



ÉCOLE NATIONALE  
D'INGÉNIEURS DE BREST



CYANIDE STUDIO

STAGE DE 4ÈME ANNÉE

---

## Conception de scripts d'administration IT

---

*Auteur :*  
Yannick GUERN

*Responsables :*  
M. Bastien SEMENE  
M. André PÉRENNOU

3 juin 2013

# Table des matières

Remerciements	2
Introduction	3
Conclusion	40
Références	41

# Remerciements

Je tiens à remercier L<sup>A</sup>T<sub>E</sub>X et les tutoriels sur internet et notamment <http://www.ukonline.be/programmation/latex/tutoriel/index.php>.

Bla bla bla bla bla.

# Introduction

Ce document est un exemple de rapport. J'espère aider des étudiants à réaliser leur rapport en L<sup>A</sup>T<sub>E</sub>X.

Écrit par Bruno Voisin (Hiko Sejûrô) et publié sur <http://blog.hikoweb.net/>.

# CyaVision

Ce Module permet de surveiller un groupe de machines. En effectuant une batterie de tests à intervalles périodiques sur plusieurs serveurs.

## Structure

L'application est composé de 3 classes ayant des tâches bien déterminés

- Config : son rôle est de récupérer et de traiter la configuration en base de données et si celle-ci est correcte, la mettre en forme avant de la transmettre à l'application principale.
- Supervisor : est l'interface publique de l'application, elle permet de lancer les tests désirés.
- Treatment : traite les résultats provenant de la classe Supervisor et en cas de problème envoie un mail soit au sysadmin soit à la personne responsable du projet posant problème.

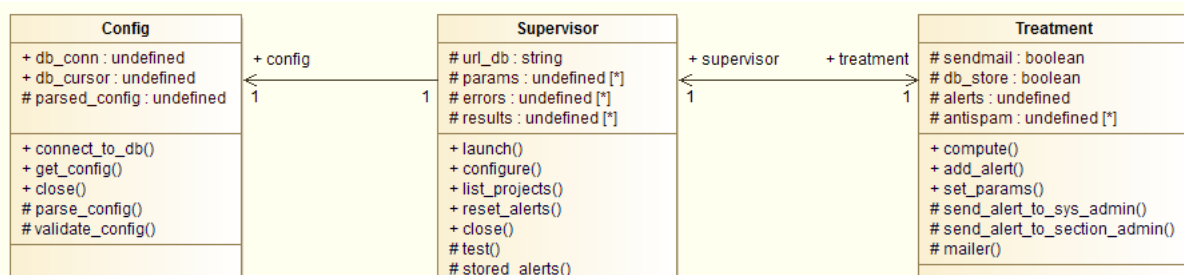


FIGURE 1 – Diagramme de classe de l'application

Ce package intègre aussi différentes classes de tests déjà existantes qui réalise les opérations sur les serveurs. Elles sont pour le moment au nombre de 3 :

- MySQL : comme son l'indique cette classe réalise des tests sur les serveurs MySQL, généralement sur port 3306.
- PyLobby : celui ci permet d'effectuer des tests sur n'importe quel port et donc n'importe quel protocole.
- Relay : permet d'opérer sur un protocole interne

### Remarque :

Pour l'instant l'intérêt de ces différentes classe n'est pas évident car la classe PyLobby peut tout à fait remplir le rôle de la classe Relay ou MySQL. Mais ceci est temporaire car dans l'avenir des tests spécifiques aux bases de données seront ajoutés, tests non compatibles avec Relay par exemple.

L'application se compose aussi de deux tables MySQL :

- config : cette table permet de configurer autant le script que les différents jeux de tests.
- alerts : celle-ci stocke l'ensemble des alertes sur les projets ayant eu des erreurs.

## Classe Config

Cette classe réalise plusieurs opération avant d'obtenir et de renvoyer une configuration correcte à la classe Supervisor, cette dernière lui transmet en paramètre de construction une URL de connexion à une BDD.

### Connexion à la BDD

La première opération consiste à vérifier le format de l'URL transmise. Pour rappel une URL standard sans paramètres s'écrit ainsi :



FIGURE 2 – Structure d'une URL selon le standard

La transformation de l'url en paramètres de connexion est réalisé via le module `urlparse`, qui permet à partir d'une chaîne de caractère de récupérer les différents éléments constituant une URL.

Pour que l'URL soit valide dans notre cas il faut que celle-ci possède un schéma égal à "mysql", sinon on retourne false.

Ensuite l'URL possède obligatoirement un nom d'utilisateur, un mot de passe et une adresse de connexion. Si l'un de ces paramètres est inexistant, l'url est considérée comme erronée et on retourne là aussi False.

Si tous les tests réussissent, on tente de se connecter, si la connexion échoue pour quelques raisons que ce soit on retourne False. Autrement on renvoie True.

### Récupération de la configuration

Si la connexion à la BDD est une réussite, on peut récupérer la configuration qui y est stockée.

La configuration se trouve dans la BDD sous la forme d'une table ayant par exemple cette forme :

section	directive	content
general	antispam_delay	3600
general	tpl_delimiter	#LINE##
general	admin_email	admin@cyanide-studio.com
general	tpl_html	<html> <head> <meta http-equiv='\"con...
general	tpl_plain	Alert(s) on \$project_name\$delimiter \$test_na...
projet1	emails	mail_projet1@cyanide-studio.com
projet1	projet1_sql	MySQL://projet1.cyanide-studio.com
projet1	projet1_prod	PyLobby://projet1.cyanide-studio.com:12026
projet1	projet1_fail	MySQL://projet1.cyanide-studio.com:1
projet2	projet2_sql	MySQL://projet2.cyanide-studio.com
projet2	emails	mail_projet2@cyanide-studio.com
projet2	projet2_prod	PyLobby://projet2.cyanide-studio.com:8016

FIGURE 3 – Contenu de la table de configuration

## Extraction de la configuration

Pour chaque ligne de la configuration on vérifie la section concerné si c'est la section "general" ou si la directive correspond à "emails" alors on ne fait que rajouter la clef "directive" est son contenu au dictionnaire de la "section".

Par exemple si la section est général et la directive "antispam\_delay" et la valeur associée est 3600 alors :

```
1 >>>print config
2 {"general":{}}
3 >>>config["general"]["antispam"]=3600
4 {
5     "general":
6     {
7         "antispam":3600
8     }
9 }
```

De même si l'on trouve une directive "emails" dans la section "projet1" avec la valeur "cyanide-studio.com" alors :

```
1 >>>print config
2 {"general":{"antispam":3600}, "projet1":{}}
3 >>>config["projet1"]["emails"]="admin_projet1@cyanide-studio.com"
4 {
5     "general":
6     {
7         "antispam":3600
8     },
9     "projet1":
10    {
11        "emails": "admin_projet1@cyanide-studio.com"
12    }
13 }
```

Bien entendu le nom des sections est généré au fur et à mesure que l'on en rencontre de nouvelles.

Si le nom de la section est différentes est différent de "general" ou que la directive n'est pas "emails" alors il s'agit de l'url d'un test il faut donc l'ajouter au dictionnaire de tests du projet.

Par exemple, le projet1 possède une url de test "test1" de valeur "scheme ://serveur.test"

Avant d'être ajouté au dictionnaire chaque url de test est vérifié pour s'assurer que celle-ci contient bien un schéma nécessaire à la suite des opérations. Si ce n'est pas le cas une erreur est revoyée.

Si c'est bien le cas on découpe l'url au niveau du séparateur " ://" et on récupère le schéma

```
1 >>>print config
2 {"projet1":{"tests":{}}}
3 >>>config["projet1"]["tests"]["test1"]=["scheme", "scheme://serveur.test"]
4 {
5     "projet1":
6     {
7         "tests":
8         {
9             "test1":["scheme","scheme://serveur.test"]
10        }
11    }
12 }
```

Si la clef "tests", n'existe pas déjà dans le projet alors elle est créée.

## Validation de la configuration

Cette configuration est ensuite validée selon plusieurs critères, il faut tout d'abord que les directives :

- `antispam_delay` : définit le temps entre deux rappels par mail des erreurs qui ont pu apparaître sur les projets.
- `tpl_delimiter` : permet de réaliser les délimitations dans les templates de mails
- `admin_email` : adresse mail à qui envoyer un mail en cas de crash du script
- `tpl_html` : template de mail au format html
- `tpl_plain` : template de mail au format plain text

En plus de devoir exister ces directives doivent aussi ne pas être vides.

Chaque projet doit aussi posséder une directive `"emails"`, afin de pouvoir contacter le ou les chefs de projets en cas d'échecs de tests sur leur serveur.



## Classe Supervisor

La méthode principale de cette classe est "launch", elle réalise de nombreuses opérations qui dans l'ordre sont de :

De vérifier que la connexion avec la BDD est correcte.

De lancer des tests spécifiques s'ils sont passés en paramètres de lancement, autrement l'ensemble des tests de la bases de données sont lancés. Et d'en générer un jeu de tests.

Si l'utilisateur à déclaré vouloir vider la table des alertes avant de lancer les tests.

Ensuite pour tous les projets possédants un jeu de tests, on réalise chacun son tour le test, si celui-ci échoue on réessaie un autre fois, chaque résultat est stocké dans une structure de données

Ayant la forme :

```
1 | test_results = [  
2 |     ["url_test1", True, "libellé du test1"],  
3 |     ["url_test2", False, "libellé du test2"],  
4 | ]
```

Le deuxième paramètre étant le résultat du test.

Si le test spécifié par l'utilisateur n'existe pas dans la base de données alors un message d'erreur le signal à l'utilisateur et l'on passe au test suivant.

## Le déroulement d'un test

La structure d'un module de test est composé de deux classes, une classe spécifique qui contient les différents jeu de test pour le protocole et une classe wrapper possédant le nom Supervise héritant de la classe spécifique, celle ci possédant elle même :

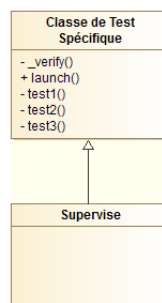


FIGURE 4 – Structure d'un module de test

- une méthode `_verify` qui permet de s'assurer de la validité des informations transmises par l'url de test.
- une méthode `launch` qui réalise le lancement des jeux de tests et renvoie le résultat de ceux-ci sous forme d'une liste de string, représentant le nom des tests ayant échoués.

Une URL peut-être composée de paramètres ce qui nous assure une souplesse totale quand aux options transmis lors des tests, toujours dans une optique de extensibilité de l'application.

Chaque module porte le même nom que le schéma de l'URL de test, il est donc aisé d'importer la bonne classe `Supervise` et de lancer la méthode "launch" de celle-ci.

Pour cela nous utilisons le module `importlib` de python. Qui permet de récupérer le module correspondant au schéma de test.

## Réinitialisation des alertes

Voici un exemple de contenu de la table d'alerts :

project	testbench	test	datetime	first_alert
projet1	projet1_prod	Ping	2014-04-09 10:50:03	2014-04-09 10:50:03
projet1	projet1_fail	Ping	2014-04-09 10:50:03	2014-04-09 10:50:03
projet2	projet2_prod	Ping	2014-04-09 10:50:03	2014-04-09 10:50:03
projet1	projet1_sql	Ping	2014-04-09 10:50:03	2014-04-09 10:50:03
projet2	projet2_sql	Ping	2014-04-09 10:50:03	2014-04-09 10:50:03

FIGURE 5 – Table d'alertes

Tout projet ayant subi un échec lors de l'un des tests se retrouve en état d'erreur et renvoie périodiquement un mail pour soit signaler qu'un problème existe soit que le problème est survenu postérieurement mais qu'il est résolu désormais.

Il faut donc un mécanisme permettant de supprimer manuellement les alertes des tests n'ayant plus d'importances.

Ceci est réalisé en ligne de commande en spécifiant quel projet nous désirons réinitialiser l'alerte.

Pour cela nous supprimons toutes les entrées de la table d'alertes possédant le projet qui doit être réinitialisés.

Si l'opération réussit la méthode retourne True. Autrement False.

## Liste des projets

Cette méthode permet de renvoyer sous la forme d'une liste représentant l'ensemble des projets stocké dans la table de configuration.

## Gestion des exceptions de script

La méthode "launch" est encapsulé dans un bloc "try and catch", ce qui permet lorsqu'une exception est levée d'attraper l'erreur puis de la transmettre à la classe de Traitement qui se chargera de traiter cette information.

## Structure de données transmise à la classe de traitement

Autant les erreurs de scripts que les échecs ou les succès sont envoyé pour traitement. Nous utilisons donc la méthode add\_result de la classe Treatment, cette méthode possède 2 paramètres :

- category : celle-ci peut soit être de type "admin" ce qui correspond à une erreur grave qui a entraîné la fin prématurée du script soit de type "project", alors il s'agit d'erreur de tests.
- datas peut soit être l'ensemble des erreurs ayant mis fin au script soit le résumé des différents tests réalisés.

```
1 | {  
2 |   "admin":["erreur de script 1", "erreur de script 2"],  
3 |   "project":  
4 |   [  
5 |     "projet1":["test11", "test12"],  
6 |     "projet2":["test22"],  
7 |     "projet3":[]  
8 |   ]  
9 | }
```

À noter si un projet ne subit pas d'erreur, alors la liste de "data" est vide.

## Classe Treatment

Son rôle est de traiter les informations en provenance de la classe Supervisor.

### Traitement des résultats

Celui-ci est réalisé dans la méthode compute qui se charge en fonction des résultats d'envoyer pour chacun des projets possédant des erreurs de tests un mail d'avertissement. Il nous faut donc un algorithme permettant de séparer les erreurs de scripts des erreurs de projets. Et de les traiter différemment en fonction de leur type.

Le voici :

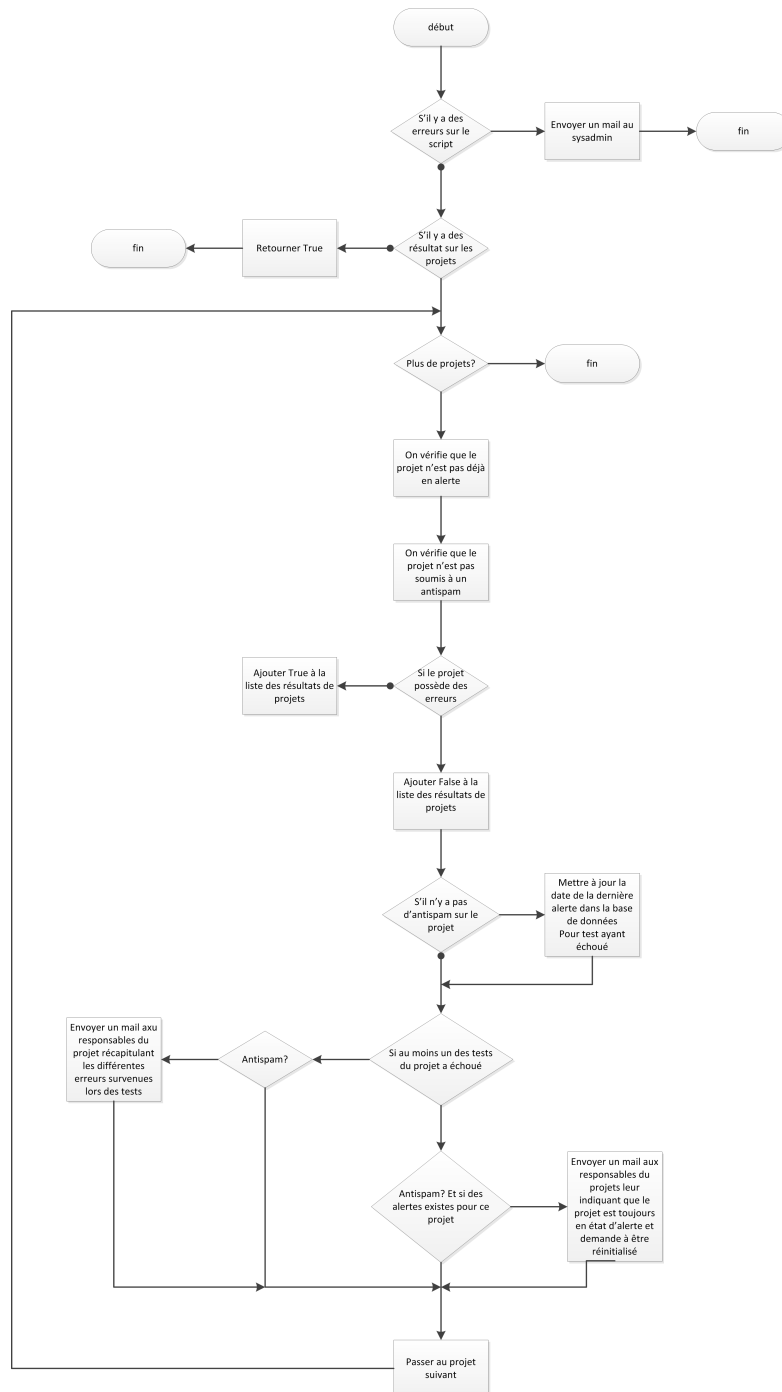


FIGURE 6 – Traitement des résultats renvoyés par Supervisor

## Envoie des mails

Les mails sont expédiés si et seulement si l'option est activée.

### Mail pour le sysadmin

Ce mail est très simple il ne s'agit que du traceback renvoyé par python. Permettant ainsi de détecter d'éventuelles erreurs de script.

### Mail pour les responsables du projet

L'envoi d'un mail aux responsables est plus compliqué car il fait intervenir une mise en forme plus élaborée.

Afin de supporter les clients mails obsolètes ou textuels, il faut pouvoir envoyer aussi bien de l'html que du plain-text.

Nous avons donc besoin de deux templates. Ceux-ci sont stockés dans la table de configuration.

Au format HTML :

```
1 <html>
2   <head>
3     <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
4   </head>
5   <body text="#000000" bgcolor="#FFFFFF">
6     Alerts on <b>$project_name</b> project:
7     <br/>
8     <ul>
9       $delimiter
10      <li>
11        $test_name
12        <br />
13        <ul>
14          $delimiter
15          <li>$routine_name <font color="#ff0000">FAILED</font></li>
16          $delimiter
17        </ul>
18      </li>
19      $delimiter
20    </ul>
21    NB : <b>first alert at <em>$first_alert</em></b>
22  </body>
23</html>
24$delimiter
25<html>
26  <head>
27    <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
28  </head>
29  <body text="#000000" bgcolor="#FFFFFF">
30    No alert on project <b>$project_name</b> but it must be <b>resetted</b> since <b>
31      ↪ $first_alert</b>
32  </body>
33</html>
```

Au format texte :

```
1 | Alert(s) on $project_name
2 | $delimiter
3 |     $test_name errors : $error_names FAILED
4 | NB : first alert on $first_alert
5 | $delimiter
6 | No alert on project $project_name but it must be resetted since $first_alert
```

La chaîne de caractères "\$delimiter" est un indicateur qui nous permettra de découper correctement le template afin de le modifier en fonction du nombre d'erreurs survenues ainsi que du type d'erreurs survenues. Et aussi permet de délimiter le template d'erreur, du template de ré-initialisation d'alertes.

Ce qui au format HTML nous donne :

Ceci pour le mail d'alerte :



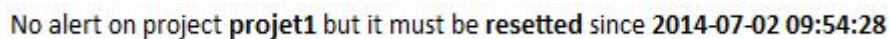
Alerts on **projet1** project:

- projet1\_sql
  - PING **FAILED**
- projet1\_fail
  - PING **FAILED**
- projet1\_prod
  - PING **FAILED**

**NB : first alert at 2014-07-02 09:39:44**

FIGURE 7 – Corps du message d'alertes en affichage HTML

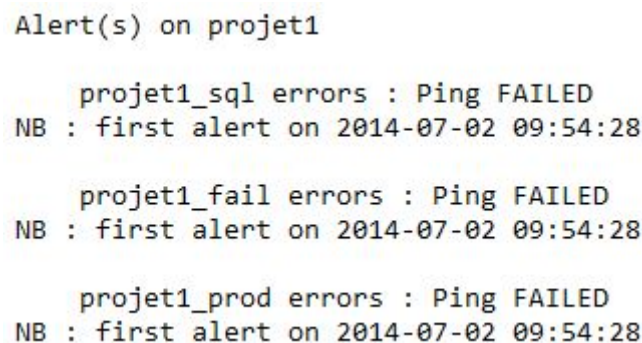
Cela pour la réinitialisation :



**No alert on project projet1 but it must be resetted since 2014-07-02 09:54:28**

FIGURE 8 – Corps du message de réinitialisation des alertes

Et la même chose en format plain-text :



**Alert(s) on projet1**

projet1\_sql errors : Ping FAILED  
NB : first alert on 2014-07-02 09:54:28

projet1\_fail errors : Ping FAILED  
NB : first alert on 2014-07-02 09:54:28

projet1\_prod errors : Ping FAILED  
NB : first alert on 2014-07-02 09:54:28

FIGURE 9 – Corps du message d'alertes en affichage plain

Le message réinitialisation est la même qu'en HTML, sauf que l'on supprime la mise en forme du texte.

## Script de lancement

L'application est pour l'instant exclusivement lancée via script qui permet à l'utilisateur de modifier certains comportements de déroulement du script de test.

Voici les différentes options de lancement :

- h, -help Affiche l'aide et quitte
- d Affiche les résultats dans la sortie standard
- D N'insère pas les alertes dans la BDD
- M N'envoie pas de mail en cas d'alertes
- r Réinitialise les alertes avant de lancer les tests
- R Réinitialise les alertes et quitter
- L Liste les projets et quitte
- t T Lancer des tests spécifiques

Ces différentes options sont ensuite traitées afin de générer un dictionnaire de configuration qui sera ensuite transmis à la classe Supervisor au moyen de la méthode "configure" de celle-ci. Si la configuration est jugée incorrecte, une erreur est renvoyée à l'utilisateur sur la sortie d'erreur de la console.

Si aucune erreur n'a été détectée et si l'option d'affichage des résultats est activée, alors les résultats provenant du Supervisor qui pour rappel ont la forme :

```
1 (
2     ['projet1',
3         [
4             [
5                 ['pylobby://www.google.fr:80', True, 'projet1_google'],
6                 ['pylobby://projet1.cyanide-studio.com:80', False, 'projet1_fail'],
7                 ['mysql://sql.cyanide-studio.com', True, 'projet1_sql']
8             ]
9         ],
10    ['projet2',
11        [
12            [
13                ['PyLobby://projet2.cyanide-studio.com:8016', False, 'projet2_prod'],
14                ['MySQL://projet2.cyanide-studio.com', False, 'projet2_sql']
15            ]
16        ]
17    ]
18 )
```

Sont mis en forme et affichés ainsi :

```
=====
1 -----
2 Project: projet1
3     test : pylobby://www.google.fr:80 : passed
4     test : pylobby://projet1.cyanide-studio.com:80 : FAILED
5     test : mysql://sql.cyanide-studio.com : passed
6 -----
7 Project: projet2
8     test : PyLobby://projet2.cyanide-studio.com:8016 : FAILED
9     test : MySQL://projet2.cyanide-studio.com : FAILED
10 -----
```

# Mailing

Cette a été réalisé dans l'optique de rajouter des fonctionnalité à un script de manipulation de mailing-list.

Ses principales fonctionnalité sont de permettre à un utilisateur de pouvoir s'inscrire ou se désinscrire d'une liste de diffusion. Mais aussi de gérer ces listes de diffusion.

## Structure de l'application

Celle ci est formée d'un script écrit en python, ainsi que de trois tables MySQL contenant les informations sur les utilisateurs ainsi que sur les liste de diffusion et une table de relation entre les deux.

Voici la structure de la table contenant les mailing-list :

- **id** : identifiant unique de la mailing-list
- **name** : nom de la mailing list, celui-ci sera utilisé comme adresse mail de la liste, exemple si "name" est "cyanide\_ml" alors l'adresse de la mailing-list sera "cyanide\_ml@cyanide-studio.com"
- **hidden** : indique si la liste est visible, en effet lors d'un abonnement ou désabonnement un mail d'informations signale, non seulement les mailing-list auquel l'utilisateur est abonné mais aussi la liste de toutes les mailing-list visibles existantes. Cette variable permet de cacher cette liste si sa valeur est de 1.
- **private** si mis à 1, cela interdit les utilisateur de pouvoir s'abonner.
- **public\_can\_send** : indique si les utilisateurs extérieurs à cyanide ont le droit ou non de poster sur cette liste
- **public\_can\_subscribe** : permet ou non aux utilisateurs externes à l'entreprise de pouvoir s'abonner

Et voilà celle concernant à proprement par les des utilisateurs symbolisés par leur mailbox ;

- **username** : nom de l'utilisateur et de sa boîte mail par la même occasion. Il permet aussi d'identifier l'utilisateur afin de lui transmettre un mail.
- **password** : mot de pass hashé de l'utilisateur
- **name** : nom et prénom de l'utilisateur.
- **active** : active ou désactive une mailbox, si elle est désactivée elle peut ni envoyer, ni recevoir de message.
- **domain** permet de spécifier le domaine d'appartenance de la mailbox, en effet cyanide possède une filiale au Quebec, il ne faut donc pas mélanger les différents comptes.
- **uid** identifiants unique de la boîte mail, en le couplant avec le domain on est capable de créer un couple unique qui permet d'identifier singulièrement l'utilisateur.
- **ml\_admin** donne à cette boîte mail la possibilité de poster dans des listes de diffusions même si l'utilisateur n'a pas souscrit d'abonnement à la liste préalablement.

Il existe bien entendu une table de pivot réalisant la relation Many-To-Many entre les boîtes mail et les listes de diffusion. En effet un utilisateur peut s'abonner à plusieurs mailing-list et chaque mailing-list possède plusieurs mailboxes de destinations.

- **id** identifiant unique de la relation
- **mailing\_list\_id** identifiant de la mailing-list
- **uid** identifiant de l'utilisateur

## Fonctionnement en abonnement/désabonnement

L'opération d'abonnement se réalise en envoyant un mail sans à l'adresse `subscription@cyanide-studio.com`, en précisant dans le corps du message les mailing-lists auxquelles on désirent souscrire séparées par des virgules.

Une relation est alors rajouté dans la table de pivot Many-To-Many.

De la même façon la désinscription est réalisé en adressant un mail sans sujet à l'adresse `unsubscribe@cyanide-studio.com` en indiquant les listes de diffusion auxquelles ont veut se désinscrire, séparées par des virgules.

La relation est alors supprimé de la table de pivot.

Un mail est ensuite envoyé pour récapitulé les abonnements de l'utilisateur, ainsi que les autres possibles souscriptions.

## Envoie de mails sur les listes de diffusion

### Vérifier l'autorisation d'un utilisateur

Un mail n'est considéré comme acceptable si et seulement si celui-ci a été signé et vérifié par un antivirus. Un entête est alors rajouté au head du message, cet header est de la forme :

```
1 | (Authenticated sender: user@cyanide-studio.com) by mail.cyanide-studio.com (Postfix) with  
   | ↪ ESMTP id XXXXXXXXXXXXX
```

Si le script ne détecte pas la présence de cette signature alors le mail est considéré comme frauduleux et le script s'arrête sans le transmettre à la liste de diffusion.

### Autorisation de poster dans cette liste

Avant de laisser l'utilisateur envoyer un message dans la liste de diffusion on vérifie qu'il est bien habilité à le faire, pour cela on vérifie si tout le monde est habilité à le faire.

Ou alors l'utilisateur est un administrateur et aucun test n'est réalisé.

### Récupération des différents membres de la liste de diffusion

Pour cela on utilise la table de relation pivot entre les mailboxes et les mailing-lists et on obtient le les adresses des différents utilisateurs concernés par l'envoi grâce à un LEFT JOIN sur la table des mailbox

### Purification du sujet du mail

En effet à chaque fois qu'un mail est répondu ou transféré une chaîne de caractère est rajouté ce qui peut donner de très long sujet peu lisible et compréhensible pouvant ressembler à :

`[ml] FWD : [ml] Re : [ml] Re : [ml] Re : [ml] Sujet` devenant `[ml] Sujet`

Cette transformation est possible à réaliser car à chaque passage par le script le nom de la liste de diffusion est rajouté entre crochet avant le sujet, il est donc possible de récupérer le dernier groupe de caractères entre crochets et de couper le sujet un caractère après la dernière occurrence de la mailing-list.

La seule contrainte est de ne pas pouvoir ajouter "*nom de la liste*" dans le sujet.



# CyaTracking

## Qu'est-ce que c'est

CyaTracking est un outil de statistiques permettant de recueillir diverses informations sur le trafic au sein d'une page de redirection. Une fois les informations collectées, l'utilisateur est redirigé vers une autre page.

## Cahier des charges

### Formalisticion des besoins

L'entreprise Cyanide possède un ensemble de pages boutique permettant l'achat des différents jeux qu'elle vend. Elle désire pouvoir centraliser l'ensemble des adresses de boutique vers une seule et même page qui se chargera de rediriger l'utilisateur vers la bonne page.

De cet état de fait il est possible de réaliser un système d'affiliation en Cyanide et d'autres site internet qui s'ils décident de faire apparaître sur leur page ou les réseaux sociaux un lien vers la page centrale de boutique, recevront un pourcentage sur chaque vente qui aurait transité par le lien de l'affilié.

L'URL choisie pour le lien respectera ce modèle-ci :

**http://store.cyanide-studio.com/?from=XXXXX&lang=yy\_YY**

où :

- XXXXX : représente le numéro de l'affilié
- yy\_YY : indique la langue de la boutique à afficher

Si les paramètres entrés sont incorrects ou inexistant il doit exister une adresse de redirection par défaut.

Un client devra permettre la visualisation des données des affiliés afin de connaître le nombre de redirection par affilié. La technologie pour celui-ci est libre.

### Contraintes

- Il est interdit de perdre une seule entrée.
- Le script doit être suffisamment rapide pour permettre un trafic de pointe d'au moins 100 connexions à la secondes.
- Les informations doivent être stockées en base de données MySQL.
- L'intégralité du service doit tourner sur FreeBSD 10.0.
- L'application doit être écrit en python version 2.7.
- Respecter l'unité dans les services, la page de redirection devra tourner sur le serveur HTTP Apache 2.x+.

## Structure de l'application

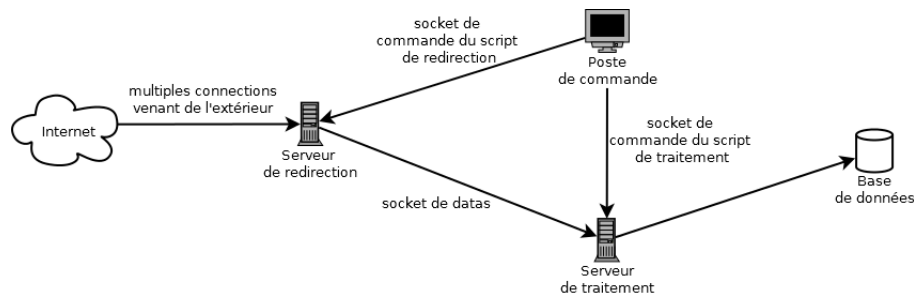


FIGURE 10 – Schéma réseau de l'ensemble de l'application

L'application était d'abord monolithique et situé sur le serveur de redirection, le traitement des informations est pour l'instant rapides mais dans l'avenir il est fort probable que ces processus soient plus long et complexes ce qui pourrait occasionner des ralentissements quant à la redirection de l'utilisateur. Il a donc été décidé de distribuer les différentes fonctionnalités dans différents deux scripts indépendants qui communiquent entre via un socket de données. Le tout commandé par un troisième script de commande permettant d'influencer le comportement des scripts, car ceux-ci tournant en arrière plan, on ne dispose pas d'accès direct pour lancer des commandes.

### Serveur de redirection

Ce serveur possède un accès vers le reste du Web, sa tâche consiste à recueillir les différentes requêtes utilisateur via un serveur HTTP Apache qui a son tour transmettra la requête au script de redirection qui se chargera de réorienter l'utilisateur vers la page voulue. Son rôle est aussi de transmettre les informations de l'utilisateur via socket au script de traitement.

### Serveur de traitement

Son rôle est d'attendre des données en provenance du serveur de redirection et d'insérer les informations en base de données. Son fonctionnement sera expliqué en détails dans une partie qui lui est consacrée.

### Poste de commande

Celui-ci permet de donner des ordres aux deux autres scripts. Là aussi de plus amples détails seront donnés ultérieurement.

### Base de données

Il s'agit d'un serveur MySQL 5.5, il contient les différentes informations recueillies lors des redirections.

#### Remarque :

Dans l'état actuel des choses le script de traitement, de redirection et de commande se trouvent sur le même serveur, mais rien n'empêche de les séparer.

## Script de redirection

Comme il a été dit plus haut, le script doit tourner sur le serveur Apache pour cela deux solutions s'offrent à nous :

- **mod\_python** : ce plugin Apache permet d'interpréter un script python de la même manière que php-fpm pour le langage PHP. Il est livré avec un ensemble de module permettant de réaliser plus facilement un site web. Ce plugin est désormais déprécié il est recommandé de ne plus l'utiliser.
- **wsgi** : le remplaçant de mod\_python, mais beaucoup plus léger s'appuyant sur l'utilisation de middlewares externes pour compléter les fonctionnalités manquantes.

## Structure minimale d'une application

```
def application(environ, start_response):  
    """  
    Simple Hello World wsgi app  
    """  
    output = "Hello World!"  
    response_headers = [('Content-type', 'text/html'), ('Content-Length', str(len("".join(output)  
    start_response("200 OK", response_headers)  
    return output
```

Comme on peut le remarquer la structure de code minimale nécessaire à l'affichage d'une page est très restreint, seul 2 paramètres sont transmis à la fonction :

- *environ* : qui contient les différentes variables transmises par Apache, on y retrouve des informations telle que l'ip publique de la personne adressant la requête, le langage par défaut du navigateur, ainsi que le libellé du navigateur lui-même et le système d'exploitation. Ainsi que l'URL complète demandée.
- *start\_response* : sert de lien entre le script python et le serveur Apache, il permet par exemple de modifier l'header et le statut de la réponse HTTP.

## Rediriger l'utilisateur

Nous pouvons donc facilement effectuer les redirection en manipulant le champ **Location** de l'header de la réponse HTTP.

```
def redirect(start_response, to):  
    response_headers = [('Content-type', 'text/html'),  
                        ('Location', to),  
                        ('Content-Length', "0")]  
    start_response("303 See Other", response_headers)  
    return [ ]  
  
def application(environ, start_response):  
    """  
    Simple Hello World wsgi app  
    """  
    output = "Hello World!"  
    return redirect(start_response, "http://somewhere.td")
```

Maintenant que nous possédons un moyen d'amener l'utilisateur sur une autre page, il s'agit de savoir où le rediriger. Pour cela il nous faut parser l'URL de requête via le module `urlparse` de python. Celui-ci nous permet de récupérer le chemin et les paramètres de l'URL simplement et même de parser les paramètres eux-même. Dans notre cas cela nous permet de récupérer les potentiels options `lang` et `from` contenus dans l'URL.

## Configuration des redirections

Afin d'apporter de la souplesse à l'application il a été décidé d'utiliser un fichier de configuration écrit en xml.

En voici la structure :

```
1 <projects default-redirection="http://www.cyanide-studio.com">
2   <project name="projet" default-redirection="http://store.cyanide-studio.com/project2"
3     ↪ disable="1">
4   </project>
5   <project name="projet1" default-redirection="http://store.project1.com">
6   </project>
7   <project name="projet2" default-redirection="http://store.project2.com">
8     <redirections>
9       <redirection lang="en_US"><![CDATA[http://store.somewhere.com/?typnews=]]></
10        ↪ redirection>
11       <redirection lang="en_GB"><![CDATA[http://store.somewhere.com/?typnews=]]></
12        ↪ redirection>
13       <redirection lang=""><![CDATA[http://store.somewhere.com/?Langue=en_GB&typnews
14        ↪ =]]></redirection>
15       <redirection lang="fr_FR"><![CDATA[http://store.somewhere.com/?Langue=fr_FR&
16        ↪ typnews=]]></redirection>
17       <redirection lang="de_DE"><![CDATA[http://store.somewhere.com/?Langue=de_DE&
18        ↪ typnews=]]></redirection>
19       <redirection lang="es_ES"><![CDATA[http://store.somewhere.com/?Langue=es_ES&
20        ↪ typnews=]]></redirection>
21       <redirection lang="nl_NL"><![CDATA[http://store.somewhere.com/?Langue=nl_NL&
22        ↪ typnews=]]></redirection>
23       <redirection lang="it_IT"><![CDATA[http://store.somewhere.com/?Langue=it_IT&
24        ↪ typnews=]]></redirection>
25     </redirections>
26   </project>
27 </projects>
```

## Déterminer redirection

La décision de la redirection se réalise sur plusieurs échelle :

Tout d'abord au niveau de l'application en elle-même, il est en effet possible d'indiquer une redirection par défaut globale, celle-ci est définie ligne 1.

Ensuite cette configuration peut spécifier pour chaque projet, on remarque d'ailleurs ici la possibilité de réaliser des alias entre les projets, comme c'est le cas pour le projet "projet" qui redirige vers le projet2, la requête est donc traité comme venant du projet 2.

Afin de rajouter de la granularité sur le choix de la langue de redirection chaque projet peut s'il le désire définir des redirections différentes par langues.

Par défaut, si aucun paramètre de langue n'existe dans la requête ou si celle-ci ne figure pas dans le fichier de configuration, alors la redirection ayant l'attribut `lang=""` sera utilisée.

Si une solution correcte de redirection n'est pas trouvée à l'étage courant on remonte au parent et ainsi de suite jusqu'à la redirection par défaut de l'application.

## **Le numéro d'affilié**

S'il est transmis le numéro d'affilié est contenu dans le paramètre *from* de l'URL. Celui-ci doit-être un nombre. Dans le cas contraire un numéro d'affilié par défaut arbitrairement choisi comme étant le 99, sera utilisé.

## **La gestion des alias**

Du fait que l'on fait rebondir l'utilisateur d'une page à une autre, il nous faut donc réécrire l'URL avant de la relancer pour traitement Ceci est grandement facilité par `urlparse` qui nous permet de récupérer les paramètres de l'URL et de simplement les ajouter au lien de redirection.

L'ensemble de l'algorithme de redirection est détaillé ci-dessous :

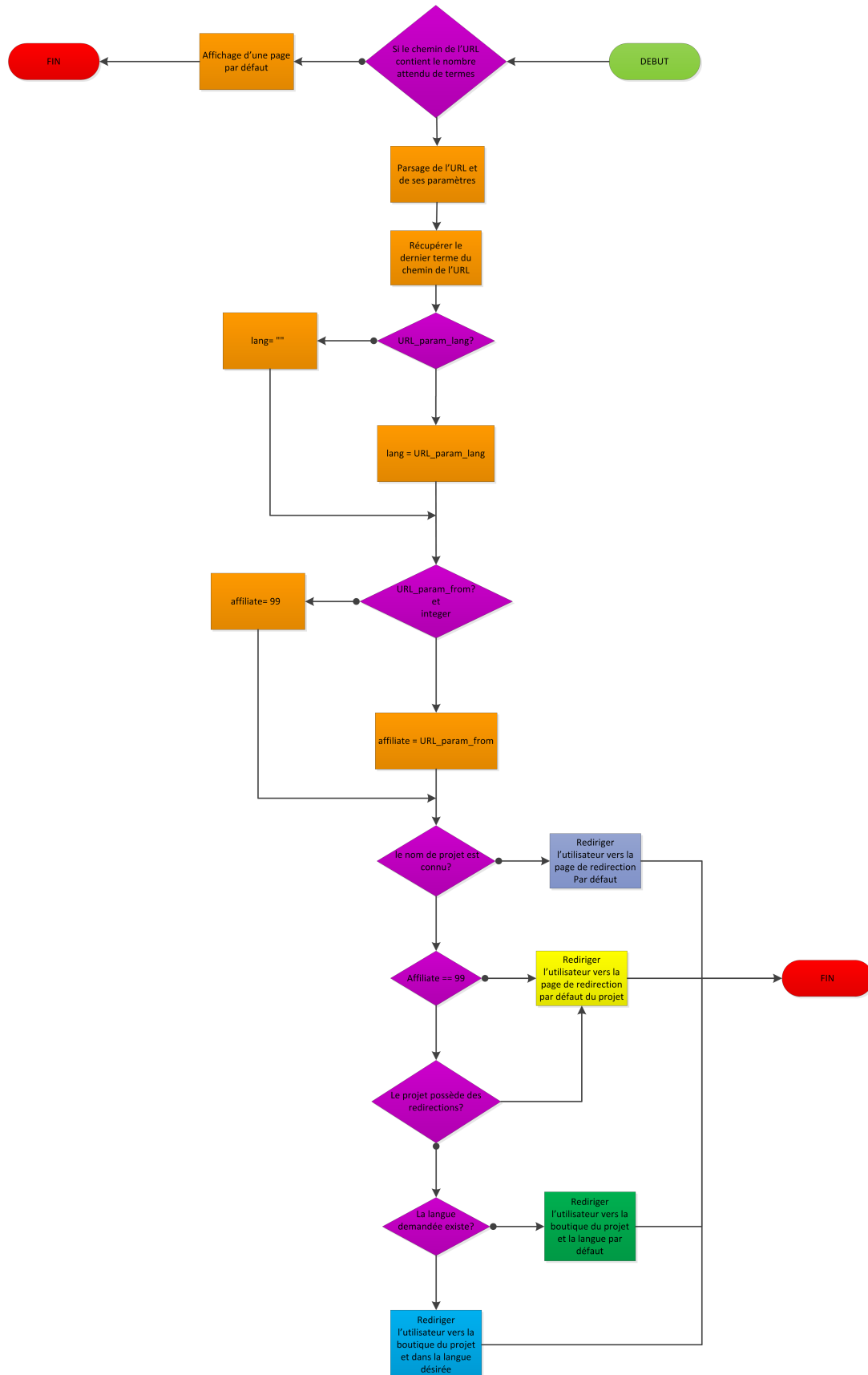


FIGURE 11 – Algorithme de redirection

## Récupération des données utilisateur

Comme dit plus haut, la variable *environ* passée en paramètre de la fonction *application* nous permet de récupérer une somme d'information intéressante si celles-ci sont transmises par le navigateur client.

Les informations considérées comme ayant de l'importance pour nous sont :

- HTTP\_ACCEPT\_LANGUAGE : contient les différentes langues comprises par le navigateur client.
- HTTP\_USER\_AGENT : permet de déterminer le navigateur (Chrom, Opéra, IE, Firefox, ...) ainsi que le système d'exploitation de l'utilisateur (Microsoft, Linux, IOS, Andorid, ...).
- REMOTE\_ADDR : récupère l'ip du client, pour de la géolocalisation par exemple.

L'ensemble de ces informations si elles existent, dont en plus du nom du projet et du numéro de l'affilié sont transformés en un dictionnaire python. Par cette portion de code :

```
1 #get interesting datas
2 datas = {
3     "affiliate" : affiliate,
4     "project" : projectName,
5 }
6 try:
7     datas["prefered_language"] = environ["HTTP_ACCEPT_LANGUAGE"]
8 except KeyError:
9     pass
10 try:
11     datas["user_agent"] = environ["HTTP_USER_AGENT"]
12 except KeyError:
13     pass
14 try:
15     datas["remote_ip"] = environ["REMOTE_ADDR"]
16 except KeyError:
17     pass
```

## Transmission des données

Si le projet n'est pas désactivé pour le tracking des données, il est possible d'envoyer les données par le moyen d'un socket et ce fourni par la librairie standard.

En début de lancement du script de redirection, le mod\_wsgi tente d'ouvrir un socket sur le port d'écoute choisi dans la configuration si le port n'accepte pas de connections l'erreur est écrite dans un fichier de log et le script effectue seulement les redirection mais ne réalise aucun enregistrement des données utilisateurs.

Dans le cas où la connexion avec le script a pu être établie, et si le projet concernée par la redirection est actif (cf :configuration des projets). Alors le transfert est réalisé par cette ligne de code :

```
1 datas = cPickle.dumps(datas)
2 size = len(datas)
3 pattern = "{:0%d}"%8
4 dataSocket.sendall(pattern.format(size)+datas)
```

Ici datas correspond au dictionnaire de données à transmettre et la taille de l'header est définie à 8 caractères.

## Comment transmettre ces données

Le choix d'avoir décider d'acheminer les données par socket python nous impose la transmission des information par chaîne de caractères. Pour ce faire il nous faut sérialiser les données contenu dans le dictionnaire. Nous avons décidé d'utiliser le module Pickle de la librairie standard et même ça version compilée cPickle qui est 25 % plus rapide que son homologue pur Python.

exemple :

```
1 >>>monDict = {"param1":"value1", "param2":42, "param3":["element1", 12, True]}
2 #une fois dumpée
3 >>>dump = cPickle.dumps(monDict)
4 #génère cette chaîne de caractère
5 >>>print dump
6 (dp1
7 S'param3'
8 p2
9 (lp3
10 S'element1'
11 p4
12 aI12
13 aI01
14 asS'param2'
15 p5
16 I42
17 sS'param1'
18 p6
19 S'value1'
20 p7
21 s.
22 #de même il est possible de retransformer cette chaîne de caractères en dictionnaire
23 >>>print cPickle.loads(dump)
24 {'param3': ['element1', 12, True], 'param2': 42, 'param1': 'value1'}
```

Nous possédons désormais un moyen efficace pour effectuer les transactions entre les différents script.

Mais ceci n'est pas suffisant car il s'est avéré qu'à un très nombres de connections simultanées, le système d'exploitation FreeBSD contrairement à Microsoft, fusionnait les différents dictionnaire en une seule chaîne de caratère et non en plusieurs paquets de données comme c'est le cas sur Windows, il a donc fallut réaliser un protocole de transmission de données.

La taille et le nombre des paquets d'informations étant très variables et donc imprédictible côté réception. Un problème se pose lorsque l'on désire dissocier les groupes de données entre eux.

Nous avons donc choisi de réserver un certains nombre de caractères pour indiquer la longueur de la chaîne de caractère contenant les données sérialisée

## paquet de données transmis par la netstack de FreeBSD



FIGURE 12 – Structure de données transmise par socket



## Script d'insertion

Celui-ci est composé de deux thread et d'une fonction permettant de générer un mail en cas d'alerte :

- un thread serveur écoutant sur un éventuel client de datas ou de commandes.
- un thread s'occupant d'insérer les données en BDD.

## Structure du serveur d'écoute

Celui-ci se présente sous la forme d'une boucle infinie, cette structure débute par select qui est un appel système qui prends en paramètre trois listes de socket :

- connexions entrantes : dans notre cas elle est composée du socket de datas et du socket de commande
- connexions sortantes
- erreurs inexplicables en entrée ou en sortie

Le select renvoie en sortie si un évènement correspondant aux sockets contenu dans les listes est survenu, un tuple de trois liste contenant les sockets concernés par l'évènement. Sinon le select bloque la boucle évitant au script de vérifier périodiquement si l'évènement à eu lieu, en effet c'est le système d'exploitation qui effectue ce test.

Ensuite on vérifie quel socket à été réveiller et pour quelle raison, si le socket se trouve être celui de commande, on utilise la commande système *accept* sur le socket ainsi réveiller cette méthode renvoie un autre socket contenant la connexion entre le client et le serveur. On ajoute ce socket à la liste des connexions entrantes ainsi qu'à une autre liste contenant tous les sockets de commandes actifs. De même si le socket activé s'avère être celui de datas on réalise les mêmes opérations à ceci prêt que l'on ajoute cette nouvelle connexion dans la liste des sockets de datas actifs, puis on retourne au select.

Si le socket n'est ni un socket de datas ni un socket de commande , c'est donc qu'il s'agit d'un socket généré par la méthode *accept*, il peut donc soit être de type *datas* soit de type *cmd*.

Pour le déterminer on vérifie l'existence de ce socket dans la liste des sockets de commandes actifs, si c'est le cas on récupère un paquet de donnée de longueur fixe grâce la méthode *recv* du module **socket**, si aucune donnée valide n'a été transmise alors cela signifie que le client à simplement envoyer un signal SIGPIPE de fermeture de socket, il faut donc fermer le socket grâce à *close*, puis supprimer cette connexion détruite des connexions entrantes ainsi que des sockets de commandes actifs. Si les datas sont valides on les traite on exécute l'ordre si possible et on revient au select en attente d'un nouvel évènement.

Si le socket réveillé n'était pas non plus de la commande, on vérifie par acquis de conscience que le socket est bien du type *datas*, et de même si aucune donnée n'est transmise c'est donc le client (le script de redirection dans notre cas) a mis fin à la connexion, il faut donc là aussi fermer le socket, supprimer la connexion de la des connexions entrantes et enfin retirer la connexions des sockets de datas actifs. dans le cas contraire on les traite et on retourne au select.

## Traitement de la commande

Pour que données de commande il faut qu'il s'agisse d'un nombre, si ce n'est pas le cas on retourne au select.

Pour le moment seul deux ordres sont disponibles :

- 0 : ceci correspond à l'ordre **stop** qui permet de stopper le script en mettant à True une variable globale
- 1 : cet ordre **antispam** permet en modifiant une variable globale de réactiver l'envoi de mail en cas d'erreur

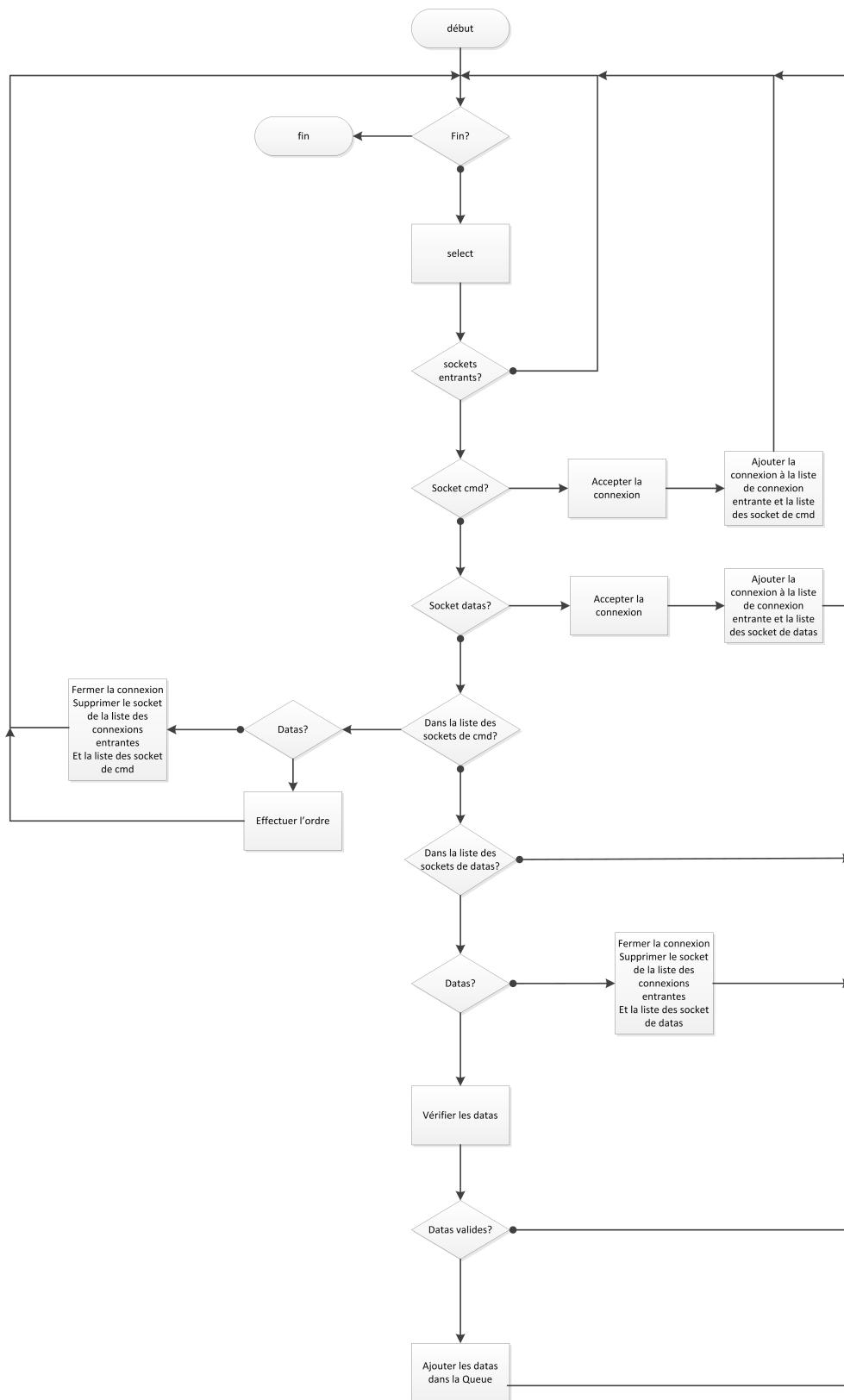


FIGURE 13 – Structure du serveur d'écoute

## Traitement des données

### Cas de figures défavorables

Pour rappel un paquet d'informations est constitué d'un header contenant la taille en nombre de caractères des données sérialisée ainsi que les données a proprement parler.

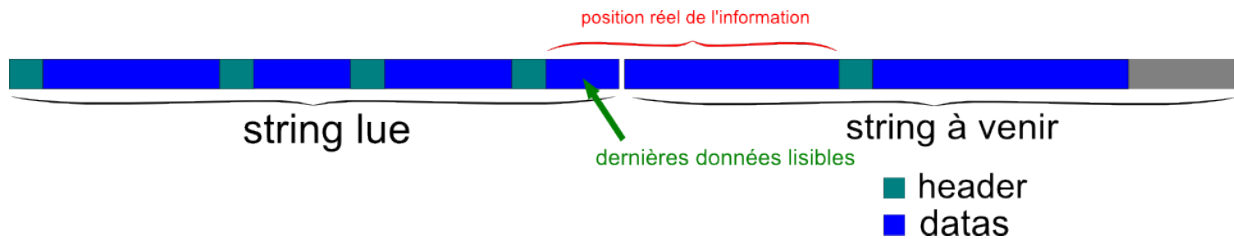


FIGURE 14 – Exemple de réception de paquet incomplet

À chaque appel de la fonction *recv* sur le socket de datas nous lisons un nombre fixe de caractères, il peut donc arriver que les datas ne puissent pas tenir sur la zone de lecture courante et donc se retrouve tronquée. Il faut donc attendre la prochaine lecture afin de reconstituer la donnée dans son ensemble.

De la même manière il peut aussi arriver que l'header soit réparti entre deux lectures. Dans ce cas il est impossible de se prononcer quand à la longueur des données attendues, il faut là aussi attendre un prochain train d'information pour prendre une décision.

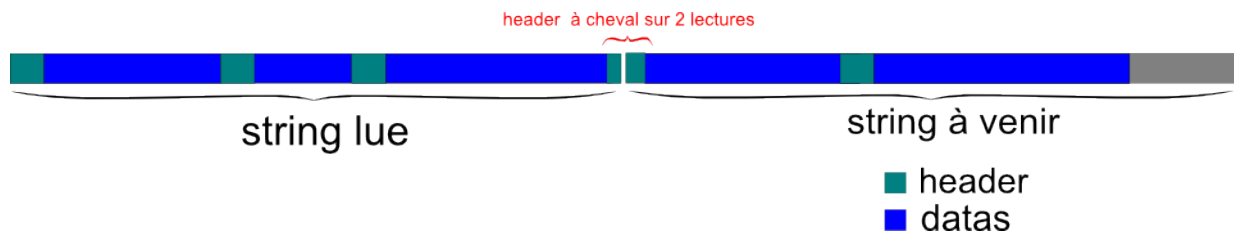


FIGURE 15 – Exemple de réception d'header incomplet

## Algorithme de lecture

Toutes données reçues par le socket de datas est transmis à un buffer de chaîne de caractères. Tant que la longueur de ce buffer est supérieur à la longueur d'un header et donc que l'on est capable de déterminer la taille d'une donnée à attendre, ce annulant l'hypothèse de la figure 6. On applique cette algorithme :

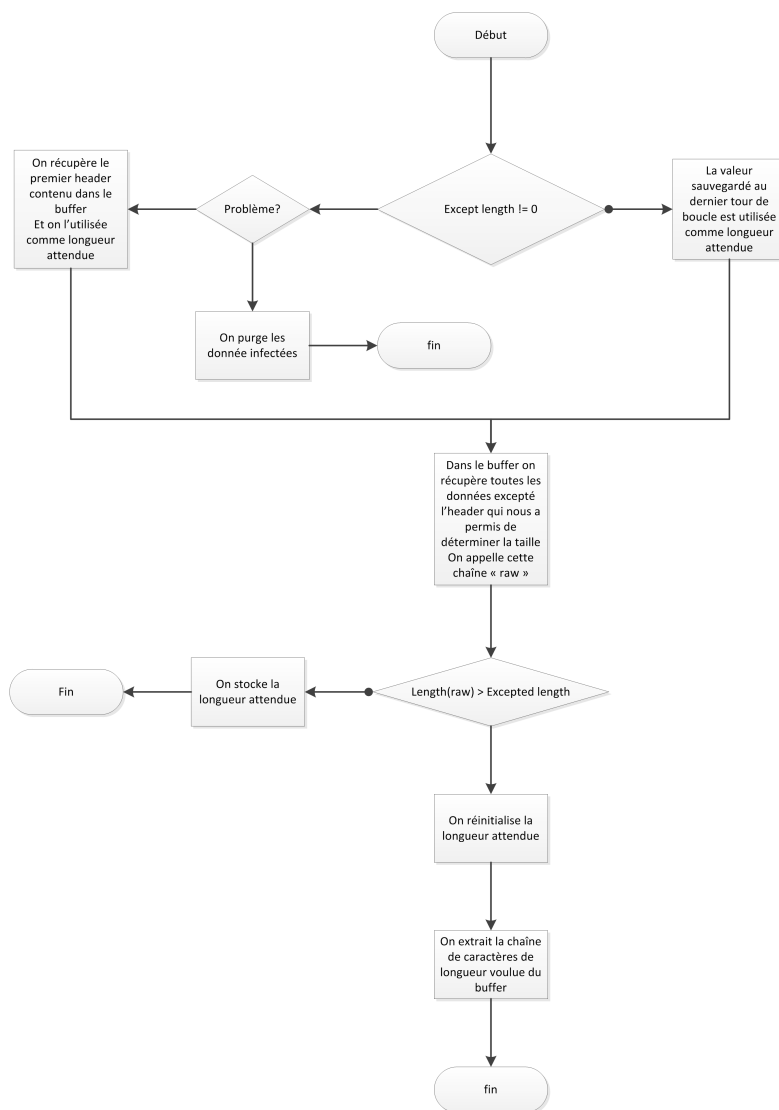


FIGURE 16 – Algorithme de lecture de données transmises

## Désérialisation

Par sécurité, si la désérialisation des données échoue on purge les données car on ne peut pas déterminer à quel moment l'erreur est apparue.

Si aucun problème n'est survenu, le dictionnaire obtenu est ajouté à une FIFO commune aux threads de select et d'insertion en BDD.

# Stockage des données

## Moteur de BDD

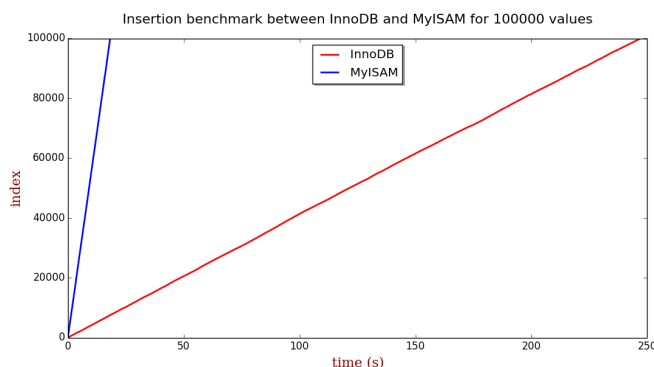
Un moteur de base de données est l'ensemble des différents mécanismes permettant la manipulations des informations stockées.

## Deux prétendants

Les moteurs les plus utilisés sur les serveurs MySQL se nomment :

- MyISAM
- InnoDB

Pour tester les performances de ces deux moteurs, réalisons une série de 100000 insertions de 6 champs chacun. Cette expérience nous présente le résultat suivant :



Engine	inserts.s <sup>-1</sup>
InnoDB	404
MyISAM	5479

On remarque que le moteur InnoDB est plus de 13.5 fois plus lent que MyISAN pour effectuer la même opération.

Ceci est dû au fait que le moteur InnoDB est ce que l'on nomme transactionnel. Pour rappel une transaction est une séquence de commande qui commence par un BEGIN, continue par des opération CRUD (SELECT, INSERT, UPDATE, DELETE) et fini soit par un COMMIT qui valide les opérations précédente soit par un ROLLBACK qui les annulent. Contrairement à MyISAM qui se contente des opérations CRUD.

Mais la vitesse d'insertion n'est pas l'unique condition de choix. En effet nous désirons aussi une sécurité sur l'insertion des données, ce qui nous est permis grâce au système de transaction qui en cas d'erreur permet d'annuler l'insertion et de ne pas compromettre les données.

De plus InnoDB semblent tout à fait capable d'assurer les 100 insertions par secondes requis. Nous utiliserons donc ce moteur pour la table qui contiendra les données utilisateur.

## Gérer les problèmes de communication avec la BDD

Les erreurs d'enregistrement peuvent être provoquées par des problèmes temporaires comme de la latence sur la ligne ou un redémarrage du serveur contenant la base de données. Nous avons donc besoin d'un moyen de pouvoir nous reconnecter en cas de déconnexion non désiré. Pour se faire nous avons décidé d'utiliser le connecteur mysql développé par Oracle "python mysql connector" celui-ci permet en effet se reconnecter à base de données

## Stockage des données

L'insertion des données est effectuée dans un thread infini retardé par un sleep de 0.001 seconde afin de ménager le temps processeur. A chaque tour de boucle on vérifie si la FIFO commune au 2 threads n'est pas vide, si ce n'est pas le cas alors on dépile un élément, celui-ci est bien évidemment le dictionnaire généré par le script de redirection lors du passage d'un utilisateur, afin de l'horodater on rajoute une clef "date" puis on essaie 3 fois d'insérer les données. Si cela échoue les deux premières fois on effectue un rollback de la transaction on rempile le dictionnaire dans la FIFO, puis on attends 5 secondes avant de réessayer, au bout de 3 essais on abandonne définitivement et on envoie un mail. Si tout c'est bien passé on commit la transaction.

## Gestion de la configuration

Comme évoqué pour la gestion des redirections des projets, l'ensemble de la configuration est géré par un fichier XML, il est structuré en deux parties :

- La configuration des scripts
- La configurations des projets

# CyaBackup

## Présentation

CyaBackup est système de journalisation de fichiers, permettant la sauvegarde à intervalles régulier de données. En outre il est aussi capable de générer des backups sur des systèmes clefs comme les annuaires LDAP ou les base de données MySQL.

## Spécification techniques

L'intégralité de l'application est réalisée en python ou en utilisant exclusivement les composants de la librairie standard.

La configuration de la journalisation se fait au moyen d'un fichier de configuration au format .ini tel que définit ici [http://en.wikipedia.org/wiki/INI\\_file](http://en.wikipedia.org/wiki/INI_file) à l'exception du symbole de commentaire qui devient #.

L'utilitaire est lancé en ligne de commande.

### Ligne de commande

#### fichier de configuration

Un fichier de configuration peut-être spécifié au moyen du paramètre -config FILE

Si aucun fichier de configuration n'est spécifié, le fichier Backup.ini du répertoire d'exécution du script sera utilisé en tant que fichier de configuration, si celui-ci n'existe pas une erreur est levée. Si un fichier de configuration est spécifié il sera utilisé.

Si le fichier spécifié contient une incohérence, le script s'arrête et une erreur est levée,

### Debug

L'utilisateur peut demander plus d'information au moyen du paramètre -v ou -vv

### Abstraction de la destination du backup

Le script permet de spécifier un dossier de destination qui est soit sur un système de fichier local ou monté en lecteur réseau soit sur un serveur distant en passant par le protocole FTP, L'abstraction est réalisée grâce à la librairie AbstractFS.

## Le fichier de configuration

Comme dit plus haut, un fichier de configuration incorrect entraîne l'arrêt du script, il faut donc respecter la syntaxe des .INI, mais pas seulement :

- Le fichier doit contenir obligatoirement une section [general], celle-ci doit elle même être renseigné sur les champs **email** et **tmpdir**.
  - Le champs **email** est pour l'instant inutilisé, mais il pourra être couplé à un système de mailing pouvant avertir l'utilisateur d'une erreur de journalisation.
  - Le champs **tmpdir** est le répertoire par défaut de travail du script.

La section [general] possède 3 sous-sections :

- [[MySQL]] : permet de spécifier les options par défaut de journalisation de BDD, plusieurs champs peuvent/doivent être remplis :
  - **mysqldumppath** (obligatoire) : correspond au chemin absolu de l'exécutable mysqldump sur le pc
  - **tmpdir** : argument facultatif permettant de spécifier un répertoire de travail différent pour la journalisation de BDD, peut-être pratique pour les dump volumineux.
  - **skiplist** : spécifie les tables d'une BDD qui ne seront pas journalisées
  - **dst** : spécifie le répertoire destination des dumps, les chemins de destination seront développés dans une section qui leur est propre
  - **version** (obligatoire) : indique la version de l'utilitaire mysql
  - **login-path** : permet de spécifier un login-path pour les version supérieure à la 5.6, en effet il est déprécié d'envoyer le mot de passe en clair durant une transaction MySQL depuis la version 5.6.
- [[Ldap]] : permet de spécifier les options par défaut de journalisation des annuaires Ldap
  - **tmpdir** : argument facultatif permettant de spécifier un répertoire de travail différent pour la journalisation d'annuaires Ldap
  - **dst** : spécifie le répertoire destination des dumps, la configuration des chemins de destination sera développée dans une autre section
- [[File]] : permet de spécifier les options par défaut de journalisation des annuaires Ldap
  - **tmpdir** : argument facultatif permettant de spécifier un répertoire de travail différent pour la journalisation de fichier et répertoires
  - **dst** : spécifie le répertoire destination des dumps

Les sections suivantes contiennent diverses option qui écraseront les données de la section générale si les noms de champs coïncident :

La section [MySQL] permet de lister les différentes BDD et de paramétrer la journalisation pour chacune d'elles.

Chaque sous section est un label qui n'a aucune incidence sur la sortie mais permet seulement à l'utilisateur de s'y retrouver dans les différentes BDD qu'il souhaite journaliser, ainsi qu'au programme pour les séparer les unes des autres.

- **src** (obligatoire) : spécifie le serveur à journaliser, la source est définie en tant qu'une URL et doit donc respecter le format mysql ://user :pass@host :port/?db=db1,db2, une URL minimale étant mysql ://user :pass@host.
- **dbList** : indique quelle bdd du serveur doivent être journalisées, si le paramètre db de l'url src est spécifié, celui-ci tient lieu de dbList et le présent paramètre devient caduque.
- **skipdefaultlist** : si ce paramètre est à true, la skiplist de la section [general] sera remplacé par la skiplist de la présente sous-section
- **skiplist** : permet d'indiquer les bdd qui ne doivent pas subir de journalisation
- **dst** : indique le répertoire de destination de cette journalisation
- On peut aussi indiquer 2 sous-sous-section [[[archive]]] et [[[rotation]]], mais leur utilité sera développée dans une partie qu'il leur est consacrée.



La section **[Ldap]** permet de lister les différents annuaires Ldap et de paramétrer la journalisation pour chacune d'eux.

Chaque sous section est un label qui n'a aucune incidence sur la sortie mais permet seulement à l'utilisateur de s'y retrouver dans les différents annuaires qu'il souhaite journaliser, ainsi qu'au programme pour les séparer les unes des autres.

- **src** (obligatoire) : spécifie le serveur à journaliser, la source est définie en tant qu'une URL et doit donc respecter le format d'une URL minimale étant ldap ://host.
- **dst** : indique le répertoire de destination de cette journalisation.
- **binddn** (obligatoire) : spécifie l'utilisateur ldap.
- **passwd** (obligatoire) : spécifie le mot de passe ldap
- **basedn** (obligatoire) : indique le noeud de recherche racine
- De la même façon que pour la section **[MySQL]**, il est possible de spécifier les sous-sous-section **[[[archive]]]** et **[[[rotation]]]**

La section **[File]** permet de lister les différents répertoires et fichiers voulant être journalisés et d'en spécifier les paramètres.

Chaque sous section est un label qui n'a aucune incidence sur le code mais permet seulement à l'utilisateur de s'y retrouver dans les différents annuaires qu'il souhaite journaliser, ainsi qu'au programme pour les séparer les unes des autres.

- **src** (obligatoire) : spécifie le serveur à journaliser, la source est définie en tant qu'une URL
- **dst** : indique le répertoire de destination de cette journalisation
- **tmpdir** : spécifie le répertoire de travail
- Tout comme la section **[MySQL]** et **[LDAP]**, on peut aussi indiquer 2 sous-sous-section **[[[archive]]]** et **[[[rotation]]]**

Ci dessous un exemple de fichier de configuration valide

```
1  #Always use / instead of \ for paths on Windows or it can escape characters.
2  # || can be used to escape itself, but then || must be written ||| which can become painfull
3
4  [general]
5      email    = yguern@cyanide-studio.com
6      tmpdir    = C:/tmp
7
8      [[MySQL]]
9
10     #this the path to the mysqldump executable
11     mysqldumppath = C:/Program Files/MySQL/MySQL Server 5.6/bin/mysqldump.exe
12
13     #temp dir, cleared after running
14     tmpdir        = s:/tmp
15
16     #Any subsection specific option put here will be interpreted as default:
17     #You should always skip 'performance_schema' is this db is not dump-able
18     # list => merged (can be replaced)
19     # other => replaced
20     skiplist       = information_schema, performance_schema, mysql
21
22     #destination folder for backup. Will be created if non-existent, avoid root partition
23     dst            = C:/dst
24
25     #pattern either x.y or xx i.e. that 5.2 or 52
26     version        = 5.6
27
```

```

28     #local-path config, if none value is specified, local-path sets to "local"
29     login-path      = bsd
30
31     [[Ldap]]
32         tmpdir = s:/tmp
33
34     [[File]]
35         tmpdir = s:/tmp
36
37     [MySQL]
38
39     [[postfix]]
40         #If query db is specified (i.e mysql://domain.com/?db=db1,db2)
41         #Only databases listed in this query will be backedup and skipList will be ignored
42         src      = mysql://root:test@host:3306
43
44         dblist   = db1
45
46         skipdefaultlist = true
47
48         dst      = s:/bkup
49
50         skiplist =
51
52         [[[archive]]]
53             compress = tar
54
55         [[[rotation]]]
56             monthsoff  = 12
57
58             max_files  = 3
59
60     [Ldap]
61     [[IT]]
62         dst      = ftp://user@ftp.host.dst
63
64         src      = ldap://ldap.host.somewhere
65
66         #put binddn and basedn between "" because of , character
67         binddn   = "cn=User,dc=cyanide-studio,dc=com"
68
69         passwd   = *****
70
71         basedn   = "dc=cyanide-studio,dc=com"
72
73     [File]
74     [[test]]
75         dst      = s:/dst
76
77         src      = s:/src
78
79         tmpdir   = s:/tmp
80

```

```

81     [[[archive]]]
82         compress      = gz
83
84     [[[rotation]]]
85         rotate        = true
86
87         daysoff       = 3
88
89         weeksoff      = 2
90
91         dayofweek     = 7
92
93         monthsoff     = 5
94
95         max_files     = 0
96
97         min_files     = 0

```

## Les urls

Les urls permettent de spécifier des répertoire d'entrées-sorties, facilement en utilisant le système de schéma qui indique sur quel système de fichier la ressource se trouve, par exemple une url débutant par ftp ://, indique que la ressource est située sur un serveur FTP à l'inverse une ressource dans un système de fichier local débuttera par file ://. Remarque : si aucun schéma n'est défini, le système considérera la ressource comme locale.

## La sous-sous-section [[[archive]]]

Permet d'indiquer que l'on souhaite archiver les backups compress : permet de paramétrer le mode compression de l'archive Pour l'instant les seuls modes disponibles, sont :

- tar : pas de compression
- bz2
- gz

## La sous-sous-section [[[rotation]]]

Si celle ci est présente, permet d'activer le système de rotation des backups Elles présente de nombreuses options

- **rotate** : si mis a true active la rotation
- **max-files** : limites le nombres de fichiers généré
- **min-files** : cap le minimum de backup présent, ceci est une sécurité

Remarque : si rotate est à false : le système ne gardera que les max-files plus récents fichiers

Maintenant dans le cas d'une rotation activée :

- **daysoff** : spécifie le nombre maximale de jours qui seront conservés.
- **weeksoff** : nombre de semaine conservées
- **dayofweek** : jour de la semaine où la journalisation se lance
- **monthoff** : nombre de mois conservés
- **dayofmonth** : jour du mois où la journalisation est effectuée

Remarques : Si les paramètres de rotation sont en contradiction avec les paramètres max-files et min-files, la rotation sera affectée par ces deux paramètres.

## Déroulement du script

### Script de ligne de commande

Vérifie l'existence du fichier de configuration et le transforme en dictionnaire avant de le donner en paramètres à la classe Backup.

### Class Backup

Le point d'entrée du script se fait par la classe Backup, son rôle est de s'assurer que la syntaxe du fichier de configuration est respectée et que les champs de configuration minimal sont bien remplis, si ce n'est pas le cas celui-ci lève une erreur et le script s'arrête. Dans le cas où la fichier de configuration est valide, lorsqu'il rencontre une section qui n'est pas la section [general], il tente d'importer le module class[non de la section].py, si celui-ci n'existe pas une erreur est levée. Si le module est importé avec succès, le script tente d'instancier la classe Bkup commune à tous les classes nom de section en utilisant comme paramètre de constructeurs les argument de la sous-section non de la section de la section general , si l'instanciation échoue une erreur est levé et le script s'arrête. Si la classe est correctement instancée, le script lance la fonction launch avec en paramètre les arguments contenus dans la sous-section du label de backup.

### Les classe de Backup

#### Classe Bkup de Mysql

La classe vérifie tout d'abord la validité des options transmis en paramètre de la fonction launch, si ceci sont invalide une erreur est ajouté à la liste des erreurs et on quitte la fonction. Si les options sont valides, la classe tente de se connecter au serveur Mysql, si cela échoue une erreur est renvoyé, si la connexion réussis l'utilitaire mysqldump effectue le dump de toutes les bases de spécifiées. Les dump sont ensuite archivés si besoin, les dump sont ensuite déplacés dans le répertoire de destination, puis, si la rotation est demandée, les fichiers indésirables sont supprimés.

#### Classe Bkup de Ldap

La classe vérifie tout d'abord la validité des options transmis en paramètre de la fonction launch, si ceci sont invalide une erreur est ajouté à la liste des erreurs et on quitte la fonction. Si les options sont valides, la classe tente de se connecter au serveur Ldap, si celà échoue une erreur est renvoyé, si la connexion réussis ldapsearch effectue le dump de l'annuaire.

#### Classe Bkup de File

La classe vérifie tout d'abord la validité des options transmis en paramètre de la fonction launch, si ceci sont invalide une erreur est ajouté à la liste des erreurs et on quitte la fonction. Les dump sont ensuite archivés si besoin, les dump sont ensuite déplacés dans le répertoire de destination, puis, si la rotation est demandée, les fichiers indésirables sont supprimés.

### classArchive

Effectue les opération de compression et de packaging grâce à la librairie tarfile, il est possible de spécifier si l'on veut ou non garder le fichier à archiver après archivage.

## **classRotate**

Effectue les opérations de rotation, en utilisant les paramètres qui lui sont donnés, si l'un d'eux est incorrects une erreur est levée. Pour réaliser la rotation, la classe génère une liste de validité en se basant sur le contenu du fichier qui possèdent tous le même pattern : `nom_annee-mois-jour`. Les fichiers n'étant pas dans la liste de validité sont supprimés.

## **Mailing**

Afin d'avertir l'utilisateur de potentielles erreurs durant un run via cron (donc sans stdout d'accès), un système de mailing a été mis en place.

## **Abstraction de la destination**

Le système d'URL nous permet une relative souplesse quant à la destination des backups, mais cela pose aussi le problème de pouvoir manipuler plusieurs systèmes de fichiers indifféremment.

Ceci a poussé à la création d'une librairie réalisant les différentes manipulations voulues, tel que copier sur un FTP des données stockées sur un NTFS et pourquoi pas d'un FTP vers un SSH ou en passant via un web-service en HTTP.

La prochaine section parle plus amplement de cette bibliothèque de fonctions.

# AbstractFS

Est une librairie de fonctionnalité permettant de réaliser des opérations de manipulation de système de fichier sur plusieurs différent support comme le FTP, le SSH ou les web-service.

Elle est implémenté en python en utilisant seulement les composant disponible dans la librairie standard, ce qui lui permet d'être utilisé sur n'importe quel système possédant python 2.6 – 2.7.

Son rôle est de permettre la création, la suppression de dossier ou les opérations de copie, déplacement de suppression de fichiers.

## Structure

La librairie est composée de plusieurs modules ayant chacun leur rôle :

- `absException.py` : contient les exception spécifique à la librairie
- `abstractFS.py` : interface utilisateur de la librairie
- `base.py` : module contenant la classe abstraite `FS()` d'où dériveront toutes les classe spécifique de manipulation de fichier, elle sert principalement au développeur pour lui permettre de déterminer les fonctionnalités qui lui reste à implémenter
- `ftp.py` : module de système de fichier FTP, en utilisant la `ftplib` de la STL python
- `file.py` : essentiellement un wrapper de la librairie `os` de la STL python.
- `ssh.py` : non implémenté pour l'instant mais il le sera dans le futur.

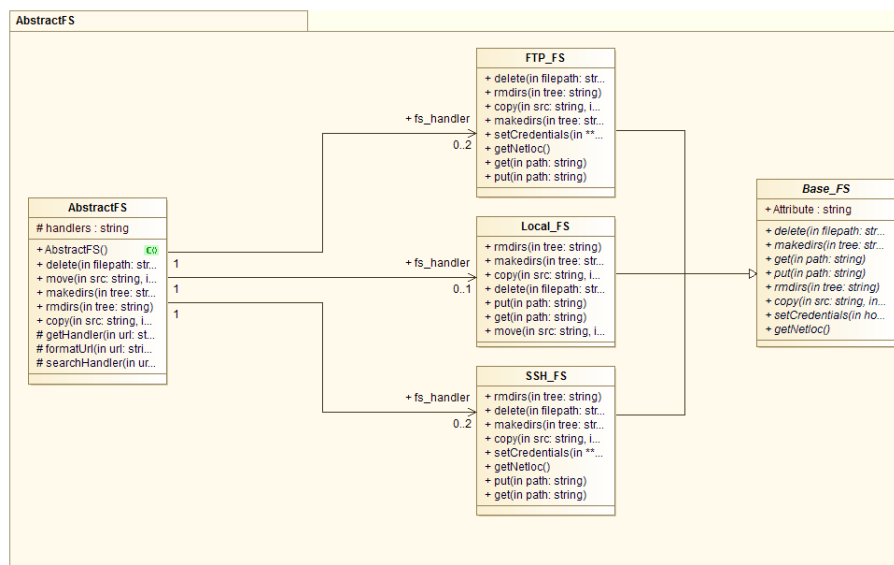


FIGURE 18 – Package AbstractFS

## Fonctionnement de la couche d'abstraction

### mode d'utilisation

La librairie présente un objet `fs` qui est utilisé comme le serait `os`

## Les différentes interfaces publiques

Les noms sont essentiellement calqués sur les noms de fonction des bibliothèques `shutils` et `os`

- `copy(src, dst)` : copie un fichier source vers une destination
- `move(src, dst)` : déplace un fichier source vers une destination
- `remove(path)` : supprime un fichier
- `makedirs(path)` : créer une arborescence
- `rmdir(path, delete)` : supprime une arborescence, si `delete` vaut `true` les fichiers sont supprimés, sinon si l'arborescence contient des fichiers une erreur est levée.
- `listdir(path)` : liste les éléments contenu dans `path`
- `isFile(path)` : vérifie si `path` est un fichier

Ces méthodes sont les seuls disponibles à l'utilisateur.

## Les méthodes privés

### `__get_handler`

Cette méthode permet soit d'instancier une classe de manipulation de système de fichier soit de récupérer une instance, si celle-ci est disponible.

Tout d'abord on vérifie si un handler du type désiré est disponible dans la liste des handler déjà enregistrés. Si aucun handler ne correspond on en instancie un nouveau du type désiré. Dans le cas où le handler du bon type est trouvé on utilise celui, afin de vérifier si les paramètres (utilisateur, mot de passe, adresse du serveur, etc...) sont correctement renseignés. Si la configuration est correcte, on compare les identifiants de chacun des handlers afin de vérifier si un handler possédant les mêmes identifiants de connexion, n'existe pas déjà, évitant ainsi des doublons de connexion FTP par exemple. Si rien de concluant n'est trouvé, on instancie un nouvel handler.

### `__instanciateNewHandler`

En utilisant le schéma de l'url, elle instancie la classe `FS()` du module `nom_schema_url.py`, grâce à la bibliothèque `importlib`. Par exemple, si l'url est `ftp://host`, la méthode va récupérer une instance de la classe `FS()` du module `ftp.py`. Il lui indique ensuite les différents identifiants de connexion. Si une erreur survient lors de la connexion de l'handler, une exception est levée. Sinon celui-ci est renvoyé.

## La classe Streaming

Elle tient lieu de FIFO, et stocke les différents blocs d'information avant traitement. La taille de la FIFO et la taille d'un bloc peuvent être paramétrées.

### La méthode `add_task()`

Permet de rajouter un bloc de données supplémentaire dans le FIFO, si celle-ci est pleine, un timeout se déclenche et le déroulement du script est arrêté, si à la fin du timeout la FIFO est toujours pleine, une erreur est déclenchée.

### La méthode `read()`

Renvoie un bloc de données, si la FIFO est vide, un timeout se déclenche et le déroulement du script est arrêté, si à la fin du timeout la FIFO est toujours vide, une erreur est levée.

## Les méthodes move et copy

Celles-ci sont différentes dans leurs conceptions par rapport aux autres méthodes de la classe AbstractFS, en effet ces méthodes manipulent des fichiers et dossiers situés sur des systèmes de fichiers différents, ce qui impose d'utiliser deux manipulateurs de fichiers, un pour la source et autre pour la destination :

Avant de faire quoique ce soit on vérifie que les dossiers de sources et de destination

```
1  streaming = Streaming(FIFO_size, blocksize)
2
3  tget = Thread(target=handler_src.get, args=(src["path"], streaming, error_get))
4  tput = Thread(target=handler_dst.put, args=(dst["path"], streaming, error_put))
5
6  tget.start()
7  tput.start()
8
9  tput.join()
10 tget.join()
```

On génère tout d'abord une FIFO, puis deux threads l'un lançant méthode get de la src l'autre la méthode put de la destination, possédant une même référence sur une FIFO. L'idée première était d'utiliser un objet de type Stream qui aurait fait le buffer entre la fonction get qui aurait stocké l'intégralité des données binaire dans le stream donc en RAM!!! Puis cet objet aurait été donné à la méthode put qui aurait écrit les données.

Cette piste a vite été abandonnée pour des raisons évidentes de consommation de mémoire. Le choix s'est donc porté vers une structure producteur-consommateur, afin de réaliser un gain en performance d'utilisation mémoire. Or ceci a un coup en vitesse de transfert entre la source et la destination, qui je le rappelle, peut-être être situé entre la boucle locale. Il a donc fallu ajuster la taille des blocs et la taille de la FIFO afin d'obtenir les meilleures performances possibles.

Ceci grâce à l'utilitaire cProfile de python qui permet de logger le temps passé dans les différentes méthodes d'un Thread. Nous remarquons ici que la méthode get passe énormément de temps à attendre la possibilité de pouvoir ajouter un bloc dans la file d'attente, ceci signifie que le consommateur (put), n'est pas assez rapide dans le dépilage (contrainte réseau, écriture sur le disque, etc...)

Une fois les paramètres de la FIFO optimisés, la totalité du temps est consacrée à la lecture des données, ce qui est le fonctionnement voulu et attendu.



# Conclusion

Pour conclure, avec  $\text{\LaTeX}$  on obtient un rendu impeccable mais il faut s'investir pour le prendre en main.

## Références

[REF] auteur. *titre*. édition, année.

[LPP] Rolland. *LaTeX par la pratique*. O'Reilly, 1999.