utilisé des librairies pour faciliter la récupération d'informations et permettre la connexion avec les bases de données :

- [Psutil](#) : informations sur le matériel et son utilisation. License BSD.
- [Py-cpuinfo](#) : informations détaillées sur le modèle du processeur. License MIT.
- [Influxdb](#) : communication avec des bases de données Influxdb. License MIT.
- [Psycopg2](#) : communication avec des bases de données PostgreSQL. Licence LGPL.

Nous utilisons aussi les libraires platform, os, et socket pour obtenir des informations sur la machine.

Nous avons choisi d'utiliser Psutil et cpuinfo au lieu de chercher où trouver les informations nous même car ça n'aurait pas été très intéressant. Il aurait été inutile de dupliquer les outils performants existants. Ces librairies obtiennent les informations en analysant les résultats de commandes ou en cherchant dans le registre.

## Autres logiciels

- [PyInstaller](#) pour faire un exécutable standalone.
- [Makeself](#) pour faire un installeur sur Linux.
- [NSIS](#) pour faire un installeur sur Windows.
- [NSSM](#) pour installer le programme comme un service sur windows. License MIT.

Nous avons aussi utilisé [Grafana](#) pour vérifier facilement l'état de nos bases de données.

## Aspect légal

Nous devons inclure dans le code final toutes les librairies que nous avons utilisé, ainsi que NSSM pour Windows, afin que ces logiciels soient automatiquement installés. Cette opération est autorisée par leur licences.



## Informations récupérées

Si aucune librairie n'est mentionnée, toutes les données sont récupérées par psutil.

|  |   |
|--|---|
| Machine                                  | Platform : nom de la machine, système d'exploitation.<br>Psutil : date et heure de boot.  |
| Utilisateurs                             | Utilisateurs connectés, date et heure de leur connection.   |
| CPU                                      | Py-cpuinfo: modèle de processeur.<br>Psutil: nombre de cœurs, hyper threading, fréquence actuelle, maximale, et minimale, utilisation.                                  |
| RAM                                      | Espace total, utilisé et disponible en octets et espace utilisé en pourcentage.   |
| SWAP                                     | Espace total, utilisé, et disponible en octets et espace utilisé en pourcentage. Octets écrits et lus.  |
| Interfaces réseau                        | Nom de toutes les interfaces réseau, nombre d'octets et de paquets qu'elles ont envoyés, et les erreurs.  |
| Partitions                               | Nom et point de montage de toutes les partitions, leur espace total, utilisé, et disponible en octets et espace utilisé en pourcentage.                                 |
| Lectures et écritures sur disque         | Nom de tous les disques, et leurs nombre de lectures et écritures, d'octets lus et écrits, et temps passé à lire et écrire.   |
| Température                              | Température actuelle et maximale de tous les capteurs de température.<br>N'est pas disponible sur certains ordinateurs.   |
| Ventilateurs                             | Vitesse de rotation de tous les ventilateurs.<br>N'est pas disponible sur certains ordinateurs.   |
| Batterie                                 | Si l'ordinateur est connecté à une batterie (portable, onduleur), pourcentage et autonomie restantes sur la batterie.<br>N'est pas disponible sur certains ordinateurs. |
| Utilisation des ressources par le client | Psutil : Pourcentage de CPU et RAM utilisé par le programme.  |

# Installation

L'installateur copie les fichiers et installe le client en tant que service pour qu'il soit lancé à chaque démarrage. Les étapes pour créer cet installateur sont détaillées dans les annexes.

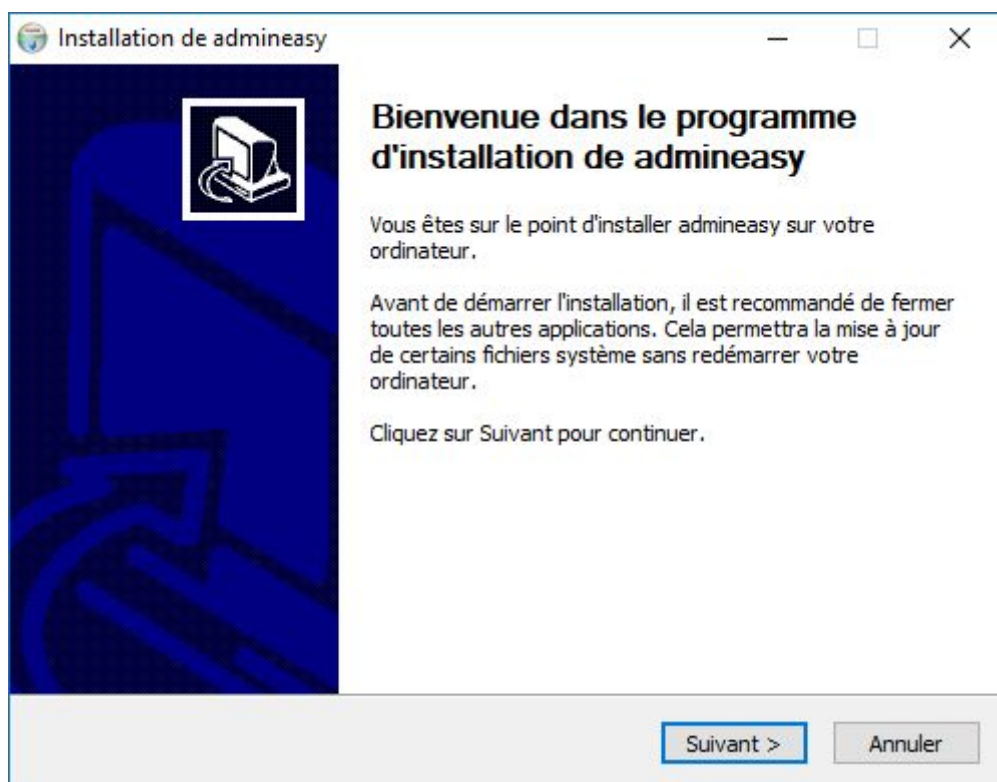
## Linux

Télécharger [harvester-setup.run](#).

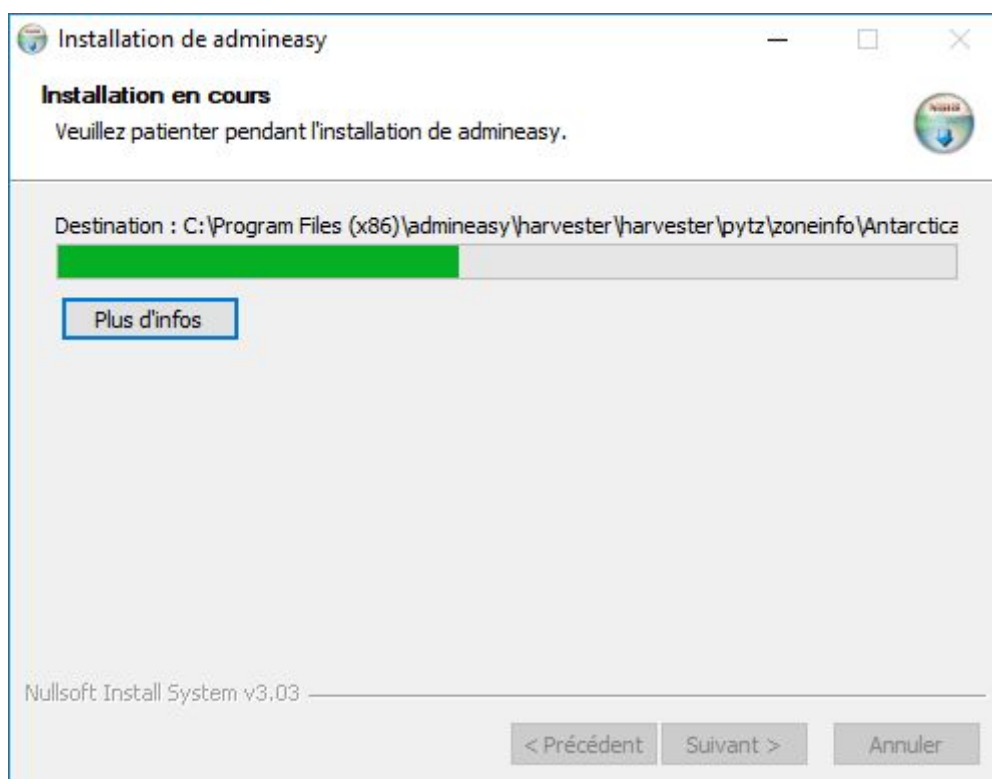
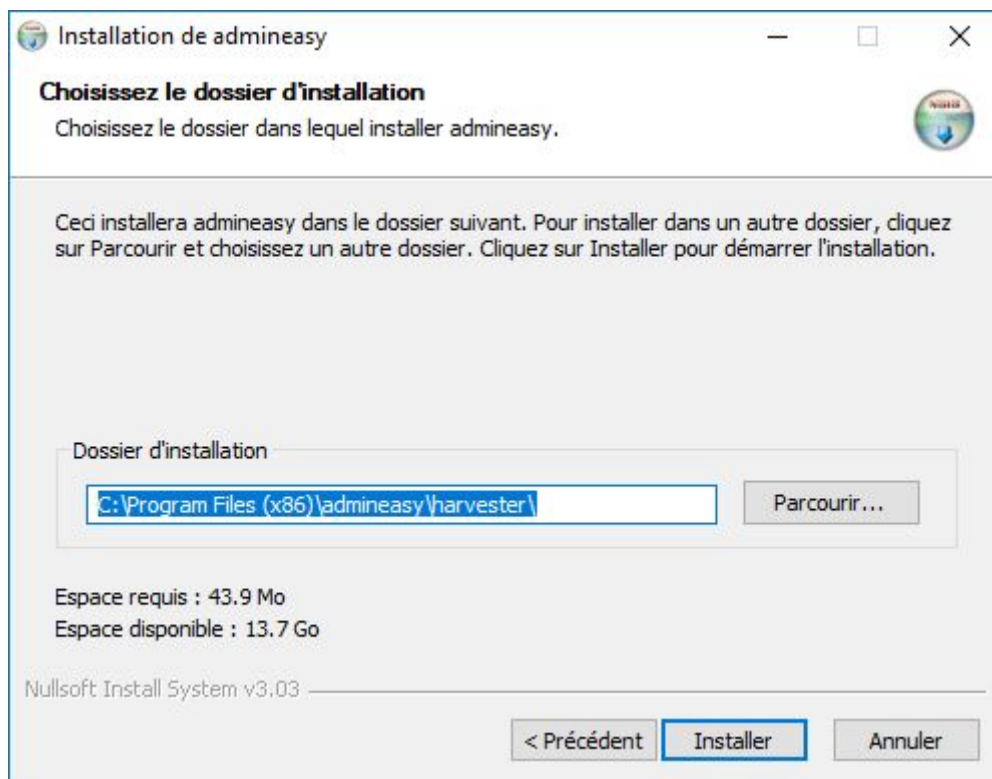
```
sudo ./harvester-setup.run --target /opt/admineasy
```

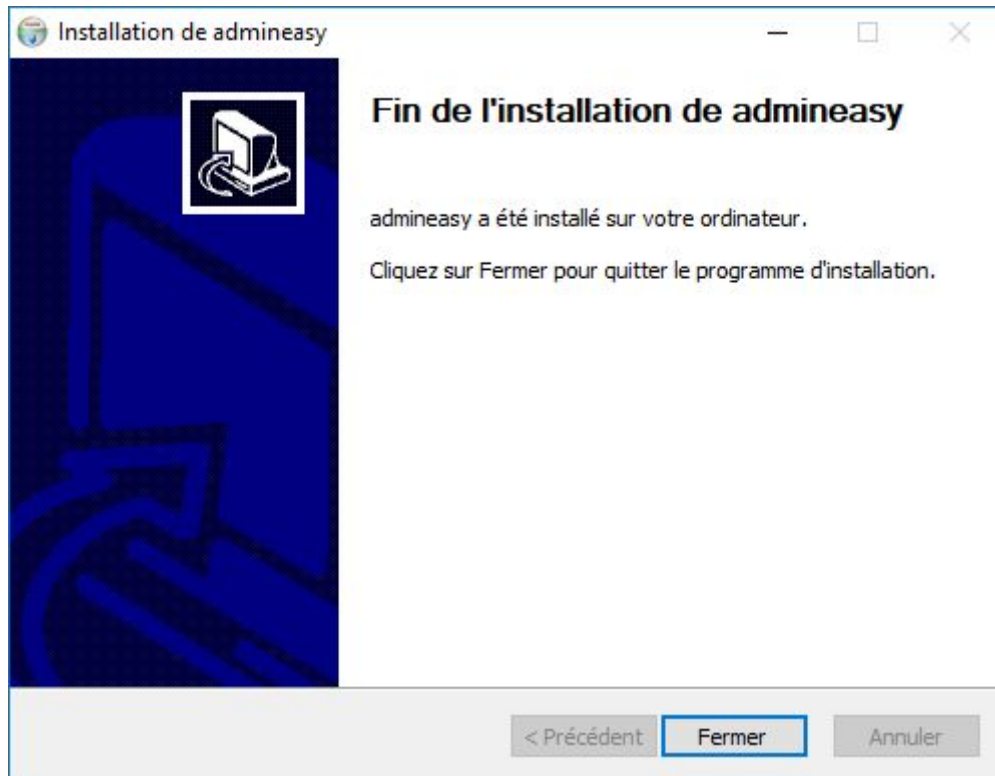
## Windows

Télécharger [admineasy.exe](#) et le lancer en tant qu'administrateur. Il peut être nécessaire de désactiver l'antivirus.



L'installateur permet à l'utilisateur de changer le répertoire d'installation :





## Explication du fonctionnement

### Linux

L'installateur, construit par Makeself, décompresse les fichiers dans le dossier choisi par l'utilisateur. Il lance ensuite harvester-setup/harvester-setup.

Ce script copie harvester-setup/admineasy-harvester.service dans /etc/systemd/system puis active et lance le service. le service sera ensuite lancé à chaque démarrage.

### Windows

L'installateur fait par NSIS décompresse les fichiers dans le dossier choisi par l'utilisateur (par défaut C:\Program Files (x86)\admineasy).

Il lance ensuite harvester-setup/harvester-setup.exe. Ce script utilise NSSM pour installer harvester/harvester.exe en tant que service. le service sera ensuite lancé à chaque démarrage.

# Désinstallation

## Linux

Sur Linux, il faut lancer le script :

```
[répertoire d'installation]harvester/harvester-uninstaller/harvester-uninstaller
```

Il va retirer le service. Il faudra ensuite supprimer les fichiers du programme.

## Windows

Sur Windows, il suffit de lancer Uninstall.exe dans le répertoire d'installation. Ce programme va supprimer tous les fichiers et retirer le service.

## Serveur

Le serveur est le lien entre les différents éléments du projet. En effet, il s'agit de l'élément qui permet d'accéder aux bases de données permettant ainsi à l'interface Web d'afficher les données souhaitées, une fois récupérées par les clients.

En réalité, Admineasy est représenté par trois serveurs :

- Le serveur "Scanner commandé" qui permet de recenser les machines présentes sur le réseau à la demande de l'utilisateur (via le site web)
- Le serveur "Serveur de connexion Base de données" qui effectue le lien avec les bases de données
- Le serveur "Scanner rémanent" qui peut être lancé dans le but de conserver les machines connectées lors du lancement. A l'avenir, il pourrait effectuer des scan automatiques de façon périodique

## Les différents serveurs et leurs fonctions

### Scanner rémanent

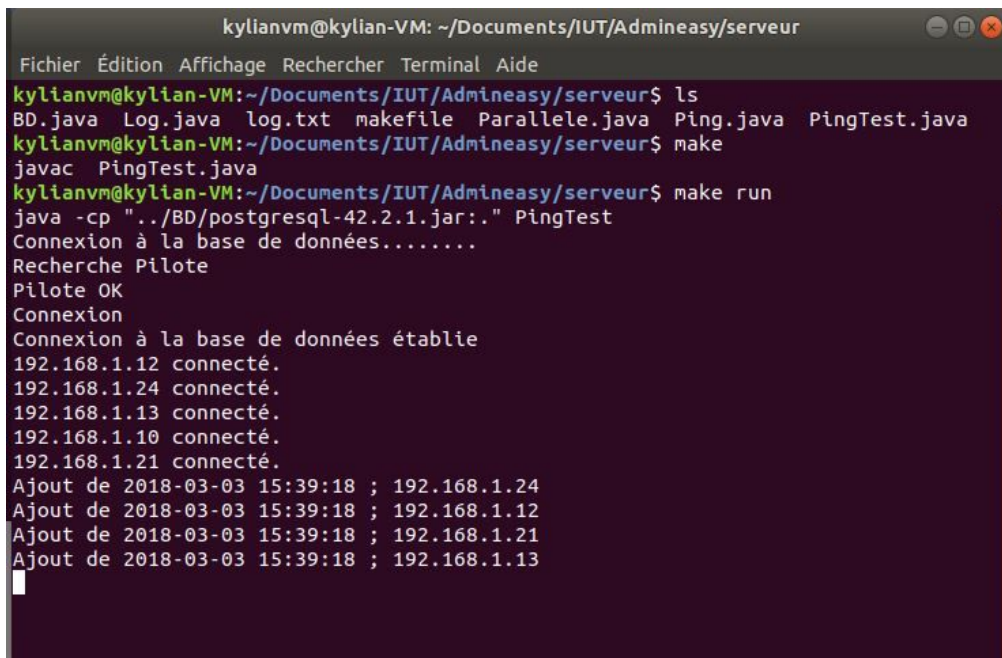
Le but premier du serveur était de trouver toutes les machines présentes sur le réseau. Pour effectuer cette opération, le C semblait être un langage efficace. La première version (voir annexe) permettait donc de récupérer les adresses IP des machines recherchées. Cependant, la gestion des données récupérées et le transfert des données vers la base de données étaient peu pratiques. Ainsi, le Java s'est imposé. Il était facile de faire une requête de ping (toujours multi threadée) et d'ajouter les pilotes de connexion à la base de données.

Ce serveur est donc opérationnel et le déploiement suivi de l'utilisation est simple :

On configure un fichier java, indiquant les identifiants de connexion à la base de données, et la plage de recherche.

```
1 public class PingTest
2 {
3     public static void main(String[] args)
4     {
5         Log.newLogBlock() ;
6         //Nom de la base, nom d'utilisateur, mot de passe, table
7         BD postgres = new BD("admineasy", "admineasy_client", "1337", "connected") ;
8         Ping.setBD(postgres) ;
9
10        Ping.pingRange34(192, 168, 1, 1, 0, 255) ;
11        //Ping.pingRange34(172, 16, 2, 2, 100, 300) ;
12        postgres.disconnection() ;
13    }
14 }
```

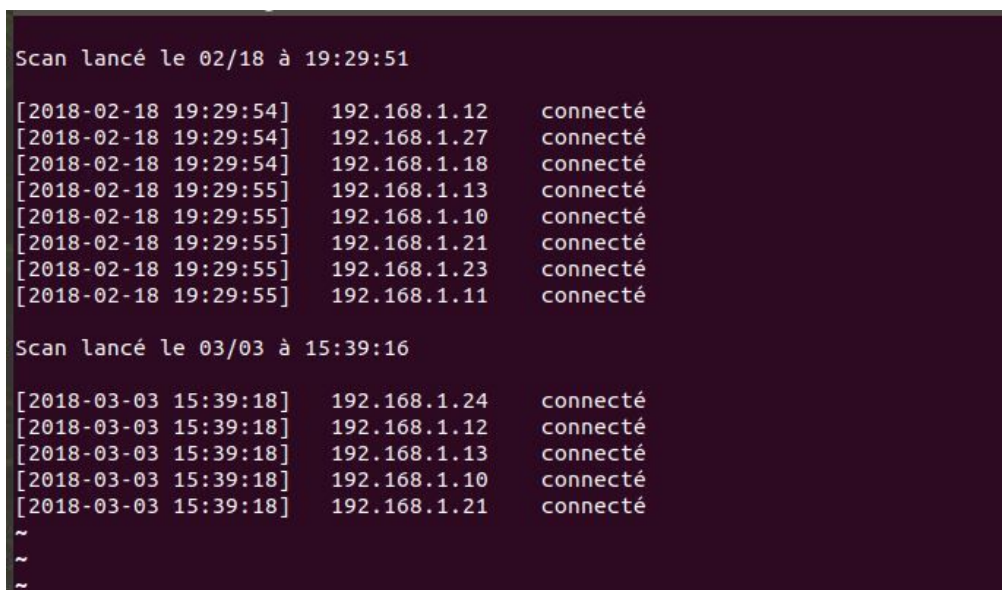
Une fois ce fichier paramétré, on compile avec “make” et on exécute le serveur, comme on peut le voir sur l’illustration suivante :



```
kylianvm@kylian-VM: ~/Documents/IUT/Admineasy/serveur
Fichier Édition Affichage Rechercher Terminal Aide
kylianvm@kylian-VM:~/Documents/IUT/Admineasy/serveur$ ls
BD.java Log.java log.txt makefile Parallele.java Ping.java PingTest.java
kylianvm@kylian-VM:~/Documents/IUT/Admineasy/serveur$ make
javac PingTest.java
kylianvm@kylian-VM:~/Documents/IUT/Admineasy/serveur$ make run
java -cp "../BD/postgresql-42.2.1.jar:." PingTest
Connexion à la base de données.....
Recherche Pilote
Pilote OK
Connexion
Connexion à la base de données établie
192.168.1.12 connecté.
192.168.1.24 connecté.
192.168.1.13 connecté.
192.168.1.10 connecté.
192.168.1.21 connecté.
Ajout de 2018-03-03 15:39:18 ; 192.168.1.24
Ajout de 2018-03-03 15:39:18 ; 192.168.1.12
Ajout de 2018-03-03 15:39:18 ; 192.168.1.21
Ajout de 2018-03-03 15:39:18 ; 192.168.1.13
```

On peut donc voir les logs de mise en place de la base de données puis l’indication des machines connectées (identifiées par leur adresse IP). Ensuite, on peut voir les lignes “Ajout ...” qui correspondent à l’ajout des lignes formatées pour la base de données.

Dans le même temps, un fichier de logs est complété (log.txt) :



```
Scan lancé le 02/18 à 19:29:51

[2018-02-18 19:29:54] 192.168.1.12 connecté
[2018-02-18 19:29:54] 192.168.1.27 connecté
[2018-02-18 19:29:54] 192.168.1.18 connecté
[2018-02-18 19:29:55] 192.168.1.13 connecté
[2018-02-18 19:29:55] 192.168.1.10 connecté
[2018-02-18 19:29:55] 192.168.1.21 connecté
[2018-02-18 19:29:55] 192.168.1.23 connecté
[2018-02-18 19:29:55] 192.168.1.11 connecté

Scan lancé le 03/03 à 15:39:16

[2018-03-03 15:39:18] 192.168.1.24 connecté
[2018-03-03 15:39:18] 192.168.1.12 connecté
[2018-03-03 15:39:18] 192.168.1.13 connecté
[2018-03-03 15:39:18] 192.168.1.10 connecté
[2018-03-03 15:39:18] 192.168.1.21 connecté
~
~
~
```

Néanmoins, la connexion permanente à la base de données ainsi que l’activité constante de ce serveur qui utilise des ressources et que l’intérêt au niveau de l’interface web est limité puisque l’utilisateur ne rafraîchira pas sa page “en temps réel”. Afin d’alléger le nombre de requêtes et de permettre au site web de commander le serveur, nous avons décidé de transférer le serveur Java en une version Node.js.



## Scanner commandé

Il s'agit donc de la version Node.js du serveur Java. Celui-ci est donc appelé par l'interface web pour connaître les disponibilités de la machine. Les résultats sont alors composés sous forme d'une chaîne de caractères - une partie de code html - qui sera récupérée par l'interface web.

Pour le commander, à l'instar du serveur en Java, il faut paramétrer la plage de données dans la fonction "reaction" :

```
var reaction = function(req, res)
{
    res.setHeader('Access-Control-Allow-Origin', '*') ; //Autorise tout le monde à accéder au serveur
    var codeHtml ; //Chaîne de caractères à écrire

    var page = url.parse(req.url).pathname ; //Permet de savoir quelle page est demandée.

    if(page == '/ping')
    {
        var arguments = querystring.parse(url.parse(req.url).query) ; //On récupère et on segmente les arguments dans un tableau
        if('ip' in arguments)
        {
            //...
        }
        else
        {
            ping_plage(192, 168, 1, 2, 1, 255, function(html) {
                //...
            })
        }
    }
    else
    {
        res.writeHead(404, {"Content-Type": "text/html"}) ; //...
    }
}

/*Le serveur est créé et on affiche le contenu souhaité*/
var server = http.createServer(reaction) ;
server.listen(PORT) ; //Le serveur écoute le port souhaité au début du code
console.log("Serveur lancé (port : "+PORT+")") ;
```

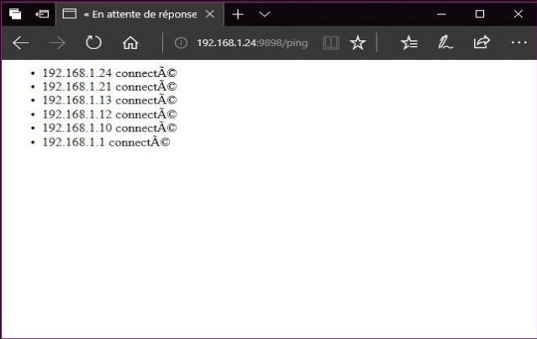
Le paramétrage permet de s'adapter à la structure et à la taille du réseau, évitant des requêtes de ping à des adresses hors du sous-réseau, ce qui optimise la vitesse de traitement. Un requête de ping comme celle de l'illustration nécessite environ dix secondes pour être réalisée (selon la machine qui tient le serveur et le nombre de machines nécessitant une écriture, les tests ayant été réalisés sur un réseau domestique).

Il suffit ensuite de lancer le serveur :

```
kylianvm@kylian-VM:~/Documents/IUT/Admineasy/nodejs$ node pingPlage.js
Serveur lancé (port : 9898)
```

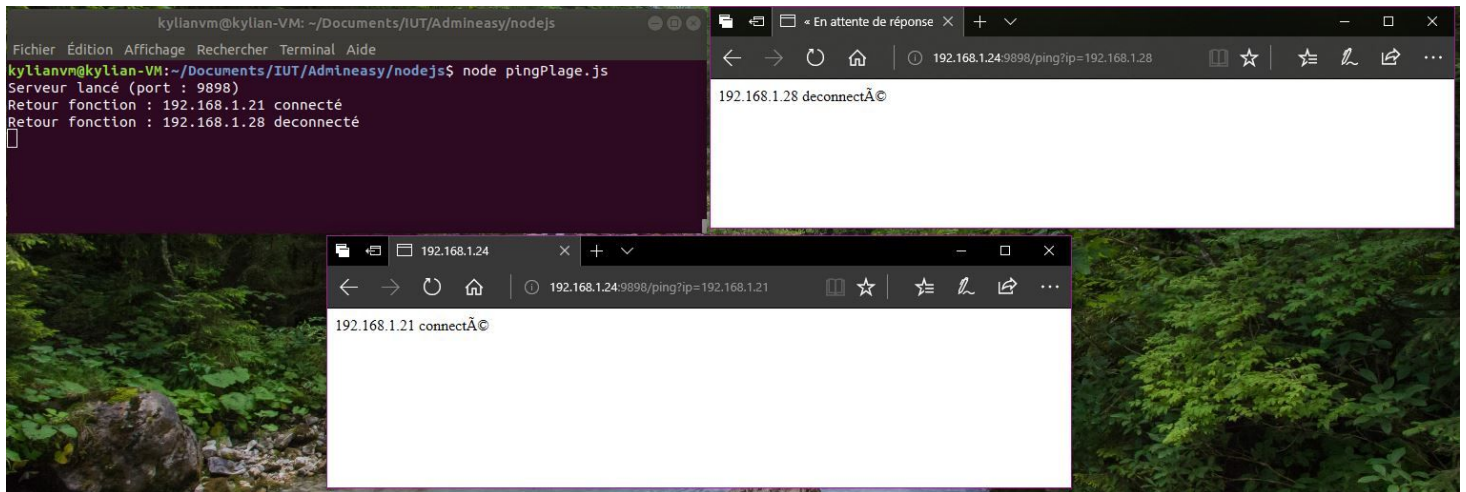
Une fois la configuration du serveur effectuée et le serveur lancé, un appel à l'url : "http://host:port/ping" exécute la fonction et en renvoie le résultat, à la fois dans le terminal et à la fois sous forme "html" (brut sur l'illustration, mais il sera récupéré et traité par le JavaScript de l'interface).

```
kylianvm@kylian-VM:~/Documents/IUT/Admineasy/nodejs$ node pingPlage.js
Serveur lancé (port : 9898)
Ping en cours
Machines testées : 510
Retour fonction : <ul><li>192.168.1.24 connecté</li><li>192.168.1.21 connecté</li><li>192.168.1.13 connecté</li><li>192.168.1.12 connecté</li><li>192.168.1.10 connecté</li><li>192.168.1.1 connecté</li></ul>
```





Il est également possible d'effectuer un test de présence unitaire avec l'url suivante :  
 "http://host:port/ping?ip=x.x.x.x "



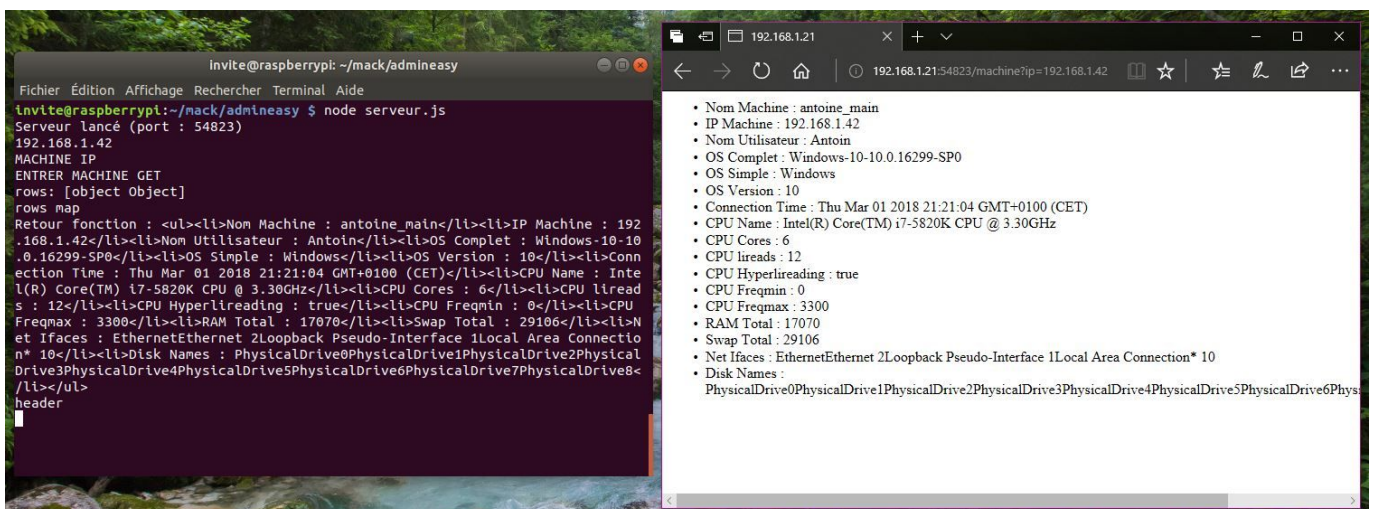
Cette version est donc facilement commandable par du Javascript et demeure performante grâce au traitement asynchrone.

## Serveur de connexion Base de données

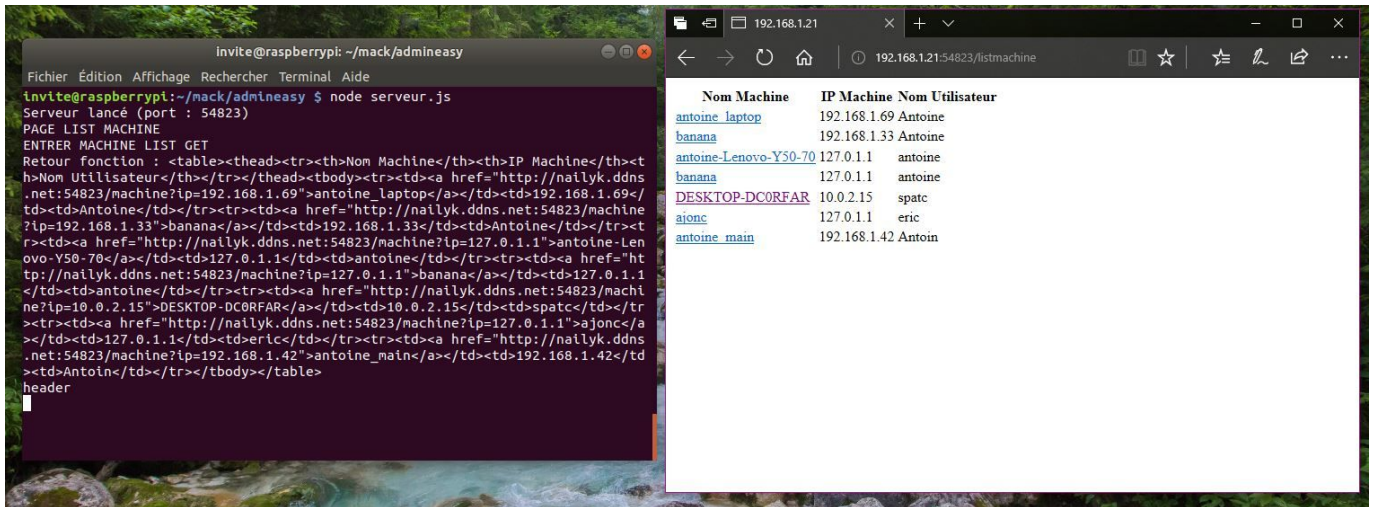
Ce serveur a pour vocation de se connecter à la base de données et d'en ressortir les informations. Il faudra paramétrer “manuellement” les identifiants de connexion à la base de données. Une fois cette opération effectuée, le serveur lancé, il suffit d’appeler ses différentes fonctionnalités pour obtenir des traitements.

Les fonctionnalités sont appelées après l'url (<http://host:port/>) et sont les suivantes :

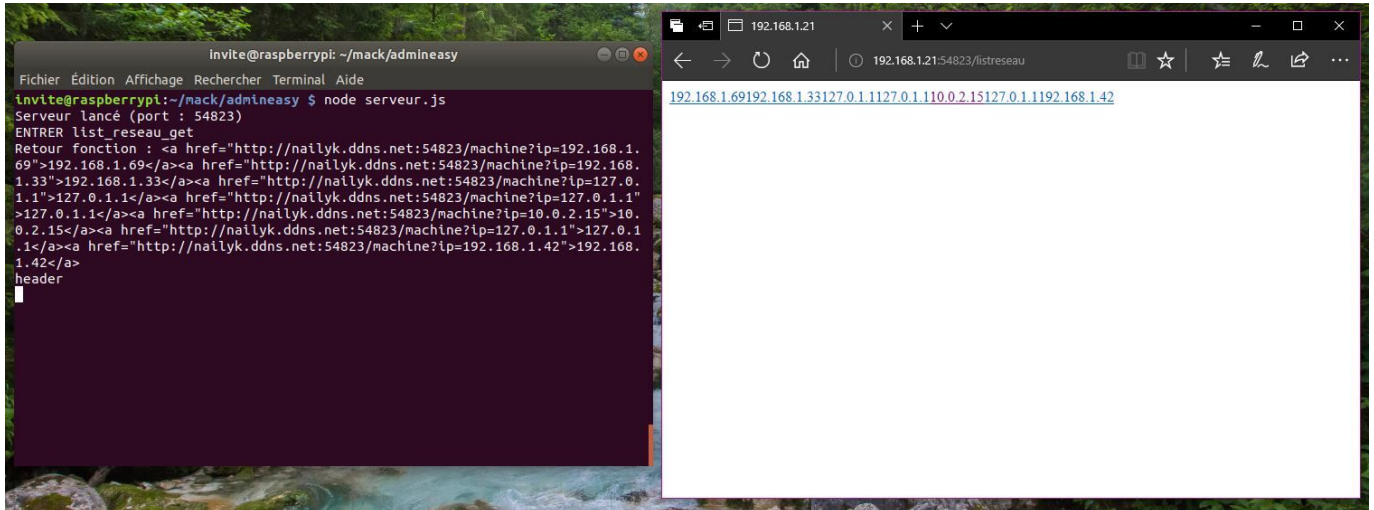
*machine* : permet d'afficher les détails d'une machine



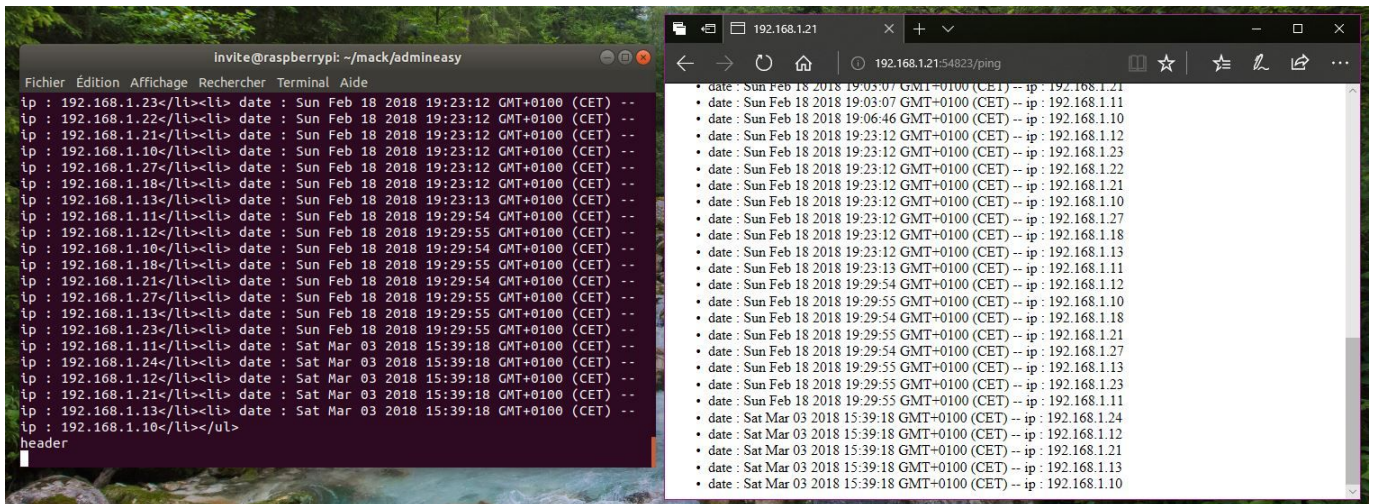
*listemachine* : permet d'afficher les informations principales de toutes les machines enregistrées sur la base de données



*listereseau* : renvoie la liste des adresses ip des machines du réseau avec un lien vers leurs détails (voir "machine")

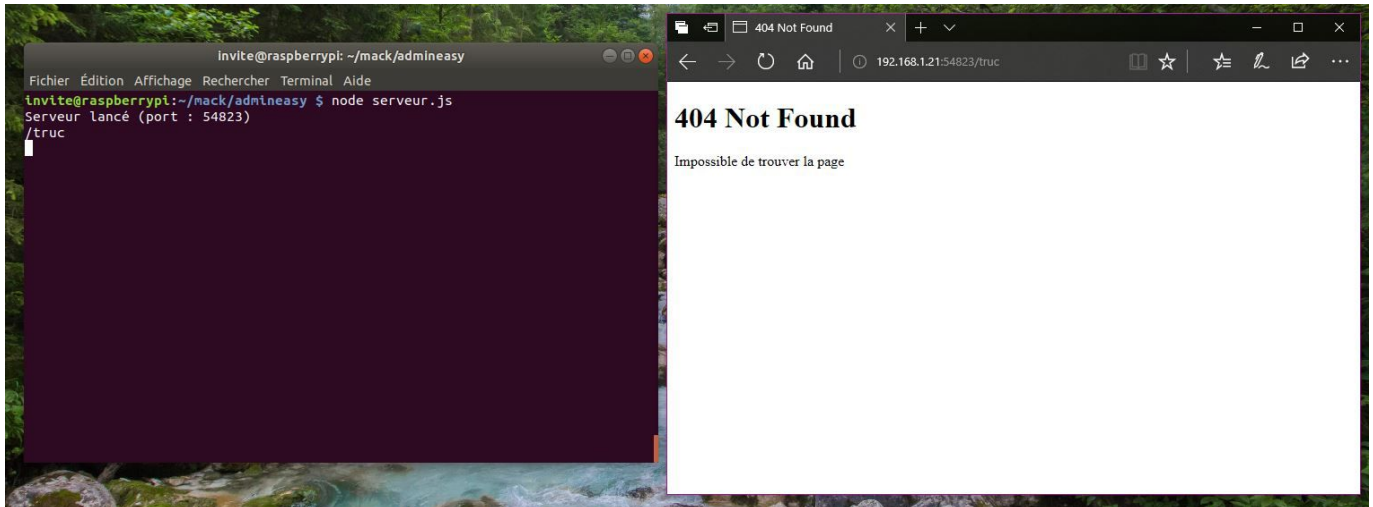


*ping* : renvoie la liste des retours de ping stockés en base de donnée





Pour tout autre commande : renvoie une erreur 404 Not Found



## Mise en place

Afin de pouvoir exécuter le serveur dans de bonnes conditions, il faudra effectuer les opérations suivantes :

1. Installation de Node.js : `sudo curl -L https://git.io/n-install | bash`
2. Après s'être placé dans le répertoire des fichiers node.js,
  - a. `npm -install ping`
  - b. `npm -install pg`
  - c. `npm -install express`
  - d. `npm -install multer`
  - e. `npm -install influx`
3. Vérifier que les ports 54823, 54824 et 54825 ne sont pas déjà utilisés. Dans le cas contraire, il suffit de changer la valeur de "PORT" au début de chaque fichier
4. Si l'installation web est située hors du réseau surveillé par les serveurs, configurer le routeur afin d'autoriser les entrées/sorties des ports
5. Télécharger les fichiers dans l'espace téléchargement (FTP) du site officiel <http://nailyk.ddns.net/admineasy/download/serveurs>

## Interface web

L'interface web fait le lien entre tous les éléments de notre application : serveur, client et bases de données.

Son objectif est d'être simple d'utilisation et de regrouper toutes les informations nécessaires à la surveillance d'un parc informatique.

L'utilisateur doit avoir accès en temps réel aux informations que contiennent les bases de données. Les informations principales doivent être accessibles en un coup d'oeil et les informations complémentaires, bien que dissimulées doivent être accessibles rapidement.

Cette interface prenant la forme d'un site web, l'*HTML* et le *CSS* ont donc été les deux langages privilégiés pour sa création. Toutefois, ceux-ci ne permettent pas la réalisation d'un site web dynamique, ni même la connexion à une base de données. Il a donc fallu utiliser le *Javascript* et par extension *Ajax*. Ce dernier permet de ne pas recharger automatiquement la page pour mettre à jour les données.

Le code source est disponible sur Github: <https://github.com/antoine-42/admineasy/tree/Site>

## Style et critères ( d'une Interface Homme Machine)

L'interface de l'application est simple et épurée. Les éléments qui construisent la page sont gris et les éléments mis en valeur sont bleus, ces couleurs ont été choisies car elles sont sobres et professionnelles. Ces couleurs permettent de se repérer facilement sur le site et d'identifier les éléments essentiels. Cette identification est facilitée par l'homogénéité des différentes pages. En effet, chacune des pages possède la même structure:

Le titre du site :

The logo for AdminEasy, featuring the word 'Admin' in a dark blue serif font and 'Easy' in a lighter blue serif font, both in a large, bold, slightly stylized typeface.

Il est mis en évidence en haut de chaque page. Dessous celui-ci on retrouve la barre de navigation.

Le menu du site:



Lorsque l'utilisateur survole un intitulé du menu, celui-ci se colore en bleu. L'onglet du menu qui correspond à la page où se situe l'utilisateur est aussi coloré ce qui lui permet de se repérer facilement dans les différentes pages du menu. De plus, cette barre de navigation, toujours présente, autorise l'utilisateur à changer de page à n'importe quel moment ainsi que de revenir à une page où il était précédemment, ce qui respecte le critère de pilotage.

De même, chaque intitulé du menu est significatif et indique clairement à l'utilisateur sa fonctionnalité. Cela respecte le critère de concision.


Ensuite, chacune des pages possède un message qui annonce le contenu de page à l'utilisateur.

Par exemple la page réseau contient :

**Voici une vision d'ensemble du réseau:**

Pour finir, au bas des pages il y a le pied de page. Sa taille représente la totalité de la longueur de la page.

La partie gauche du pied de page indique le nom des auteurs de l'application.



MACK - Da Costa, Dujardin, Trugeon, Bohl

La partie centrale indique le cadre du projet.



Projet dans le cadre du DUT informatique - IUT Fontainebleau - 2017/2018

Et la partie droite est un lien qui renvoie l'utilisateur vers la page contact du site qui lui permet de remplir un formulaire afin de contacter les auteurs du site. Ceci respecte le critère d'assistance ; l'utilisateur sait vers où se diriger s'il rencontre un problème.



Contact

Le fait que l'ensemble des pages respecte la même structure permet de satisfaire le critère d'homogénéité. Le site respecte également le critère de signifiante en n'affichant que les données essentielles ainsi que des boutons ou des liens dont la fonction est clairement définie.

## Fonctionnalités

Une des première fonctionnalité du site web est la barre de navigation du site permet d'accéder à toutes les fonctionnalités en moins de 3 clics.

Notre interface web est donc composé de cinq pages : accueil, réseau, machines, utilisation et contact.

La page **accueil** indique à l'utilisateur l'état général du parc informatique.



Lorsqu'il y a une alerte ou plus, un lien permet d'accéder aux données concernant la ou les machine(s) dont le comportement est suspect.

La page **réseau** nous affiche une vision d'ensemble du réseau avec chaque machine identifiée par son adresse IP. Un témoin coloré représentant un ordinateur permet d'indiquer l'état de chaque machine :

192.168.1.69



192.168.1.33



- Vert : la machine est connectée et tout va bien
- Rouge : la machine est connectée mais des alertes sont présentes
- Gris : la machine n'est pas connectée

L'adresse IP est alors un lien permettant d'accéder aux détails de la machine.

La page **machines** liste les machines présentes dans le parc informatique. Elle permet de trouver rapidement une machine grâce à son utilisateur, son adresse IP ou son nom. Pour obtenir des informations supplémentaires, il suffit de cliquer sur le nom de la machine.

| Nom Machine                           | IP Machine   | Nom Utilisateur |
|---------------------------------------|--------------|-----------------|
| <a href="#">antoine laptop</a>        | 192.168.1.69 | Antoine         |
| <a href="#">banana</a>                | 192.168.1.33 | Antoine         |
| <a href="#">antoine-Lenovo-Y50-70</a> | 127.0.1.1    | antoine         |
| <a href="#">banana</a>                | 127.0.1.1    | antoine         |

Toutefois, si l'administrateur souhaite accéder rapidement à une machine particulière, il lui suffira de rentrer l'adresse IP de cette machine dans le champs de recherche. Il aura alors accès aux caractéristiques de la machine.

Rechercher une machine via son adresse ip

Si une adresse IP entrée n'existe pas, le site en informe l'utilisateur :

**IP: 192.2 est inexistante**

La dernière fonctionnalité de la page machine est qu'elle permet de faire un ping (commande permettant de tester l'accessibilité d'une autre machine à travers un réseau IP).

**Faire un ping sur toutes les machines**

Après avoir cliqué on retrouve une liste représentant les différents ping qu'il y a eu:

- 192.168.1.21 connecté
- 192.168.1.13 connecté
- 192.168.1.12 connecté
- 192.168.1.10 connecté
- 192.168.1.1 connecté

La page **utilisation** permet de voir l'utilisation des machines du parc informatique. Les pages précédentes utilisaient des informations statiques (utilisation de la base de données Postgres) tandis que cette page donne des informations dynamiques (base de données Influx).

On retrouve sur cette page différentes informations comme l'utilisation du processus, sa température ou encore l'utilisation mémoire, conformément aux données récupérées par le client.

La dernière page du site web est la page **contact**, celle -ci contient un formulaire de contact (raccourcis ici) :

## Formulaire de contact

Prénom :

Ex : Jane

Nom :

Ex : Doe

Il demande à l'utilisateur plusieurs informations, comme son prénom, nom, adresse mail.

Puis le message envoyé est sous la forme d'un mail donc l'objet du mail est demandé et un champ de texte permet à l'utilisateur de saisir son message. L'utilisateur a la possibilité d'effacer son message en cours d'écriture et ensuite de l'envoyer grâce à deux boutons situé à la fin du formulaire :

Envoyer

Effacer



## Lien avec le serveur

C'est l'utilisation du langage *Node.js* associé au *Javascript* et à *Ajax* qui nous a permis de réaliser le lien avec le serveur. Prenons l'exemple de la recherche d'une machine avec son adresse IP :

Tout d'abord, la fonction principale écrite en *Javascript* utilisant la particularité d'*Ajax* (qui permet de ne pas recharger la page lorsqu'il y a de nouvelles données) est celle-ci :

```
searchIP: function(){ // recherche la machine avec l'ip qu'on lui donne
    console.log("searchIP");
    var requete= creerRequete();
    var arg=document.getElementById("ip").value;
    var url="http://nailyk.ddns.net:54823/machine?ip="+arg ;
    requete.open("POST", url, true);
    requete.onreadystatechange=function(){
        Methode.afficherResultat(requete, arg, "IP");
    }

    requete.send();
},
```

La première étape met en place l'environnement qui nous permet de communiquer avec le serveur. La communication se fait sous forme de requêtes HTTP. Ensuite, la demande est soumise vers le serveur qui récupère l'IP qu'on lui a donnée. Le serveur, écrit en *Node.js* se connecte à la base de données Postgres:

```
var conString = "postgres://admineasy_client:1337@10.8.0.1:5432/admineasy" ;
// connection a la BD
```

Il soumet la requête à la base de données afin de récupérer les informations souhaitées.

```
var query = "select * from machines where local_ip='"+ip+"'";
// recupere les machines qui ont l'ip demande
```

La réponse du serveur est de l'*HTML*, dans notre cas c'est une suite de liste qui énumère toutes les caractéristiques de la machine dont on aura donnée l'IP si bien sûr cette machine existe. Sinon ce sera un texte indiquant que la machine n'existe pas (comme décrit dans la page **machines**).

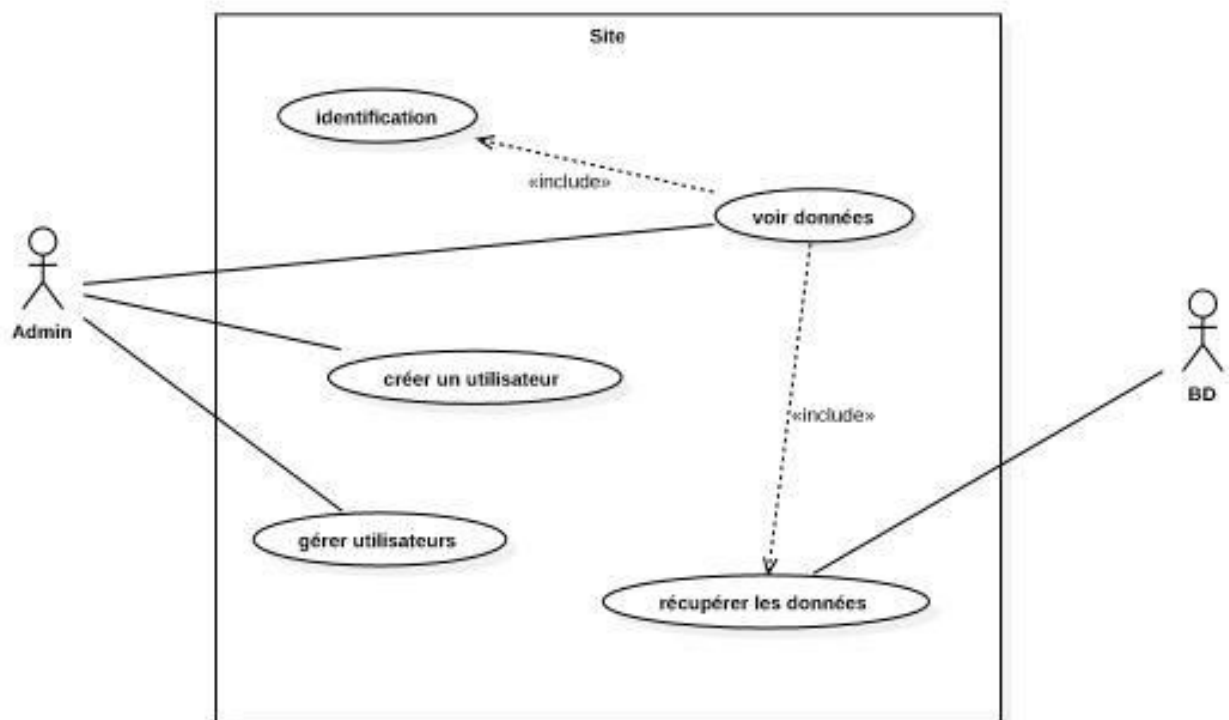
```
callback('<ul>'+code+'</ul>');
// renvoie la reponse du serveur
```

Enfin, le résultat est affiché en ajoutant la réponse du serveur directement à suite de la page web.

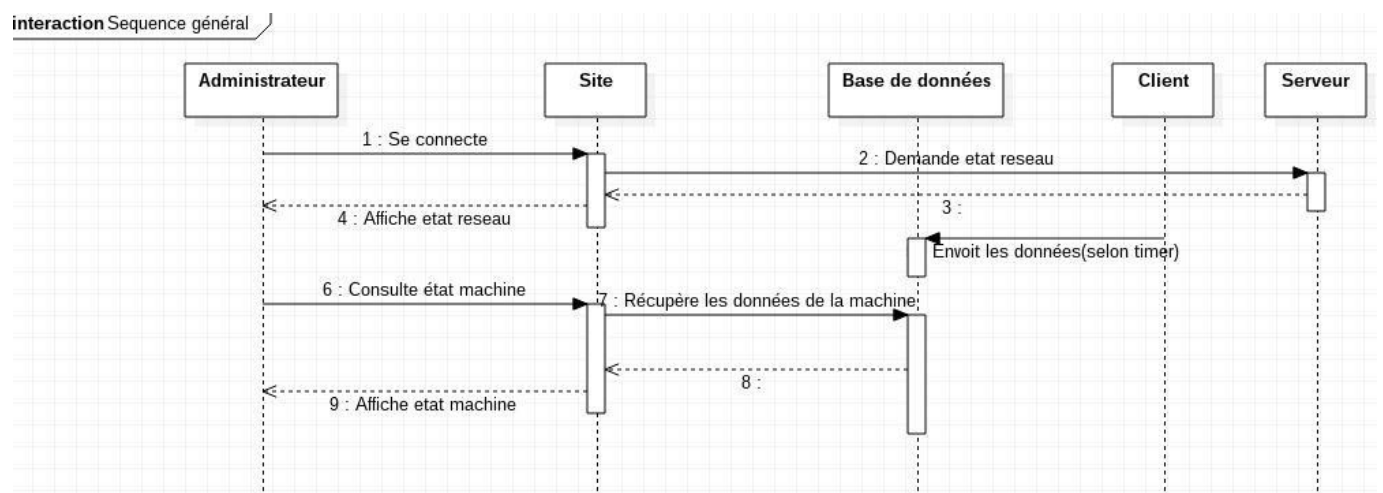
```
}else{ // si est appelé par searchIP
    elem.innerHTML=requete.responseText;
```

# Diagrammes UML

## Diagramme de cas d'utilisations

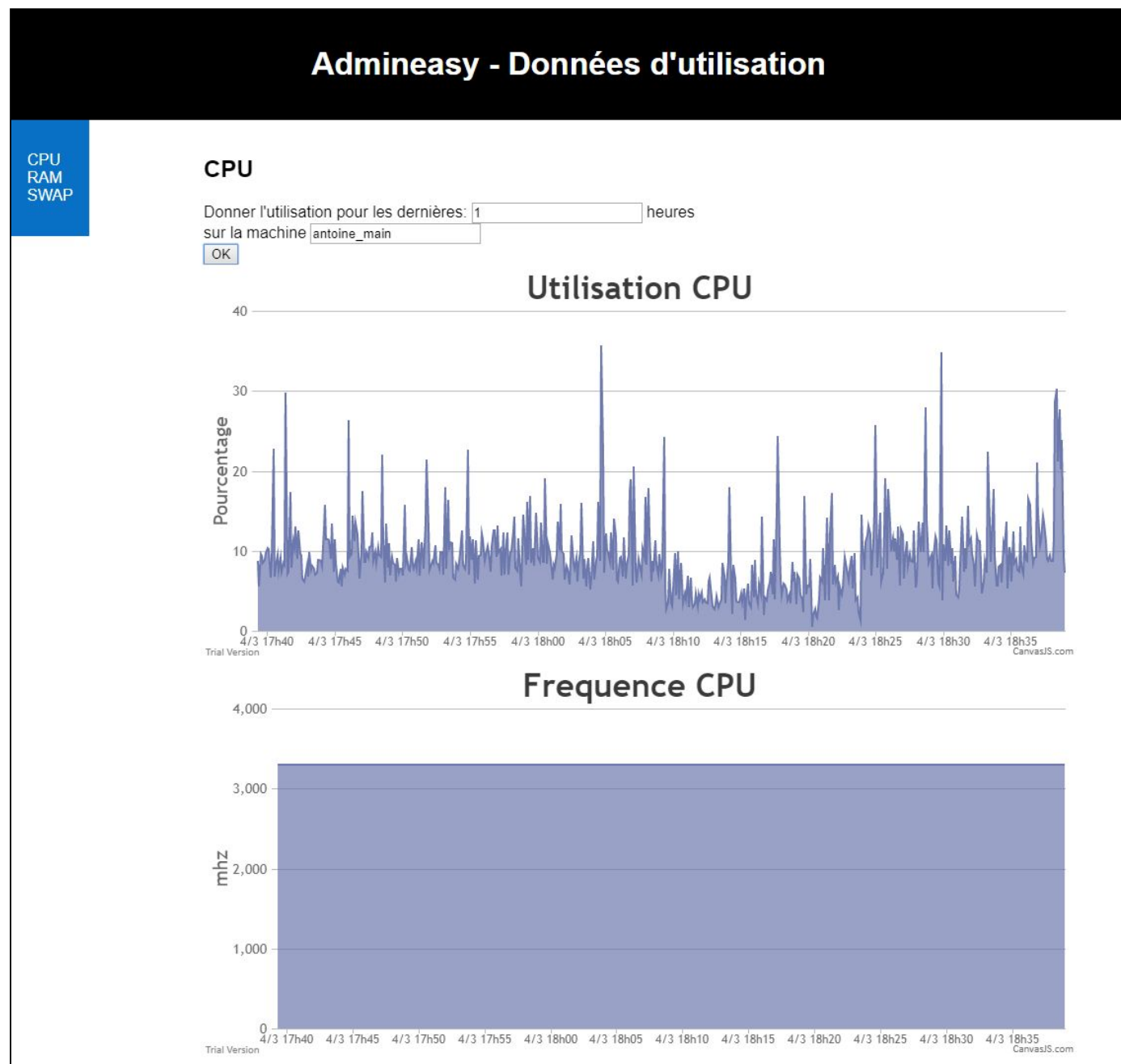


## Diagramme de séquence



# Connection a InfluxDB

L'accès à la base de données Influx nécessitait des traitements supplémentaires notamment en raison du rafraîchissement constant des données. La version actuelle ne permet donc d'afficher que les informations d'utilisation et de fréquence du CPU, de la RAM, du SWAP. Ces graphes sont mis à jour en



temps réel avec de l'AJAX.

L'utilisateur peut afficher différentes données en utilisant le menu de gauche. Pour obtenir les graphes, il doit donner la période et le nom de la machine qui l'intéresse.

Nous avons utilisé la librairie CanvasJS pour afficher les graphes.

## Améliorations possibles

### Architecture trois tiers

Pour l'instant, notre programme utilise les permissions système pour garder secrètes les informations de connection aux bases de données. Cette solution n'est pas appropriée car il serait aisé pour un utilisateur mal intentionné de les obtenir. Il faudrait mettre en place une étape de validation qui s'assure que chaque requête est bien envoyée par un client légitime. Par exemple, [ssl](#).

### Contact

Il serait possible d'ajouter une partie graphique au client qui permettrait à l'utilisateur de soumettre des rapports d'erreurs. Ces rapports d'erreurs seraient affichés sur la page web.

# Graphes d'utilisation

Toutes les données nécessaires pour obtenir des graphes d'utilisation sont présents dans la base InfluxDB. Nous pourrions donc afficher des graphes d'utilisation des ressources, comme sur cette page Grafana qui a été utilisée pour vérifier l'exactitude des données obtenues par le client :



## Le site officiel

L'objectif de ce projet était aussi de proposer une application fonctionnelle et utilisable. Ainsi, nous avons mis en place un petit site "officiel".

Il s'agit de quelques pages web qui proposent un rapide descriptif de l'application et de ses fonctionnalités. Chaque page permet de détailler un peu plus le fonctionnement des différents composants d'Admineasy. Bien entendu, ce site web contient une page de téléchargement (via l'interface FTP d'Apache) qui permet de récupérer les fichiers nécessaires à l'installation.

Il est également possible d'y trouver les indications d'installation.

[Accueil](#) [Admineasy](#) [Interface](#) [Client](#) [Serveur](#) [Base de données](#) [Documents](#) [Téléchargement](#)

### Admineasy

Bienvenue sur le site internet d'Admineasy. Vous trouverez ici les documents et les informations relatives à Admineasy.

#### Etat Actuel

Perfectionnement  
Suivez l'évolution avec le diagramme de Gantt, [ici](#)

#### Le principe

Admineasy a pour but d'aider les administrateurs réseau à lister tous les ordinateurs présents sur un réseau et connaître leur état. Il pourrait être utilisé dans tous types d'entreprise, pour une administration légère.

#### Le site Web, ou interface

Le site Web est développé par Méliissa D et Chloé T.

Il s'agit de l'interface de l'application. Elle permet à l'administrateur réseau (ou systèmes) de voir l'état des machines, et de sélectionner les machines pour lesquelles il souhaite disposer de plus amples informations.

#### Le client

Le Client est développé par Antoine D.

Le Client est placé sur chaque ordinateur à monitorer.  
Au lancement, il enverra les caractéristiques de l'ordinateur à la [base de données](#). Puis, il enverra à intervalles réguliers les informations relatives à l'utilisation et à l'état du système.  
Notre Client sera disponible pour Linux et pour Windows et est disponible [ici](#) (lien inactif pour le moment)

#### Les serveurs

Les serveurs sont développés par Kylian B.

## Difficultés rencontrées

La gestion des serveurs a quant à elle été compliquée en raison de l'apprentissage d'un langage asynchrone. Il fallait de ce fait gérer les fonctions de retour afin d'obtenir le résultat escompté.

De plus, il a fallu mettre en place l'environnement de développement sur une Raspberry (dans cas des services web) mais aussi gérer le VPN afin d'accéder aux bases de données. Ces aspects techniques ont été la raison de nombreuses recherches sur Internet mais qui se soldent par l'acquisition de nouvelles connaissances.

Dans la réalisation de l'interface web, le lien avec les bases de données a été compliqué. L'idée première était d'utiliser le langage Python, à l'aide du framework Django, adapté au développement web. Cependant, Django s'est révélé trop complexe à maîtriser pour l'utilisation que nous voulions en faire. La mise en place d'une connexion en PHP s'est également trouvée insatisfaisante. La dernière décision a été d'utiliser l'outil Node.js qui, complémentaire au Javascript, permet d'avoir accès à un environnement côté serveur et de bénéficier de sa caractéristique de programmation asynchrone.



## Conclusion du groupe

Pour ce projet, nous avons fait un système de surveillance de parc de machines. Nous avons défini son fonctionnement général, ce qui nous a permis de répartir les activités. Les techniques à mettre en oeuvre étaient très variées : des fonctions systèmes pour instrumenter les machines, des techniques de bases de données et le web. Nous avons pu intégrer ces fonctions pour présenter les résultats de l'instrumentation.

Ce projet a été une bonne expérience de gestion de projet. Nous avons suivi essentiellement un cycle en V, c'est-à-dire que l'intégration a été faite pendant la phase finale. L'exception a été le développement conjoint du serveur et du client en utilisant la base de données commune. Il aurait sans doute été préférable d'effectuer des intégrations intermédiaires avec la partie web.

Pour la réalisation, nous avons dû gérer des niveaux d'expérience hétérogènes en développement web. Toutefois, l'objectif de ce projet était de découvrir des outils que nous ne connaissions ou ne maîtrisions pas avant. C'est également le cas en entreprise, où les collaborateurs doivent se former en permanence. Tout en apprenant, il faut aussi continuer d'avancer sur le projet, ce qui n'est pas toujours évident.

A chaque étape, nous avons trouvé de nombreuses solutions techniques disponibles. A chaque fois, il faut faire un choix et se familiariser avec la solution choisie. Cela demande du temps et une bonne compréhension de la technique. Il faut de la persévérance lorsque les résultats obtenus ne sont pas satisfaisants, choses que nous avons tous apprises.



## Conclusions personnelles

### Mélissa

La réalisation de ce projet m'a permis de développer mes connaissances dans la programmation web. J'ai d'abord renforcé mes notions de Javascript, indispensable au dynamisme des sites web. L'utilisation du Javascript m'a aussi permis d'apprendre Ajax qui est indispensable à la plupart des applications web.

Enfin, j'ai appris à me servir de Node.js et particulièrement à exploiter ce qu'un serveur écrit en Node.js renvoyait.

### Chloé

Avec ce projet, j'ai d'abord pu réinvestir mes connaissances en HTML et en CSS pour l'aspect de l'interface web. L'aspect dynamique de cette dernière m'a permis d'approfondir mes quelques notions Javascript. La connexion aux bases de données s'est révélée assez difficile, mais j'ai ainsi pu appréhender l'utilisation de l'outil Node.js.

De plus, le projet tuteuré m'a permis de remettre en question mon organisation personnelle pour m'adapter à celle d'une équipe, la méthodologie et la façon de coder de chacun n'étant pas la même.

### Antoine

J'ai acquis de nombreuses connaissances techniques au cours de ce projet. J'ai approfondi mes connaissances de Python, et j'ai appris à utiliser de nombreuses bibliothèques. J'ai appris à mettre en place un serveur OpenVPN et une base PostgreSQL.

Mes connaissances de Python et Grafana m'ont aidé à obtenir mon stage chez EDF, où je vais réaliser une application de visualisation de données temporelles en Python.

### Kylian

Pour ma part, le projet a été un moyen de découvrir la gestion d'un projet complet, avec la coordination avec chaque "pôle". J'ai pu apprendre le Node.js qui est un langage serveur puissant et aux nombreuses possibilités. De plus, il m'a permis de découvrir le fonctionnement asynchrone et de le maîtriser.

Le fait de devoir mettre en place le VPN d'Antoine a été instructif sur le fonctionnement de ces réseaux virtuels. La gestion de la Raspberry afin de permettre le développement des services en ligne m'a également beaucoup appris en termes de gestion de droits (administration Linux) et de redirections nat/pat.

Il manquait néanmoins une partie de préparation du projet plus poussée, qui aurait pu faciliter le choix des langages.

Finalement, malgré certains obstacles, ce projet s'est avéré être une expérience complète et sera une source d'inspiration pour mes prochains projets.

## Termes techniques

VPN : Virtual Private Network, ou Réseau privé virtuel. Permet de créer un lien direct entre des ordinateurs distant pour qu'ils puissent communiquer comme s'ils étaient sur le même réseau local.

Swap : espace d'échange. Partie de la mémoire de masse (disque dur ou SSD par exemple) utilisé pour stocker des données qui, du point de vue des applications, se trouvent en mémoire vive. La mémoire vive est plus rapide, mais est de plus faible capacité.

Asynchrone : Un langage asynchrone est un langage dit "non-bloquant". Il ne s'arrête pas lors d'une attente d'entrée/sortie, mais continue l'exécution du code. Cela permet, sur une page web par exemple, d'afficher différentes parties une à une, dès qu'elles sont traitées, sans avoir à attendre qu'elles soient toutes traitées pour être affichées en une fois.

## Licenses

MIT, BSD : Licenses proches avec très peu de restrictions.

LGPL : si on modifie un programme qui utilise cette license, on sera obligé d'utiliser une license similaire pour le programme modifié. On peut utiliser une librairie sous license LGPL dans un programme sans devoir utiliser une license similaire, contrairement a la license GPL.