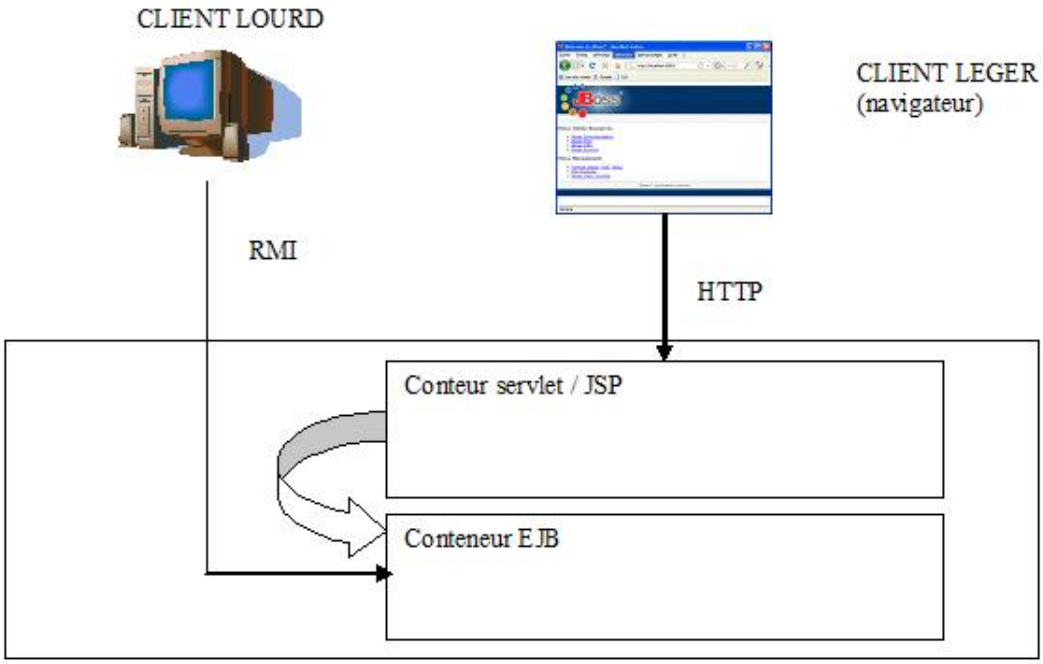
Accès aux EJB

Une fois l'authentification effectuée, le contexte de sécurité est propagé d'un conteneur à l'autre :

- du conteneur client, pour un client lourd, vers le conteneur EJB ;
- du conteneur JSP vers le conteneur EJB ;
- d'un conteneur EJB vers un autre conteneur EJB.



Au niveau de l'EJB, il est possible de gérer les autorisations d'accès aux méthodes.

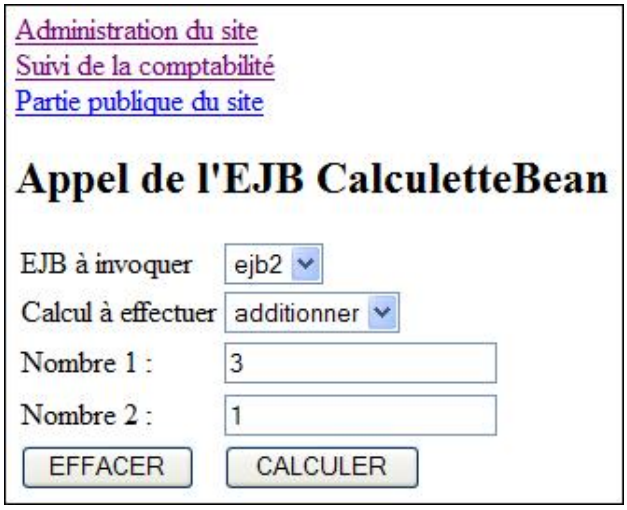
Il faut que les différents conteneurs utilisent le même contexte d'autorisation. Cela peut être vérifié en affichant les noms JNDI.

```
java:comp namespace of the Chap 8 - Sécurité Web.war application:

  +- UserTransaction[link -> UserTransaction] (class: javax.naming.LinkRef)
  +- env (class: org.jnp.interfaces.NamingContext)
  |  +- security (class: org.jnp.interfaces.NamingContext)
  |  |  +- realmMapping[link -> java:/jaas/exemple-eni]
(class: javax.naming.LinkRef)
  |  |  +- subject[link -> java:/jaas/exemple-eni/subject]
(class: javax.naming.LinkRef)
  |  |  +- securityMgr[link -> java:/jaas/exemple-eni]
(class: javax.naming.LinkRef)
  |  |  +- security-domain[link -> java:/jaas/exemple-eni]
(class: javax.naming.LinkRef)

Ejb Module: Chap 8 - EJB2.jar


java:comp namespace of the Calculette bean:

  +- env (class: org.jnp.interfaces.NamingContext)
  |  +- security (class: org.jnp.interfaces.NamingContext)
  |  |  +- subject[link -> java:/jaas/exemple-eni/subject]
(class: javax.naming.LinkRef)
  |  |  +- security-domain[link -> java:/jaas/exemple-eni]
(class: javax.naming.LinkRef)
```

```
java: Namespace

...
  +- ConnectionFactory (class: org.jboss.mq.SpyConnectionFactory)
  +- jaas (class: javax.naming.Context)
  |  +- exemple-eni (class: org.jboss.security.plugins.SecurityDomainContext)
  |  +- other (class: org.jboss.security.plugins.SecurityDomainContext)
  |  +- HsqlDbRealm (class: org.jboss.security.plugins.SecurityDomainContext)
  |  +- jbossmq (class: org.jboss.security.plugins.SecurityDomainContext)
  |  +- JmsXARealm (class: org.jboss.security.plugins.SecurityDomainContext)
```

1. Gestion des autorisations sur les EJB 2

Les extraits de code et de fichiers xml sont issus du projet "Chap 8 -EJB2". Le projet Web "Chap 8 - Sécurité Web" permet de tester l'EJB. Vous devez déployer les deux projets dans votre serveur. L'URL de test est : http://localhost:8080/secu. Pour être authentifié, vous devez d'abord cliquer sur le lien **Administration du site**, puis choisir **EJB2** dans la liste déroulante.



Descripteur de déploiement ejb-jar.xml

```
<ejb-jar>

   <enterprise-beans>

      <!-- Session Beans -->
      <session id="Session_Calculette">
         <display-name>Calculette</display-name>
         <ejb-name>Calculette</ejb-name>
         <home>fr.eni.editions.ejb2.secu.CalculetteHome</home>
         <remote>fr.eni.editions.ejb2.secu.Calculette</remote>
         <local-home>
           fr.eni.editions.ejb2.secu.CalculetteLocalHome
         </local-home>
         <local>fr.eni.editions.ejb2.secu.CalculetteLocal</local>
         <ejb-class>
           fr.eni.editions.ejb2.secu.CalculetteSession
         </ejb-class>
         <session-type>Stateless</session-type>
         <transaction-type>Container</transaction-type>

         <security-role-ref>
            <role-name>admin</role-name>
            <role-link>admin</role-link>
         </security-role-ref>

      </session>
```

```
   </enterprise-beans>

...
<assembly-descriptor id="AssemblyDescriptor_1">

      <security-role>
         <description>
           <![CDATA[Profil administrateur]]>
         </description>
         <role-name>admin</role-name>
      </security-role>

...

</ejb-jar>
```

Ce premier extrait du fichier *ejb-jar.xml* nous permet de déclarer les références aux rôles de sécurité. L'élément `<security-role-ref>` contient le mapping entre :

- `<role-name>` qui est le nom du rôle qui est utilisé dans l'EJB, en gestion programmée de l'authentification ;
- `<role-link>` correspond à un des rôles définis dans la balise `<security-role>`.

L'élément `<assembly-descriptor>` contient la définition des rôles qui seront utilisés par le composant. Les rôles sont définis dans la balise `<security-role>` qui contient :

- `<description>` correspond à une description éventuelle du rôle ;
- `<role-name>` correspond à un rôle de sécurité.

Ensuite, une autorisation est mise en place pour les méthodes de l'EJB au sein de l'élément `<assembly-descriptor>`. Le mapping entre les autorisations de méthodes et les rôles de sécurité est effectué dans la balise `<method-permission>`.

```
<ejb-jar>

...

   <method-permission id="MethodPermission_4">
      <description>
        <![CDATA[description not supported yet by ejbdoclet]]>
      </description>
      <unchecked/>
      <method id="MethodElement_4">
         <description>
           <![CDATA[<!-- begin-xdoclet-definition -->]]>
         </description>
         <ejb-name>Calculette</ejb-name>
         <method-intf>Remote</method-intf>
         <method-name>additionner</method-name>
         <method-params>
            <method-param>int</method-param>
            <method-param>int</method-param>
         </method-params>
      </method>
   </method-permission>
   <method-permission id="MethodPermission_6">
      <description>
        <![CDATA[description not supported yet by ejbdoclet]]>
      </description>
         <role-name>admin</role-name>
      <method id="MethodElement_6">
         <description><![CDATA[<!-- begin-xdoclet-definition -->]]>
</description>
         <ejb-name>Calculette</ejb-name>
         <method-intf>Remote</method-intf>
```

```
         <method-name>soustraire</method-name>
         <method-params>
            <method-param>int</method-param>
            <method-param>int</method-param>
         </method-params>
      </method>
   </method-permission>

...

</ejb-jar>
```

L'élément `<method-permission>` comporte les éléments fils :

- `<description>` contient un texte optionnel décrivant le mapping ;
- `<role-name>` rôle de sécurité permis pour utiliser la méthode. Cette balise peut être présente plusieurs fois, ou être absente ;
- `<unchecked>` précise que la méthode n'est pas liée à un rôle particulier ;
- `<method-name>` contient le nom de la méthode. Peut aussi contenir un nom générique `type *` pour toutes les méthodes de l'EJB, ou `get*` pour tous les getters ;
- `<ejb-name>` nom de l'EJB ;
- `<method-inf>` distingue le type d'interface pour la méthode : `Remote`, `Home`, `Local` ou `LocalHome` ;
- `<method-params>` contient la liste des types des paramètres de la méthode, ce qui permet de préciser une signature particulière en cas de surcharge de la méthode.

Descripteur de déploiement jboss.xml

Le domaine de sécurité a été ajouté au fichier *jboss.xml*.

```
<jboss>

...

 <security-domain>java:/jaas/exemple-eni</security-domain>

   <enterprise-beans>

      <session>
         <ejb-name>Calculette</ejb-name>
         <jndi-name>ejb2/remote/Calculette</jndi-name>
         <local-jndi-name>ejb2/local/Calculette</local-jndi-name>

        <method-attributes>
        </method-attributes>
      </session>

   </enterprise-beans>

...

</jboss>
```

Dans l'exemple, l'EJB `CalculetteBean` possède deux méthodes : `additionner(...)` et `soustraire(...)`. Pour accéder à ces méthodes, il faut que l'utilisateur soit authentifié. La méthode `additionner(...)` peut être invoquée quel que soit l'utilisateur, tandis que la méthode `soustraire(...)` n'est utilisable que par un utilisateur ayant un profil `"admin"`. Le conteneur d'EJB lève une exception si les conditions de sécurité ne sont pas respectées lors de l'invocation d'une méthode, y compris sur les méthodes du cycle de vie.

2. Gestion des autorisations sur les EJB 3

Les extraits de code et de fichiers xml sont issus du projet "Chap 8 -EJB3". Le projet Web "Chap 8 - Sécurité Web" permet de tester l'EJB. Vous devez déployer les deux projets dans votre serveur. L'URL de test est : http://localhost:8080/secu. Pour être authentifié, vous devez d'abord cliquer sur le lien **Administration du site**, puis choisir **EJB3** dans la liste déroulante.

```
...

@Local
@Stateless
@LocalBinding(jndiBinding="ejb3/local/Calculette")
@SecurityDomain("exemple-eni")
public class CalculetteBean implements Calculette
{
    @Unchecked
    public int additionner(int a, int b)
    {
        return a+b;
    }

    @RolesAllowed({"admin"})
    public int soustraire(int a, int b)
    {
        return a-b;
    }
}
```

Nous retrouvons les configurations équivalentes aux fichiers de déploiement des EJB 2, mais exprimées par les annotations. Dans notre exemple le fichier *ejb-jar.xml* n'est pas présent.

La description de configuration s'en trouve allégée, il n'y a plus besoin, entre autres, de définir les signatures.

Les annotations utilisées ici sont :

- `@securityDomain` : contient le domaine de sécurité. Il s'agit d'une annotation JBoss facultative. Si elle n'est pas présente, le fichier *jboss.xml* doit déclarer le domaine de sécurité dans l'élément `<security-domain>`. Attention, la valeur de l'annotation n'est pas le nom JNDI mais uniquement le nom du domaine. Le nom JNDI du domaine est : `java:/jaas/exemple-eni`, nous utilisons donc uniquement `exemple-eni`.

- `@Unchecked` : correspond à l'élément `<unchecked>` du fichier de déploiement *ejb-jar.xml*. La méthode n'est pas liée à l'authentification sur un rôle particulier.

- `@RolesAllowed` : correspond à l'élément `<role-name>` du fichier *ejb-jar.xml*. La valeur de l'annotation contient donc, les rôles autorisés à utiliser la méthode. La valeur de cette annotation est un tableau de String, d'où l'utilisation des accolades pour le passage de la valeur. Si plusieurs rôles sont définis, ils sont dans l'accolade, séparés par des virgules, par exemple `@RoleAllowed({"admin","gestion"})`.

Instancier plusieurs serveurs sur une même machine

Il peut être nécessaire, ou pratique, de faire exécuter plusieurs instances de JBoss sur une même machine. Dans le cadre de tests, nous n'avons pas forcément le nombre de machines nécessaires sous la main. Cette solution peut donc s'avérer très pratique, à condition de disposer des ressources système nécessaires en mémoire et puissance processeur.

Des tests de mise en cluster peuvent ainsi être effectués. De même, des test sur des instances JBoss utilisant des versions de JVM différentes peuvent être pratiqués. Dans ce dernier cas, il ne peut pas y avoir de cluster mis en place.

Plusieurs instances de JBoss ne peuvent pas être exécutées sur une même machine sans qu'apparaissent des conflits, car il ne peut pas y avoir deux applications qui écoutent sur une même adresse IP, et sur un même port. Il faut donc changer les ports d'écoute de chaque instance.

Cette opération est facilitée par l'existence d'un service `ServiceBinding`, normalement non activé dans les configurations serveur. Un fichier de configuration exemple permet de facilement accroître le nombre d'instances.

Pour réaliser cette opération, nous devons :

- afficher le fichier de configuration des liaisons entre chaque serveur et ses ports d'écoute ;
- créer par copie autant de serveurs que nécessaire ;
- modifier le fichier *jboss-service.xml* de chacun des serveurs.

1. Fichier sample-bindings.xml

Le fichier *sample-bindings.xml* se trouve dans le répertoire *<repertoire installation>docs\examples\binding-manager*.

La configuration des ports d'écoute de chaque instance de serveur est décrite dans un élément <server> contenant le nom de la configuration.

```
<service-bindings>

   <!-- ******************************************************** -->
   <!-- *                        ports-default                 * -->
   <!-- ******************************************************** -->
   <server name="ports-default">

...
```

Le fichier fourni avec la distribution JBoss comporte 4 configurations différentes, nommées :

- `ports-default` ;
- `ports-01` ;
- `ports-02` ;
- `ports-03` ;

Les ports d'écoute pour JNDI et Tomcat sont, entre autres, définis dans chacune de ces configurations :

Ainsi, pour la configuration `ports-default`, le port d'écoute du service de nommage est 1099 :

```
<service-config name="jboss:service=Naming"
 delegateClass="org.jboss.services.binding.AttributeMappingDelegate">
 <delegate-config portName="Port" hostName="BindAddress">
    <attribute name="RmiPort">1098</attribute>
 </delegate-config>
 <binding port="1099" host="${jboss.bind.address}"/>
</service-confi
```

Et le port d'écoute HTTP est 8080 :

```
<service-config name="jboss.mq:service=InvocationLayer,type=HTTP"
   delegateClass="org.jboss.services.binding.AttributeMappingDelegate" >
   <delegate-config portName="URLPort"/>
   <binding port="8080"/>
</service-config>
```

Il suffit d'ajouter 1000 au port d'écoute par défaut, pour avoir les ports d'écoute des autres configurations :

Configuration	Service de nommage	Écoute HTTP
`ports-default`	1099	8080
`ports-01`	1199	8180
`ports-02`	1299	8280
`ports-03`	1399	8380

2. Copie d'une configuration de base

Nous allons partir de la configuration `default` pour créer `default-01` et `default-02`.

Pour ce faire, il suffit de copier le répertoire *default* en le renommant. Vous devez maintenant avoir sous le répertoire server de votre installation JBoss, les configurations serveurs suivantes :

- `all`
- `default`
- `default-01`
- `default-02`
- `minimal`.

3. Changement des configurations serveurs

Nous allons changer les configurations des serveurs default-01 et default-02 pour qu'ils puissent utiliser le service `ServiceBinding`.

- Ouvrez le fichier *jboss-service.xml* de default-01 et cherchez la section contenant la référence au service `ServiceBinding`.

```
<mbean code="org.jboss.services.binding.ServiceBindingManager"
 name="jboss.system:service=ServiceBindingManager">
 <attribute name="ServerName">ports-01</attribute>
 <attribute name="StoreURL">${jboss.home.url}/docs/examples/binding-
manager/sample-bindings.xml</attribute>
 <attribute name="StoreFactoryClassName">
   org.jboss.services.binding.XMLServicesStoreFactory
 </attribute>
</mbean>
```

- Décommentez le bloc déclarant le Mbean, en prenant soin de commenter le texte précédant l'élément déclarant le MBean.

  Par défaut le nom du serveur est `ports-01`, nous allons le laisser à cette valeur pour la configuration `default-02`. Cette valeur fait référence à la déclaration du serveur dans le fichier *binding-manager.xml* vu précédemment.

Notez qu'un élément `<attribute name="StoreURL">` référencie le fichier de configuration des ports.

- Opérez la même manipulation sur le fichier *jboss-service.xml* de la configuration `default-02`, en renommant le serveur en `ports-02`. Vous devriez avoir la configuration suivante pour ce serveur :

```
<mbean code="org.jboss.services.binding.ServiceBindingManager"
 name="jboss.system:service=ServiceBindingManager">
 <attribute name="ServerName">ports-02</attribute>
 <attribute name="StoreURL">${jboss.home.url}/docs/examples/binding-
manager/sample-bindings.xml</attribute>
 <attribute name="StoreFactoryClassName">
   org.jboss.services.binding.XMLServicesStoreFactory
 </attribute>
</mbean>
```

4. Démarrage des serveurs

Ceci est devenu classique pour vous, maintenant. Les serveurs sont démarrés dans des consoles différentes, avec les commandes suivantes : `run -c default-01` pour la première configuration et `run -c default-02` pour la seconde configuration.

Vous ne devez pas avoir d'erreur lors du démarrage des serveurs. Si c'est le cas, il y a des ports communs, vérifiez alors les points précédents.

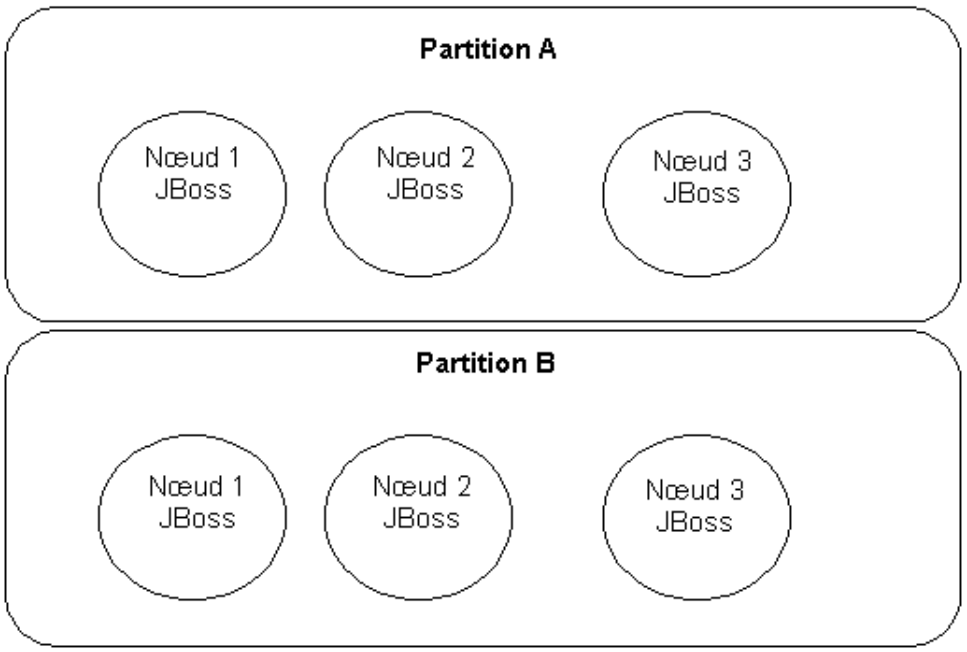
Si les serveurs ont démarré sans erreur, vous pouvez atteindre les pages d'administration de JBoss par les URLs :

- http://localhost:8180/ pour le serveur `default-01`
- http://localhost:8280/ pour le serveur `default-02`.

Mise en cluster de plusieurs serveurs

La mise en cluster de plusieurs instances de JBoss permet à une application d'être exécutée sur plusieurs serveurs. La charge peut alors être distribuée entre les serveurs, et si un des serveurs ne fonctionne plus, l'application est toujours accessible par l'intermédiaire des autres serveurs.

Un cluster est un ensemble de nœuds, un nœud étant une instance particulière de JBoss. Pour monter un cluster, il suffit donc de grouper plusieurs instances de JBoss, ce groupement est appelé une partition. Plusieurs partitions peuvent exister sur un même réseau, chaque partition ayant un nom unique au sein du réseau.



La configuration serveur `all`est configurée pour fonctionner en mode serveur. Dans cette partie dédiée à la mise en cluster de JBoss, nous utiliserons cette configuration, donc tous les fichiers et répertoires décrits ci-après seront dans le répertoire du serveur `all`.

Le nom de la partition est configuré dans le fichier *cluster-service.xml* du répertoire *deploy*.

```
   <mbean code="org.jboss.ha.framework.server.ClusterPartition"
      name="jboss:service=${jboss.partition.name:DefaultPartition}">

      <!-- Name of the partition being built -->
      <attribute name=
"PartitionName">${jboss.partition.name:DefaultPartition}</attribute>

...
```

Le nom de la partition peut être passé en ligne de commande, lors du démarrage du serveur.

Sous Windows :

```
run -c all -Djboss.partition.name=partition1
```

Sous Linux :

```
./run.sh -c all -Djboss.partition.name=partition1
```

Les fonctionnalités mises en œuvre par JBoss permettent :

- la découverte automatique des nœuds d'une même partition ;
- la reprise automatique en cas de panne d'un nœud ;
- la répartition de charge pour les services JNDI et RMI ;

- la réplication de l'arborescence JNDI au travers du cluster ;
- la distribution des applications sur tous les nœuds du cluster lorsque l'application est déposée dans le répertoire *farm ;*
- l'exécution unique d'une application au travers du cluster par le répertoire *deploy-hasingleton*.

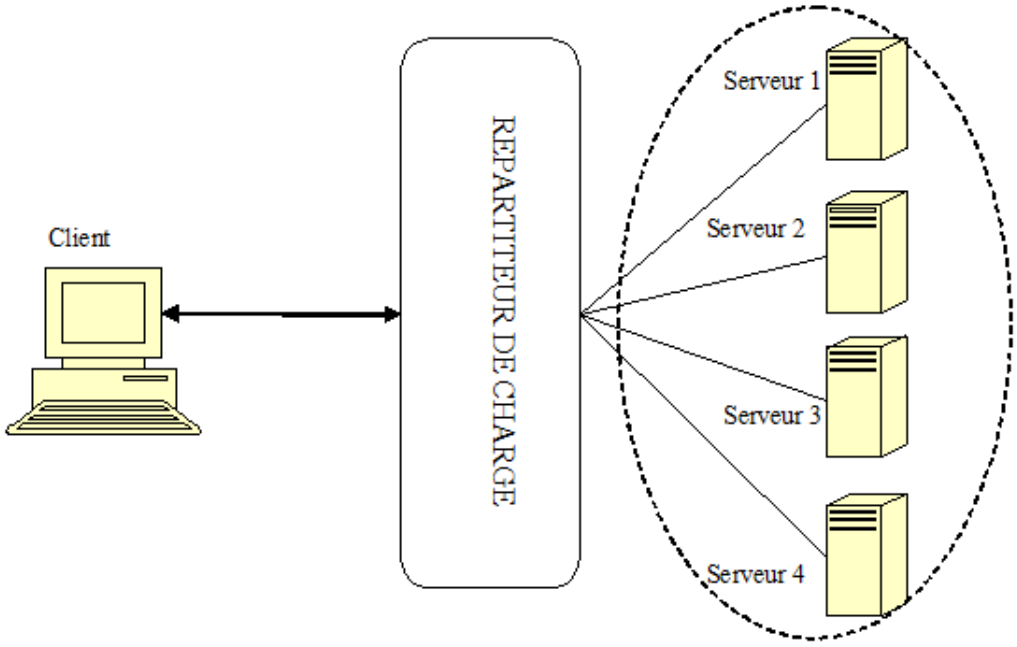
La configuration `all` possède deux répertoires supplémentaires :

- *farm* qui est le répertoire de déploiement des applications qui devront être répliquées sur les autres nœuds du cluster. Une application déposée dans ce répertoire va automatiquement être dupliquée dans les répertoires *farm* des autres nœuds.
- *deploy-hasingleton* qui est le répertoire des applications qui ne doivent pas être dupliquées sur les autres nœuds du cluster. L'application déposée dans ce répertoire sera accessible par les autres nœuds du cluster, sans que l'application soit dupliquée. Attention dans ce cas aux échanges sur le réseau, et à la parte de la fonction de contournement des pannes (fail-over), si le nœud qui contient l'application singleton tombe en panne.

1. La répartition de charge

La répartition de charge (load balancing) consiste à interposer entre le client et le cluster, un répartiteur qui va répartir les demandes des clients vers les nœuds du cluster, en fonction de leur disponibilité et de leur taux d'occupation. Ce répartiteur peut être logiciel ou matériel. Différentes stratégies de répartition de charge peuvent être mises en œuvre. JBoss intègre les stratégies suivantes :

- Round-Robin : chaque appel est relayé à un nouveau serveur. Le premier serveur étant sélectionné aléatoirement.
- First Available : un des serveurs est élu comme cible principale et utilisé pour tous les appels. Lorsque la liste des serveurs change, un nouveau serveur est élu.



Attention, la répartition de charge ne tient pas forcément compte du suivi des sessions HTTP. Ce qui signifie qu'il faut un mécanisme supplémentaire pour qu'un client HTTP puisse toujours cibler le même nœud, c'est la notion d'affinité de session, qui sera abordée plus loin.

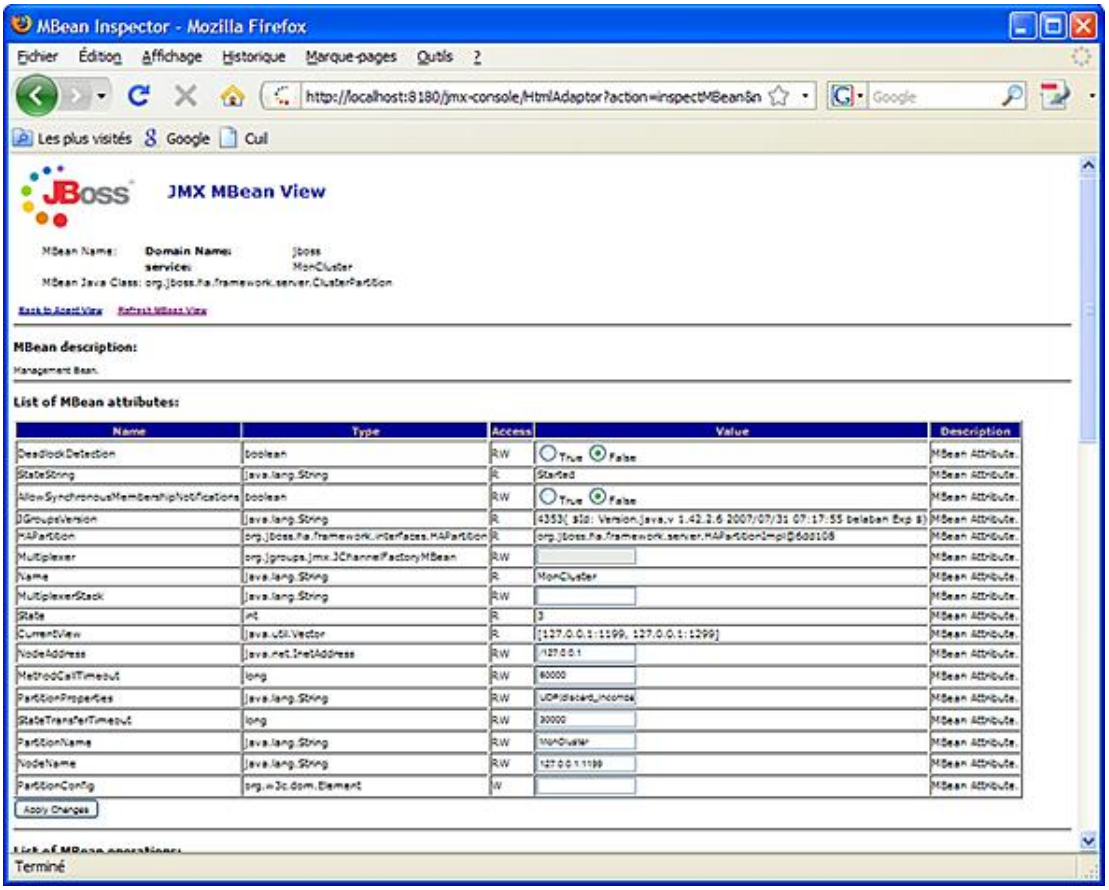
2. Lancement du cluster

Pour effectuer les tests suivants, nous devons lancer plusieurs serveurs en configuration `all`. Pour ce faire, vous pouvez :

- lancer chaque serveur sur des ordinateurs différents configurés sur un même réseau ;
- si vous ne disposez pas de plusieurs ordinateurs vous pouvez lancer plusieurs serveurs sur une machine, comme nous l'avons vu précédemment ;
- vous pouvez aussi utiliser des outils de virtualisation, comme Sun xVM VirtualBox.

Vous pouvez suivre l'évolution des nœuds de votre cluster sur la console du serveur qui a été lancé en premier.

Vous pouvez visualiser les différents nœuds du cluster en interrogeant la console JMX d'un des nœuds et en sélectionnant le service JBoss dont le service porte le nom de votre cluster.



Vous pouvez retrouver sur cet écran les adresses IP et les numéros de port d'écoute des différents serveurs du cluster nommé "MonCluster".

3. Déploiement d'application

Le déploiement est assuré par le service JBoss nommé `farm`. Il fonctionne comme le service de déploiement de base, en gérant le déploiement sur les machines du cluster.

Il suffit de déposer dans le répertoire farm une archive pour qu'elle soit déployée et copiée dans le répertoire farm de chaque nœud du cluster.

Utilisez une des archives que nous avons créées lors des précédents tests et copiez-la dans le répertoire farm d'un des serveurs. Vous pouvez vérifier que cette archive a bien été prise en compte dans la console des différents serveurs, et que l'archive a bien été recopiée automatiquement dans chaque répertoire *farm* des autres nœuds du cluster.

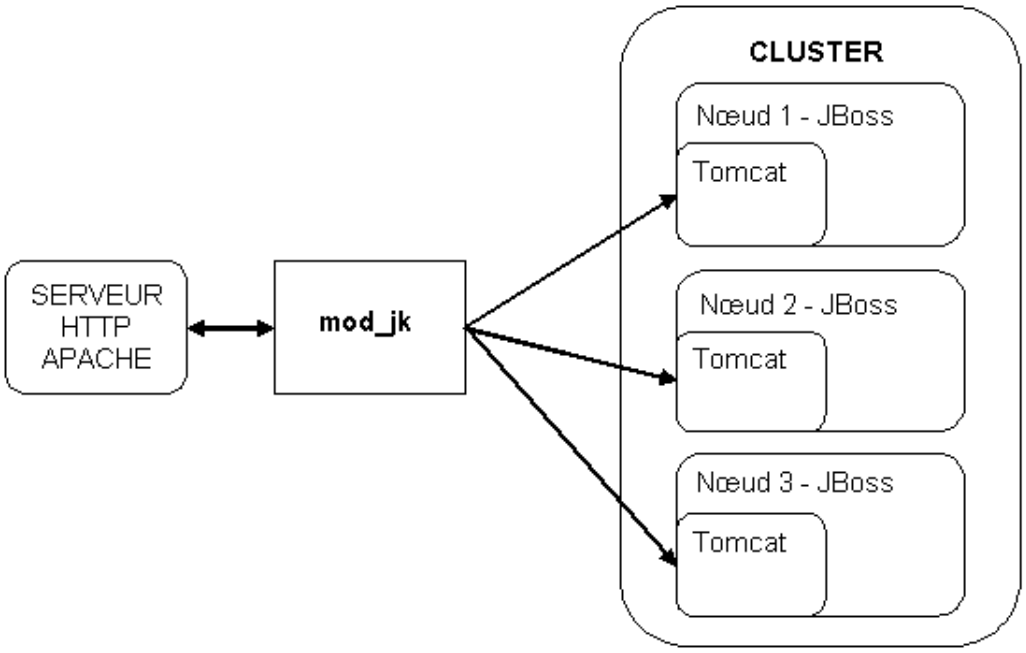
Service HTTP

La réplication des sessions permet de répliquer les états des sessions des clients entre les différents nœuds du cluster. Cette tâche est effectuée par JBoss lorsque le serveur est démarré dans la configuration `all`.

Par contre, la répartition de charge n'est pas prise en compte par JBoss et nécessite un logiciel, ou matériel supplémentaire.

Le répartiteur de charge trace les requêtes et, en fonction de la session entre le navigateur et le serveur, transmet la requête du navigateur vers le nœud qui gère cette session. Une fois qu'une session est créée les requêtes issues du navigateur seront toujours envoyées vers le nœud qui a créé cette session.

Nous allons mettre en place un répartiteur de charge en utilisant Apache comme serveur HTTP frontal, et le module mod_jk comme répartiteur.



1. Installation du module mod_jk

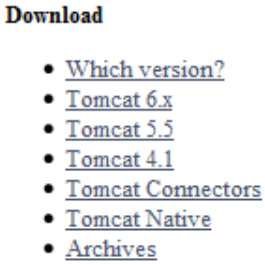
a. Pré-requis : serveur HTTP Apache

Il vous faut d'abord installer le serveur Apache. Ce serveur est, en général, installé par défaut sur les distributions Linux. Consultez la documentation de votre distribution pour retrouver les fichiers de configuration du serveur Apache. Sous Ubuntu, vous trouverez les fichiers de configuration dans le répertoire */etc/apache2*.

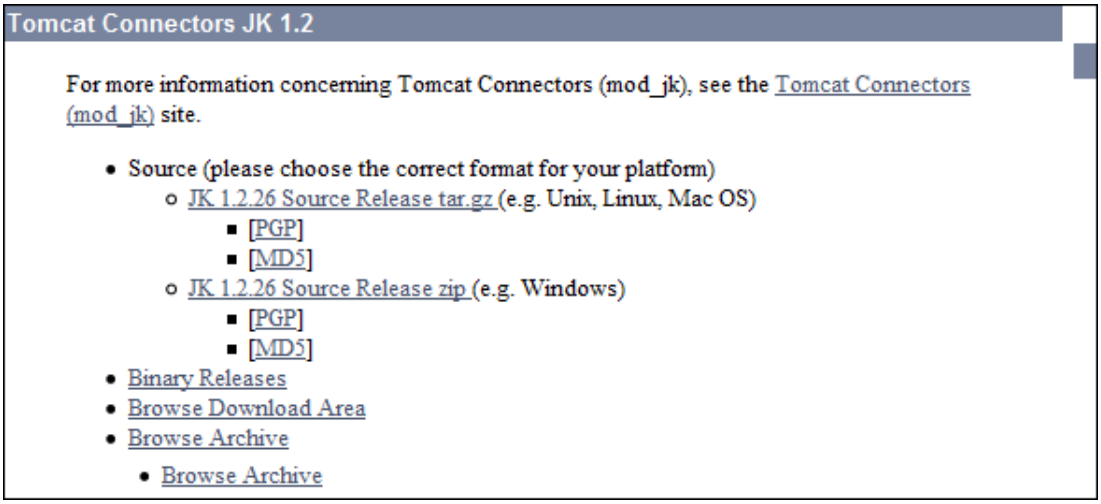
Sous Windows, vous pouvez télécharger le serveur Apache à l'URL www.apache.org, puis en suivant le lien **HTTP Server**, installer le serveur Apache2. Une fois l'installation terminée, vous trouverez les fichiers de configuration dans le répertoire *C:\Program Files\Apache Software Foundation\Apache2.2\conf*.

b. Téléchargement du module

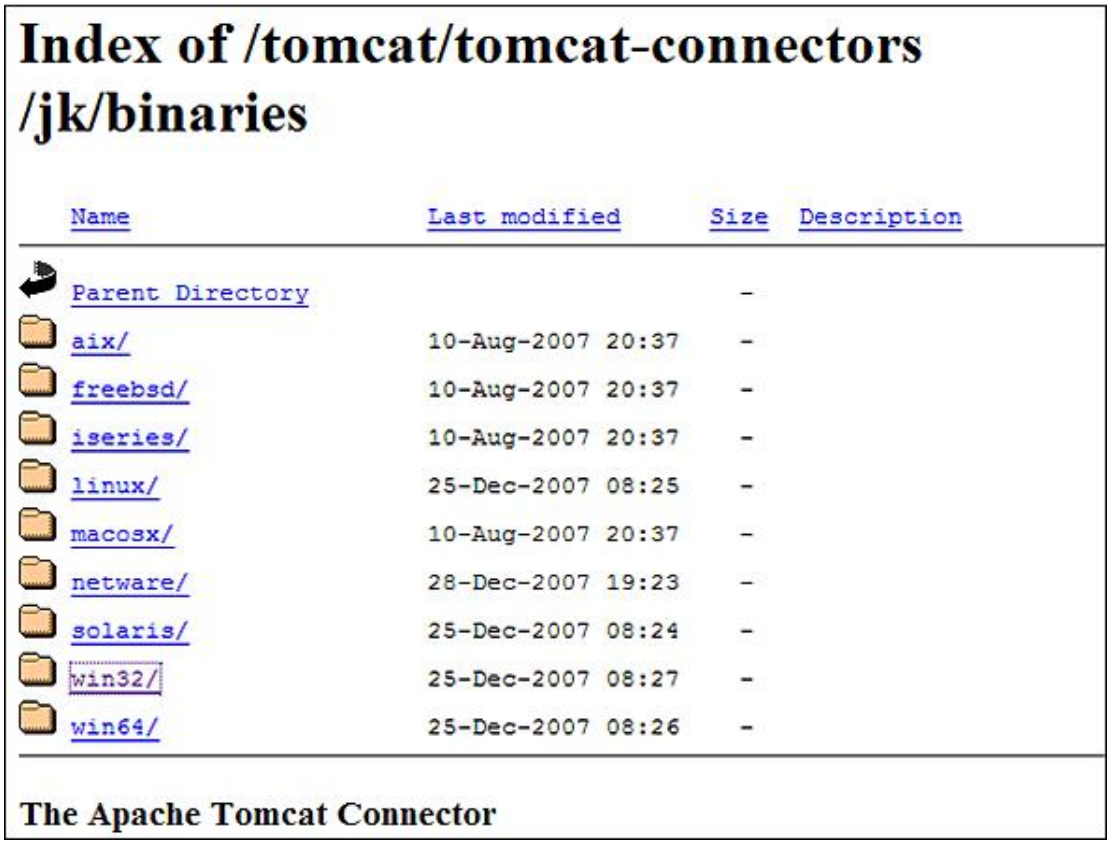
Le téléchargement du module se fait à l'URL tomcat.apache.org.



- Cliquez sur le lien **Tomcat Connectors**.



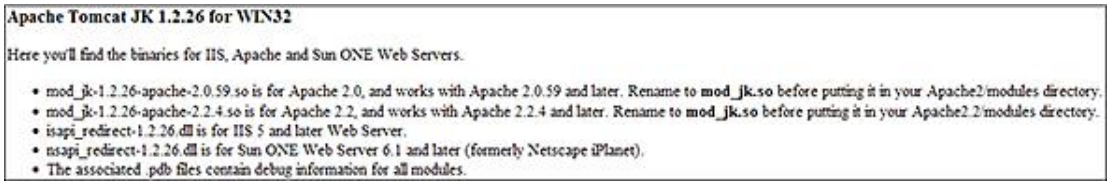
- Cliquez sur le lien **Binary Releases**.



- Cliquez sur le lien qui correspond à votre système d'exploitation. Selon le système d'exploitation choisi, divers

liens peuvent être présents. Suivez les liens de la version la plus récente pour arriver à la page de téléchargement du module. Le module est sous la forme d'un fichier SO pour LINUX comme pour Windows. Le nom du fichier est du type : *mod_jk-1.2.26-httpd-2.2.4.so*. Renommez ce fichier en *mod_jk.so*.

Attention : en fonction de la version d'Apache que vous employez, le fichier à télécharger est différent. En bas de la page de téléchargement, vous trouverez une explication décrivant les compatibilités de version.



Apache Tomcat JK 1.2.26 for WIN32

Here you'll find the binaries for IIS, Apache and Sun ONE Web Servers.

- mod_jk-1.2.26-apache-2.0.59.so is for Apache 2.0, and works with Apache 2.0.59 and later. Rename to **mod_jk.so** before putting it in your Apache2/modules directory.
- mod_jk-1.2.26-apache-2.2.4.so is for Apache 2.2, and works with Apache 2.2.4 and later. Rename to **mod_jk.so** before putting it in your Apache2.2/modules directory.
- isapi_redirect-1.2.26.dll is for IIS 5 and later Web Server.
- nsapi_redirect-1.2.26.dll is for Sun ONE Web Server 6.1 and later (formerly Netscape iPlanet).
- The associated .pdb files contain debug information for all modules.

- Copiez le fichier renommé dans le répertoire modules du répertoire d'installation d'Apache.

c. Configuration du module mod_jk

- Créez un fichier que nous appellerons *mod-jk.conf*, qui contient les lignes suivantes :

```
# Load mod_jk module
# Specify the filename of the mod_jk lib
LoadModule jk_module modules/mod_jk.so

# Where to find workers.properties
JkWorkersFile conf/workers.properties

# Where to put jk logs
JkLogFile logs/mod_jk.log

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

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


# JkOptions indicates to send SSK KEY SIZE
JkOptions +ForwardKeySize +ForwardURICompat -ForwardDirectories

# JkRequestLogFormat
JkRequestLogFormat "%w %V %T"

# Mount your applications
JkMount /jmx-console/* loadbalancer
JkMount /mon_appli_web/* loadbalancer

# Add shared memory
# This directive is present with 1.2.10 and
# later versions of mod_jk, and is needed for
# for load balancing to work properly
JkShmFile logs/jk.shm

# Add jkstatus for managing runtime data
<Location /jkstatus/>
JkMount status
Order deny,allow
Deny from all
Allow from 127.0.0.1
</Location>
```

- Modifiez ce fichier pour l'adapter au nom de votre application Web. Dans ce fichier exemple, les lignes :

```
JkMount /jmx-console/* loadbalancer
JkMount /mon_appli_web/* loadbalancer
```

indiquent au serveur Apache que les requêtes sur les URLs */jmx-console* et */mon_appli_web* seront transférées au répartiteur appelé ici `loadbalancer`.

Si vous travaillez sous LINUX et qu'il existe un répertoire du type */etc/apache2/mods-enabled*, vous pouvez directement copier le fichier*mod-jk.conf* dans ce répertoire.

Sinon, pour les autres distributions LINUX et pour Windows, ajoutez à la fin du fichier de configuration *httpd.conf* du serveur Apache la ligne suivante :

```
Include conf/mod-jk.conf
```

Le fichier *httpd.conf* se trouve dans le répertoire *conf* du serveur Apache. Sous LINUX, reportez-vous à la documentation de votre distribution.

d. Configuration des nœuds dans mod_jk

La configuration du module `mod_jk` lui-même est effectuée via un fichier nommé *workers.properties* qui doit être placé dans le répertoire *conf* du serveur Apache.

Vous pouvez changer ce répertoire par l'intermédiaire de la directive`JkWorkersFile` dans le fichier *mod-jk.conf*.

Dans ce fichier, nous allons déclarer les différents conteneurs de Servlet/JSP, et définir comment le suivi de session HTTP est effectué.

Le cluster est ici sur une même machine, ce qui explique les numéros de port, et les adresses IP utilisées.

```
# Define list of workers that will be used
# for mapping requests
worker.list=loadbalancer,status


# Define Node1
# modify the host as your host IP or DNS name.
worker.noeud1.port=8109
worker.noeud1.host=127.0.0.1
worker.noeud1.type=ajp13
worker.noeud1.lbfactor=1

# Define Node2
# modify the host as your host IP or DNS name.
worker.noeud2.port=8209
worker.noeud2.host= 127.0.0.1
worker.noeud2.type=ajp13
worker.noeud2.lbfactor=1

# Load-balancing behaviour
worker.loadbalancer.type=lb
worker.loadbalancer.balance_workers=noeud1,noeud2
worker.loadbalancer.sticky_session=1
#worker.list=loadbalancer
# Status worker for managing
```

Les éléments, qui traitent les requêtes, sont appelés`worker`. Une instance de Tomcat est un `worker`. Le répartiteur de charge est un autre type de `worker`.

Nous avons donc ici quatre workers :

- `noeud1` qui correspond à une instance de Tomcat ;
- `noeud2` qui correspond à une autre instance de Tomcat ;
- `loadbalancer` qui correspond au répartiteur de charge ;

- `status` qui est un worker virtuel. Il ne traite pas les requêtes HTTP, il permet de récupérer l'état et les informations de mod_jk.

Chaque worker est configuré par une suite de directives constituées de la manière suivante :

`worker.<nom_du_worker>.<directive>=<valeur>`

- `worker.list` : liste les workers qui seront instanciés par mod_jk, donc les workers correspondant aux instances de Tomcat ne doivent pas apparaître ici ;
- `type` : protocole utilisé, ici ajp13 pour les instances de Tomcat, et lb pour le répartiteur ;
- `host` : IP ou nom de domaine de l'instance de Tomcat;
- `port` : port d'écoute pour le protocole ;
- `lbfactor` : pondération du répartiteur de charge, plus ce nombre est élevé, plus le conteneur de servlet recevra de requêtes;
- `sticky-session` : permet de faire suivre la session au nœud qui l'a créée ;
- balance_workers : liste des instances de Tomcat utilisées par le répartiteur de charge.

e. Configuration de JBoss

Pour chaque instance de JBoss utilisée dans le cluster, nous devons maintenant configurer le fichier *server.xml* du service Web. Ce fichier se trouve dans le répertoire *deploy/jboss-web.deployer* de votre configuration serveur.

- Dans l'élément `<Engine>`, configurez le suivi des requêtes en précisant le nom du worker (un worker par nœud du cluster). La valeur de jvmRoute doit correspondre aux workers déclarés dans le fichier *workers.properties*.

```
<Engine name="jboss.web" defaultHost="localhost" jvmRoute="noeud1">
```

- Et dans le fichier *deploy/jboss-web.deployer/META-INF/jboss-service.xml*, positionnez l'attribut UseJK.

```
<attribute name="UseJK">true</attribute>
```

f. Test final

- Démarrez les différents nœuds de votre cluster. Lorsque tous les nœuds sont démarrés, démarrez le serveur Apache.
- Vous pouvez maintenant atteindre votre site web par l'URL http://localhost/mon_appli_web/

Notez que vous passez maintenant par le port 80, celui sur lequel écoute Apache, et non plus par un port d'écoute de JBoss, 8180 ou 8280 dans notre exemple.

JBoss RichFaces

Le projet RichFaces est une librairie de composants permettant l'intégration de comportements AJAX dans les vues, et ce, de manière très simple. Ce framework s'appuie sur les composants JSF *(Java Server Face)*. Le projet Ajax4Jsf a été intégré au projet RichFaces. RichFaces permet aussi de définir des thèmes pour personnaliser simplement le rendu des pages.

Le comportement AJAX est très simple à mettre en place. Il faut définir :

- un événement qui invoquera une requête AJAX ;
- la zone de la page qui sera synchronisée après le traitement de la requête AJAX par le serveur.

Nous allons présenter ici dans un exemple très simple, la mise en place du framework.

Avant de commencer, nous devons télécharger l'ensemble des bibliothèques nécessaires :

- la librairie RichFaces 3.2.x à l'URL http://www.jboss.org/jbossrichfaces/downloads/ où vous choisirez une archive avec les binaires : richfaces-ui-3.2.2.GA-bin.zip par exemple.
- les librairies "commons" du site Apache que vous pouvez télécharger à http://commons.apache.org/.

Décompressez les archives récupérées dans un répertoire spécifique. Pour mieux organiser son espace de travail l'auteur regroupe ainsi toutes les librairies nécessaires aux différents projets dans un répertoire *librairies_java*.

L'ensemble des librairies nécessaires au projet sont les suivantes :

- librairies JSF 1.2 (incluses dans la distribution JBoss)
  - *jsf-api.jar*
  - *jsf-impl.jar*
- librairies RichFaces
  - *richfaces-api-3.2.2.GA.jar*
  - *richfaces-impl-3.2.2.GA.jar*
  - *richfaces-ui-3.2.2.GA.jar*
- librairies Apache
  - *commons-beanutils.jar*
  - *commons-collections.jar* (incluse dans la distribution JBoss)
  - *commons-digester.jar*
  - *commons-logging.jar* (incluse dans la distribution JBoss)

Le projet présenté est dans l'archive des exemples, sous le nom "Chap 10 - RichFaces". Vous retrouverez dans le répertoire WEB-INF/lib, les librairies nécessaires, qui ne sont pas incluses dans la distribution JBoss.

Le fichier web.xml de l'application doit prendre en compte les librairies JSF et RichFaces.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
```

```
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    id="WebApp_ID" version="2.5">
    <display-name>Chap10 - RichFaces</display-name>

    <context-param>
        <param-name>org.richfaces.SKIN</param-name>
        <param-value>blueSky</param-value>
    </context-param>
    <filter>
        <display-name>RichFaces Filter</display-name>
        <filter-name>richfaces</filter-name>
        <filter-class>
            org.ajax4jsf.Filter
        </filter-class>
    </filter>
    <filter-mapping>
        <filter-name>richfaces</filter-name>
        <servlet-name>Faces Servlet</servlet-name>
        <dispatcher>REQUEST</dispatcher>
        <dispatcher>FORWARD</dispatcher>
        <dispatcher>INCLUDE</dispatcher>
    </filter-mapping>

    <!-- Faces Servlet -->
    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>
            javax.faces.webapp.FacesServlet
        </servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <!-- Faces Servlet Mapping -->
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>*.jsf</url-pattern>
    </servlet-mapping>
    <login-config>
        <auth-method>BASIC</auth-method>
    </login-config>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

Les pages JSP doivent au moins comprendre les librairies de balises personnalisées suivantes :

```
<%@taglib uri="http://richfaces.org/a4j" prefix="a4j" %>
<%@taglib uri="http://richfaces.org/rich" prefix="rich" %>
<%@taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
```

Notre page index.jsp met en place une zone de saisie. À chaque appui sur une touche dans cette zone, une autre zone sera remplie au fur et à mesure, par des appels AJAX.

```
<%@ page contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@taglib uri="http://richfaces.org/a4j" prefix="a4j" %>
<%@taglib uri="http://richfaces.org/rich" prefix="rich" %>
<%@taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@taglib uri="http://java.sun.com/jsf/core" prefix="f" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Chapitre 10 - RichFaces</title>
```

```
</head>
<body>
<f:view>
    <h:form>
        <rich:panel header="Exemple Echo">
            <h:inputText value="#{bean.text}">
            <a4j:support event="onkeyup" reRender="rep" />
            </h:inputText>
            <h:outputText value="#{bean.text}" id="rep"/>
        </rich:panel>
    </h:form>
</f:view>
</body>
</html>
```

La zone de saisie est de type `inputText`, et sa valeur est reliée à un Bean, que nous allons définir juste après.

La zone remplie est de type `outputText`, sa valeur est reliée au même Bean que la zone de saisie. Cette zone possède un attribut `id`, donc unique dans la page, qui vaut `rep`.

L'élément `<rich:panel ...>` permet de mettre en place une apparence de base pour notre page.

L'élément ajoute le comportement AJAX en définissant quel événement JavaScript est utilisé, et quel champ doit être mis à jour par l'attribut `reRender` qui contient la liste des `id` des champs.

Notre Bean est extrêmement simple :

```
public class BeanTexte
{
    private String text;

    public BeanTexte()
    {}

    public String getText()
    {
        return text;
    }

    public void setText(String text)
    {
        this.text = text;
    }
}
```

La prise en charge de ce Bean est effectuée par le fichier *faces-config.xml* qui se trouve dans le répertoire *WEB-INF*.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE faces-config PUBLIC "-//Sun Microsystems, Inc.//DTD
JavaServer Faces Config 1.1//EN"
    "http://java.sun.com/dtd/web-facesconfig_1_1.dtd">

<faces-config>
    <managed-bean>
        <managed-bean-name>bean</managed-bean-name>
        <managed-bean-class>
            fr.editions.eni.chap10.rf.BeanTexte
        </managed-bean-class>
        <managed-bean-scope>request</managed-bean-scope>
        <managed-property>
            <property-name>text</property-name>
            <value />
        </managed-property>
    </managed-bean>
</faces-config>
```

La propriété `text` du Bean est maintenant accessible par `#{bean.text}` dans la page JSP.

Une fois l'application déployée sous JBoss, elle est accessible par l'URL : http://localhost:8080/richf/index.**jsf**