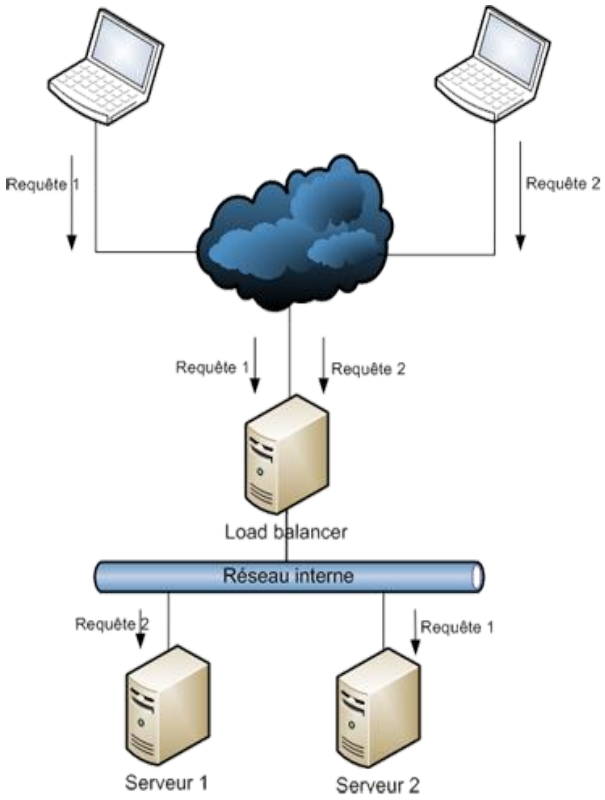
I. Introduction

Suite à la formation JBoss plusieurs tests s'offrent à moi mais il y en a un qui me tient plus à cœur : le **load balancing**. Pourquoi cela? Après tout en tant qu'Architecte logiciel certains considèrent que notre problématique et de savoir que cela existe. La technique restant le domaine de prédilections des administrateurs et architectes SI. Personnellement ce n'est pas ma conception. Même si je ne prétend pas avoir les mêmes compétences, j'aime savoir de quoi je parle.

Tout d'abord une petite définition du load balancing : le « **load balancing** » ou « **répartition de charge** » est une technique utilisée en informatique pour distribuer un travail entre plusieurs processus, ordinateurs, disques ou autres ressources (source wikipedia).



Le load balancing peut se faire de différentes manières :

- **matériel** : solution permettant de résister à la plus forte charge. Elle aura des problèmes en cas de répartition sur un protocole haut niveau utilisant les sessions. Cette solution à un coût élevé.
- **logiciel** : permet de facilement gérer des répartitions de charge avec respect de session et un coup faible. Plus lent que le hard, elle dépend du serveur qui l'héberge pour définir ses performances.

Pour ce tutoriel, nous allons nous intéresser uniquement à la partie logicielle qui est plus simple à mettre en œuvre. Nous utiliserons pour se faire la configuration la plus courante dans le monde java :

- deux serveurs d'application JBoss (avec des configurations différentes mais une application

commune. Habituellement les deux serveurs seront totalement iso),
- un serveur apache 2 sur lequel nous activerons le module mod_jk.

II. Préparation

Je vous conseille de commencer par créer un répertoire où vous installerez les différents serveurs nécessaires.

A. Installation Jboss

Pour ce tutoriel, j'ai utilisée une version 5.1.0.GA de Jboss que vous pouvez télécharger sur le site officiel de Jboss http://jboss.org/jbossas/downloads/.

Une fois l'archive obtenue, decompilez là dans le répertoire précédemment créé. Vous devez alors avoir un répertoire « *jboss-5.1.0.GA* » qui contient un répertoire bin. Dans ce répertoire bin, double cliquez sur le fichier run.bat sous windows (sous unix lancer le run.sh).

Si tout se passe bien vous ne devez pas avoir d'erreur dans les traces et la dernière ligne doit se finir par :

Started in 58s:376ms

Si ce n'est pas le cas reportez vous à la documentation très complète de JBoss[1].

Une fois cette étape franchie, il nous reste à dupliquer l'installation pour avoir un deuxième serveur JBoss.

Pour se faire arrêter le serveur en cours (fermer la fenêtre de commande), puis recopier entièrement le répertoire « *jboss-5.1.0.GA* » en un répertoire « *jboss-5.1.0.GA-server2* ».

Si maintenant vous lancez les deux serveurs vous allez avoir une erreur du style :

java.lang.Exception: Port 8083 already in use

Cette erreur est normale! Effectivement vous lancez en réalité deux serveurs avec exactement la même configuration. Il va nous falloir sur le deuxième serveur changer les ports d'écoutes. Pour faciliter l'opération et éviter toute erreur nous ajouterons 100 à tout les numéros de ports trouvés.

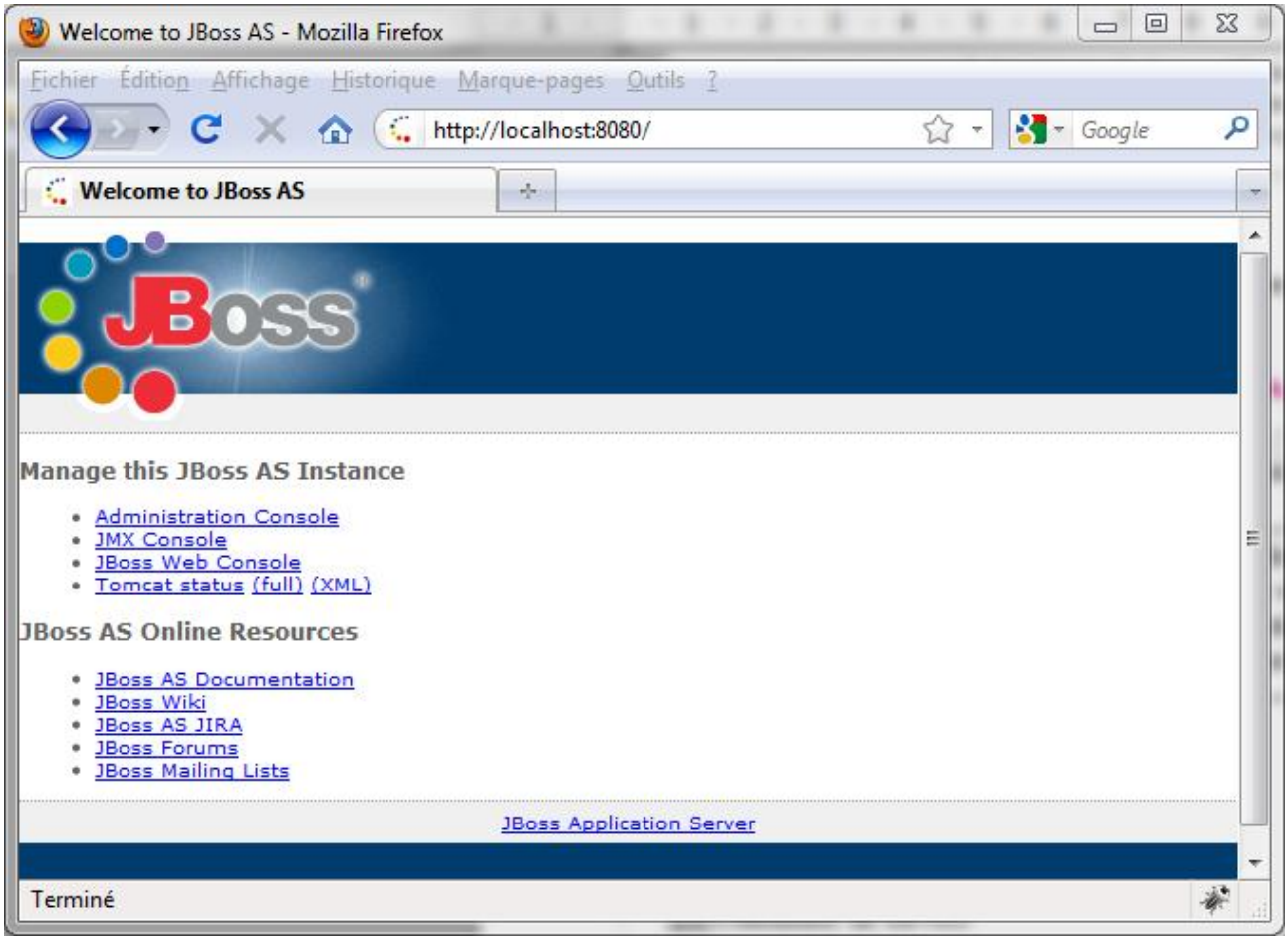
Pour se faire dans « ***jboss-5.1.0.GA-server2/server/deploy/*** » supprimer tout les répertoires sauf « **web** ». Renommer « **web** » en « **default** ». Suite à cela chercher le fichier « ***jboss-5.1.0.GA-server2/server/default/deploy/jbossweb.sar/server.xml*** » et incrémenter tout les numéros de port de 100.

Relancer les deux serveurs et ouvrer un navigateur. Vous avez maintenant accès aux urls :

http://localhost:8080

http://localhost:8180

1 http://www.jboss.org/jbossas/docs/5-x/gettingstarted.html



Accéder pour chacune des urls précédentes à la console JMX pour vérifier le bon fonctionnement des serveurs (login : **admin** et pass : **admin**).

Nous avons donc maintenant deux serveurs d'application actifs. Il reste à mettre en place notre load balancer, dans notre cas un serveur web.

B. Installation d'apache

Apache s'avèrera plus simple à installer vu que nous n'avons besoin que d'un serveur de ce type. Pour des problèmes de compatibilité avec le module mod_jk que nous allons utiliser, je vous conseille de télécharger la dernière version disponible d'apache 2.0 (dans mon cas la 2.0.63) que vous pouvez télécharger sur le site d'apache http://httpd.apache.org/download.cgi.

Exécuter l'installation et vérifier en lançant le serveur que tout s'est bien passé.

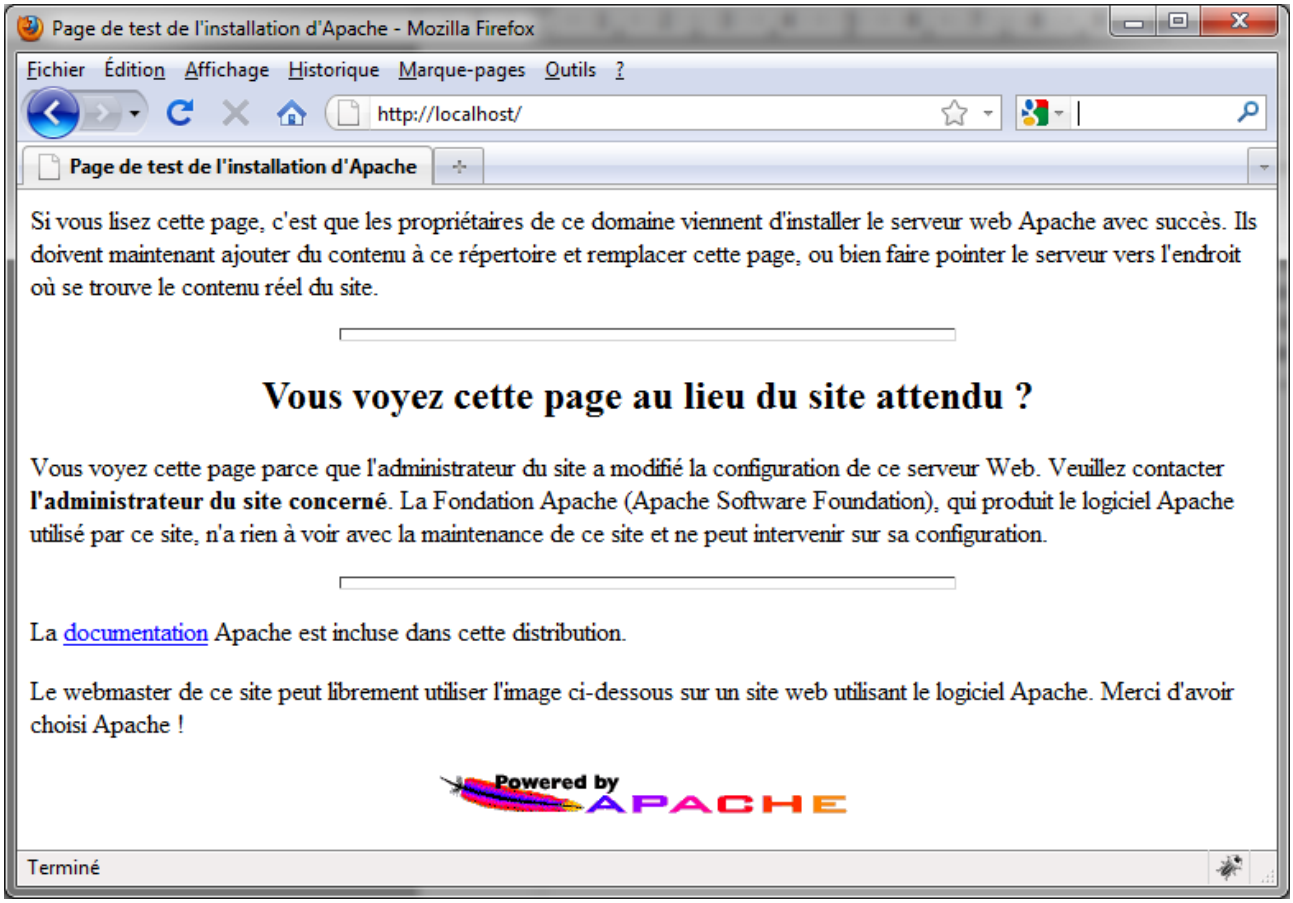
Remarque 1 : La relance du serveur apache se fait :

- sous Windows à travers un service. Ainsi l'installeur ajoute Apache en service et le lance automatiquement. Si vous voulez être sûr de la prise en compte d'une nouvelle configuration, relancer le en cliquant droit sur l'icône Apache dans le menu démarrer puis cliquer sur « **open service** ». Une fenêtre s'ouvre avec la liste des services où Apache est

sélectionné. Il vous suffit alors de cliquer sur « **redémarrer** ».

- Sous unix, aller dans /etc/init.d/ et lancer « **apache2 restart** ».

Dans votre navigateur allez alors sur http://localhost/, vous devez avoir un écran comme celui-ci:



Remarque 2 : Sous Windows attention! Il vous faudra désactiver le serveur IIS qui utilise le port 80 sinon Apache ne pourra se lancer correctement. Pour se faire aller dans « **menu démarrer>panneau de configuration>outils d'administration** » et lancer « **gestionnaire de service internet** » là vous aurez tout le loisir d'arrêter complétement IIS. Relancer à nouveau le serveur Apache et là tout est ok.

Nous avons maintenant 3 serveurs indépendants et nous allons pouvoir entrer dans le cœur du sujet en commençant par une simple répartition de charge.

III. Configuration du load balancer

Nous allons commencer par ajouter le module nécessaire à Apache afin de dialoguer avec les serveurs d'applications. Nous concernant, il existe plusieurs modules pour se connecter à JBoss :

- mod_proxy

- mod_cluster
- mod_jk

Nous utiliserons mod_jk qui est le plus répandu. Pour se faire, rendez-vous sur http://tomcat.apache.org/download-connectors.cgi et télécharger la version binaire qui correspond à votre plateforme. Copier le fichier « **mod_jk***.so » dans le répertoire « **apache2/modules** » et renommer le en « **mod_jk.so** ».

Ici nous ne toucherons pas aux serveurs d'application ceux ci étant considérés comme fonctionnels et à l'écoute de connexions avec le protocole AJP13.

Nous allons commencer par déclarer ces serveurs d'applications à notre load balancer (apache 2). Pour ce faire rendez vous dans « **apache2*/conf/*** » et créer un fichier « **workers.properties** » dans lequel nous allons inscrire les serveurs de la manière suivante :

workers.properties

```
# Define 1 real worker using ajp13
worker.list=loadbalancer,status

# Set properties for worker1 (ajp13)
worker.worker1.type=ajp13
worker.worker1.host=localhost
worker.worker1.port=8009
worker.worker1.lbfactor=1
worker.worker1.connection_pool_size=10

# Set properties for worker2 (ajp13)
worker.worker2.type=ajp13
worker.worker2.host=localhost
worker.worker2.port=8109
worker.worker2.lbfactor=1
worker.worker2.connection_pool_size=10

#fonctionnement de l'equilibrage de charge
worker.loadbalancer.type=lb
worker.loadbalancer.balance_workers=worker1,worker2
worker.loadbalancer.sticky_session=True

worker.status.type=status
```

Si on analyse ce fichier, on peut voir plusieurs zones :

- une zone bleu qui nous permet de définir les workers qui seront visibles de l'extérieur. Ici le seul à nous intéresser est « **loadbalancer** ».
- En vert et rouge, nous voyons la définition de connexion avec les deux serveurs d'application. On voit que les deux connexions sont de type **AJP13** (standard de communication Apache-Tomcat). Les deux serveurs sont hébergés sur la machine d'où le localhost, par contre on peut voir que les ports de connexions sont différents. Ici nous avons réparti de manière égale les requêtes entre les serveurs car nous avons mis le même **lbfactor**. On voit aussi que l'on prévoit une dizaine de connexions par serveurs qui

seront ouvertes au fur et à mesure.

- En jaune enfin le vrai loadbalancer. On le reconnaît au « **type=lb** ». Ensuite on indique tout les serveurs à prendre en compte. Enfin la propriété « **sticky_session** » indique si le load balancing se fait avec affinité de session.

Remarque 3 : L'affinité de session indique que si une requête est retransmises sur le serveur 1 pour un client toute les autres requêtes appartenant à la même session de ce client seront retransmise sur le serveur 1.

Il nous suffit maintenant de déclarer l'utilisation du module mod_jk à apache ainsi que le fichier que nous venons de créer.

Pour cela nous allons modifier le fichier httpd.conf du même répertoire de la manière suivante :

Httpd.conf

```
…
LoadModule jk_module modules/mod_jk.so
…
# Where to find workers.properties
# Update this path to match your conf directory location (put workers.properties next to httpd.conf)
JkWorkersFile conf/workers.properties

# Where to put jk shared memory
# Update this path to match your local state directory or logs directory
JkShmFile     logs/mod_jk.shm

# Where to put jk logs
# Update this path to match your logs directory location (put mod_jk.log next to access_log)
JkLogFile     logs/mod_jk.log

# Set the jk log level [debug/error/info]
JkLogLevel    info

# Select the timestamp log format
JkLogStampFormat "[%a %b %d %H:%M:%S %Y] "

JkMount /* loadbalancer

<IfModule worker.c>
        StartServers    2
        MaxClients      150
        MinSpareThreads      25
        MaxSpareThreads      75
        ThreadsPerChild      25
        MaxRequestsPerChild          0
</IfModule>
```

La ligne bleu permet d'ajouter mod_jk à la liste des modules chargés. En jaune nous indiquons le fichier de configuration à prendre en compte. En orange les lignes de configurations de sortie du module, à ne pas toucher.

La ligne verte nous intéresse plus, elle permet d'indiquer quelles URL nous redirigeons et vers qui.

Ici toute les requêtes sont redirigés sur le worker loadbalancer. Elle seront donc au vu de notre configuration réparties équitablement entre les deux serveurs.

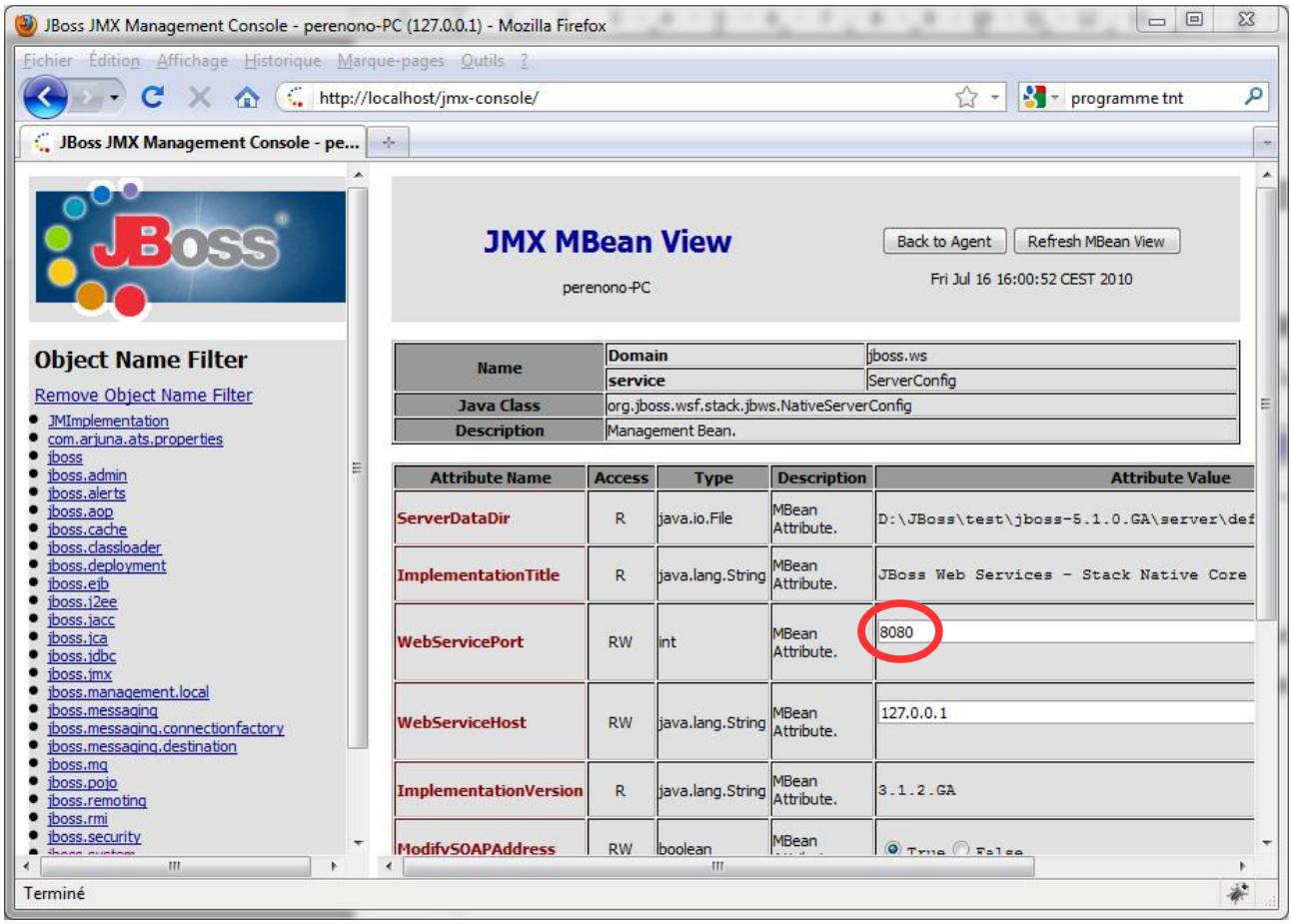
Il vous reste un dernier fichier à ajouter dans le répertoire conf pour faire un lien entre le worker status et une url. Cette page **status** permettra d'afficher l'état du connecteur mod_jk. Ceci n'est en rien obligatoire mais conseillé en mode de développement/Pré-production.

uriworkermap.properties

```
/status=status
```

A présent c'est le moment relancer le serveur apache pour prendre en compte cette nouvelle configuration (voir **remarque 1**).

Maintenant, vérifions le bon fonctionnement de notre configuration. Pour ce faire, ouvrez un navigateur et connectez vous sur http://localhost/ vous devez maintenant voir la page d'accueil de JBoss et non plus celle d'Apache. Entrez sur la console JMX comme au début et cliquez sur « **jboss.ws** » à gauche puis sur « **service=ServerConfig** ». Dans la fenêtre ainsi apparue vous voyez le port d'écoute du serveur : 8080 ou 8180.

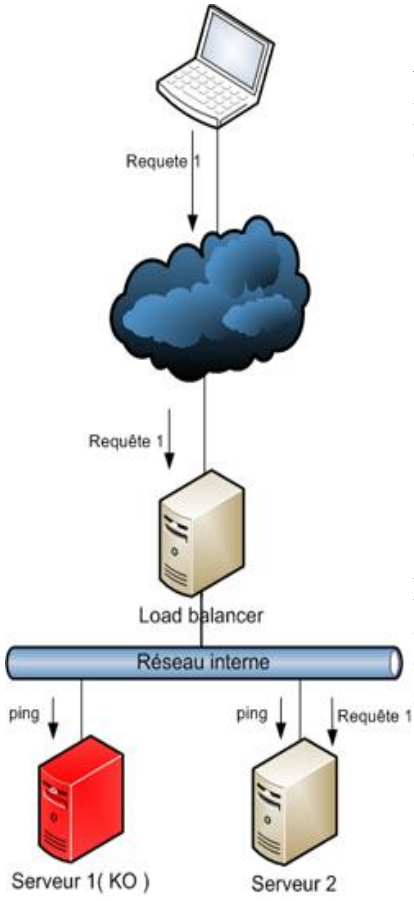


Ouvrer alors une nouvelle fenêtre avec un autre navigateur (si vous travaillez avec le même navigateur vous aurez la même session et donc accès au même serveur grâce à l'affinité de session). En reproduisant la même procédure, vous verrez que le serveur sera en écoute sur l'autre port.

IV. Continuité de service

Maintenant que nous sommes capables de repartir la charge d'un site entre plusieurs machines, on peut se demander ce qui se passerait si un des serveur venait à ne plus fonctionner. Et bien horreur rien du tout! La communication étant rompue une erreur parviendra au client ce qui vous en conviendrai n'est pas du meilleur effet.

Pour répondre à cela tout à été prévu. On peut ainsi, au cas ou apache détecte que le serveur qu'il tente d'accéder n'existe plus, lui dire de rediriger les requêtes vers un autre serveur.



Alors comment cela se passe t'il?

Et bien au moment ou apache reçoit une requête, il détermine le serveur qui doit répondre :

- Soit une connexion libre est ouverte vers celui ci et la requête est lancée,
- Soit aucune connexion n'est disponible, Apache va alors pinger la machine pour savoir si elle existe ou non. Si le ping échoue, apache regarde s'il existe une directive de redirection. Si c'est le cas, il va tenter de communiquer avec le serveur de remplacement.

Dans le cas qui nous intéresse, le serveur 1 étant en panne, apache essaie d'ouvrir une connexion sur le serveur 2. Comme la connexion peut être créer alors la requête est émise vers ce nouveau serveur.

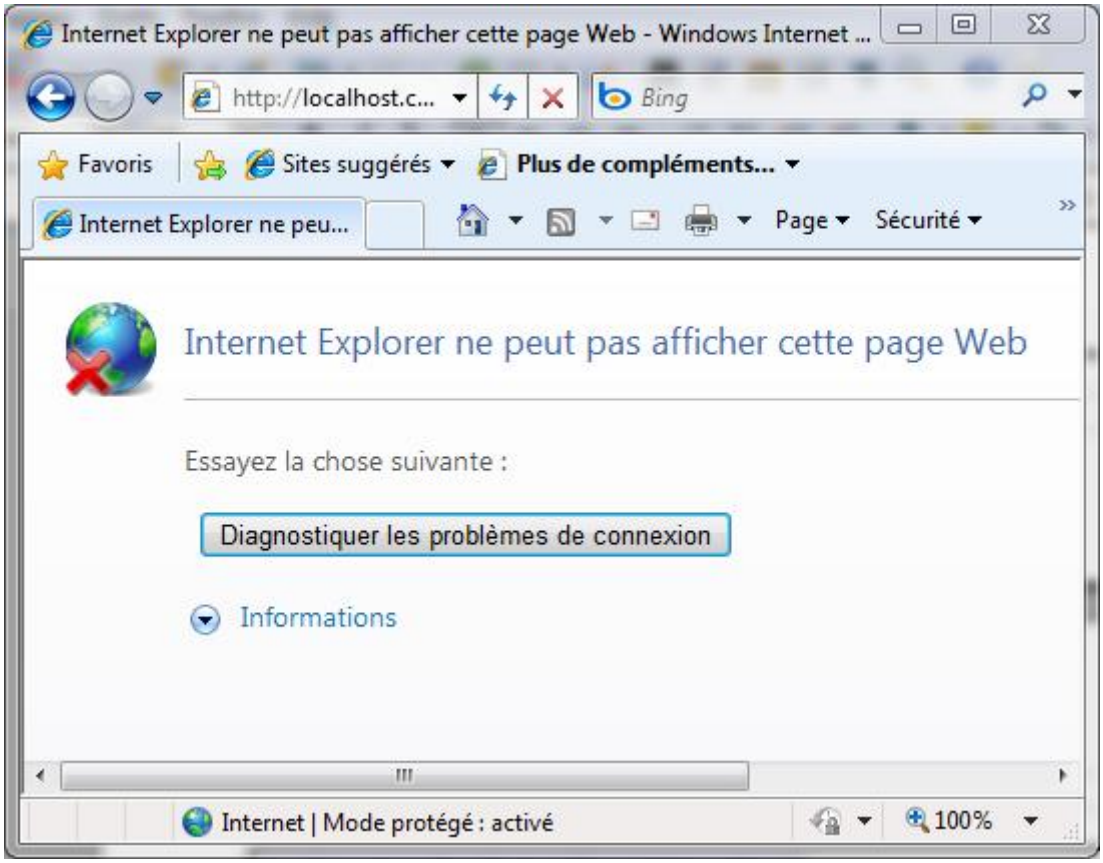
Pour ce faire, nous allons ajouter une ligne au worker1 dans notre fichier **workers.properties**.

workers.properties

```
...
# Set properties for worker1 (ajp13)
worker.worker1.type=ajp13
worker.worker1.host=localhost
worker.worker1.port=8009
worker.worker1.lbfactor=1
worker.worker1.connection_pool_size=10
# Define preferred failover node for worker1
worker.worker1.redirect=worker2
...
```

Nous venons d'indiquer que si le serveur worker1 n'est plus accessible les requêtes sont redirigées sur le worker2.

Pour tester cette configuration, arrêter le serveur1 et retenter d'accéder depuis les deux navigateurs. L'un des deux navigateurs va afficher une erreur comme suit.



Relancer apache pour qu'il prenne en compte la nouvelle configuration et refaite la manipulation. Maintenant, si vous vérifier dans la console JMX les deux navigateurs accèdent au même serveur.

Si nous nous contentons de cela, nous redirigeons toute la charge vers un serveur avec le risque qu'il ne puisse répondre à l'ensemble de la charge. Pour ce faire, on préfère ajouter un worker comme les autres mais que l'on met en attente. Ainsi il ne recevra de requête que si l'un des serveurs tombe.

workers.properties

```
...
# Set properties for worker2 (ajp13)
worker.worker3.type=ajp13
worker.worker3.host=localhost
worker.worker3.port=8209
worker.worker3.lbfactor=1
worker.worker3.connection_pool_size=10
# Disable worker3 for all requests except failover
worker.worker3.activation=disabled
...
```

Ajouter le worker3 dans la propriété `worker.loadbalancer.balance_workers` pour le rendre disponible et bien se rappeler qu'un worker disabled est un worker qui ne reçoit pas de requête directement ainsi si vous définissez uniquement 2 workers et que l'un est disabled toute les requêtes iront sur l'autre serveur.

Encore quelques optimisations et nous voici avec un site prêt à répondre à la charge d'une situation réelle.

V. Optimisation annexe

Tout à l'heure nous avions vu une zone d'optimisation des connexions entre Apache et JBoss, il est temps de mieux comprendre à quoi celle-ci sert[2].

Httpd.conf

```
…
<IfModule worker.c>
        StartServers    2
        MaxClients      150
        MinSpareThreads         25
        MaxSpareThreads         75
        ThreadsPerChild         25
        MaxRequestsPerChild             0
</IfModule>
```

Cette zone permet de définir le nombre de requêtes qui peuvent être acceptées en parallèle ainsi que le nombre de requêtes pouvant être mises en attente. Ainsi nous indiquons que 2 processus serveurs sont créées au démarrage d'apache, il n'influe que peu car Apache va au fur et à mesure des requêtes arrêter et lancer de nouveaux processus serveurs. On indique aussi que l'on pourra avoir au maximum 150 clients simultanément.

Remarque 4 : Il faut normalement compter 20Mo par client actif, on peut donc à partir de la mémoire libre définir la valeur de **MaxClients** à appliquer. Ainsi pour 1Go, on pourra accepter 50 clients parallèles.

Les valeurs **MinSpareThreads** et **MaxSpareThreads** permet d'entretenir des processus serveur de secours en fonction du nombre de requêtes.

Remarque 5 : Les valeurs **MinSpareThreads** et **MaxSpareThreads** doivent être en cohérence avec les valeurs présentes sur les serveurs d'application.

ThreadsPerChild définit le nombre de threads au sein de chaque processus enfant.
La propriété **MaxRequestPerChild** est ici désactivée, elle permet de limiter le nombre de requêtes acceptées par processus serveur. Si ce maximum est atteint le processus serveur est cloné puis arrêté.

2 Information plus complète sur http://www.linux-kheops.com/doc/redhat72/rhl-rg-fr-7.2/s1-apache-config.html

VI. Conclusion

Il ne reste plus qu'à appliquer ce que nous avons vue sur un projet concret et se frotter au réglage plus fin du serveur. Mais là, je ne me fait pas de soucis la documentation existe en nombre, particulièrement pour JBoss et Apache.